Parts Inventory Requirements

Ahmed Al-Taiar

December 12, 2023

1 Overview

The purpose of Parts Inventory is to manage all the Arduino modules, tracking when, who, and how many of each part was taken and returned at any given time. Users will have to create an account with their details in order to be able to achieve this. Administrator users will have access to the database through a special menu and will be able to create, update, retrieve, and delete parts as they wish. Administrator users are also able to view a log of all transactions, both incoming and outgoing. The website also adapts to fit small screens for use on mobile devices.

2 Data Store

Most user, part, and transaction data will be stored in their respective files located in the server for quick access to data. The only exception is going to be the images associated with each part (if provided), which will be stored in a third-party image hosting service.

3 Account System

3.1 Sign-up

Users will be need to sign up for an account through a sign-up page on the website in order to access transaction privileges. Users will input their first and last name, email address, and password in order to create their account. The following information about that user will be stored in the data store:

- A unique identifier as a number
- Email address as text
- First name as text
- Last name as text
- Encrypted password as text, so that it is stored securely

- List of transactions they have done
- Their role (either “user” or “admin”)

3.1.1 Role

If an account’s email address is part of the list of administrator email addresses (upon sign-up), that account will have administrator privileges. The list is hidden from the public.

3.2 Login

Users with an account will be able to log in using their email address and password. If the password is correct for the account with the given email address, proceed with the login. Otherwise, reject the login. Users will stay logged in between sessions using the web browser’s cookies. Users are able to log out of their account at any time through a button in the navigation bar.

3.3 Resetting Password

If the user forgets their password, they will be able to reset it by entering their email address. If an account with that address exists, an email will be sent with a link that users can go to, so they can reset the password.

4 Parts

A part is defined as any unique item, whether it is an Arduino board, module, wire, resistor, LED, etc. Administrator users will be able to create, update, and delete parts.

4.1 Creating Parts

Parts can be created by entering the following information:

- The name of the part
- A description (optional)
- Amount available, with a minimum of zero
- An image (optional)

If provided, the image will be uploaded to a third party image hosting service. Otherwise, use a placeholder “no image provided” image. Using the aforementioned inputs, the following data will be stored in the data store:

- A unique identifier as a number
- The name as text

- The description as text, if no description was provided, “No description provided” is used instead
- The amount available
- A link to the uploaded image as text, if no image was provided, the link to the “no image provided” placeholder image is used instead
- The date and time the part was created at

4.2 Updating Parts

Parts can be updated manually by retrieving the following information and then edit:

- The name of the part
- Description
- Amount available
- The image

The updated fields will then be overwritten in the data store, the unchanged fields stay the same.

4.3 Deleting Parts

Parts can be deleted off the data store. If the deleted part has an uploaded image, the image will be deleted off the third party image hosting service.

4.4 Retrieving Parts

4.4.1 Regular Users

Using the parts stored in the data store, a catalog will be created with pagination, having eight parts per page. Each part that is displayed will include:

- The name
- Description that cuts off if it is too long
- Availability, or “out of stock” if the part is not available for withdrawal
- Image
- A button to add the part to the user’s list of parts they want to withdraw from the inventory

Users will be able to sort the catalog by any one of:

- Unique identifier (default)
- Name
- Description
- Availability
- Date of creation

The catalog can then be ordered ascending or descending. Users will also be able to search for parts by the name, the catalog will filter the parts so that the search query is present in the results' names. When the user clicks or taps on any of the parts in the catalog, they will be redirected to a page dedicated to that part containing the same information mentioned above, but they will be able to read the full description. In that page, users will also be able to add more than one of that part at a time using a number field where the maximum is the amount available, and the minimum is zero.

4.4.2 Administrators

Administrators will be able to everything a regular user can do. They will also have access to a page where they can see all the parts in a compact list, including some information hidden from regular users, being:

- Unique identifier
- Date of creation

Each element of the list will also include three buttons:

1. View the part in a new page in detail (including the hidden information mentioned above)
2. Edit the part
3. Delete the part

Access to the administrator view of parts will be through a menu in the navigation bar visible to administrators only.

5 Basket & Transactions

All users, including administrators, will be able to create, update, and retrieve transactions.

5.1 Creating Transactions

Users will be able to add parts to a virtual basket that is stored in the web browser's cookies, it will consist of a list of parts, as well as the quantity of each part. The quantity should be limited to the availability of the part and should be greater than zero. If the part is out of stock, prevent the user from adding the part to the basket. The user is able to check out the basket by going to the basket page and press a button. When that button is pressed, save the transaction in the data store, storing:

- The user's unique identifier
- The basket, having the parts and the quantity of each part

When the transaction is complete, empty the basket and decrement the stock of each part by the specified quantity in the data store.

5.2 Retrieving Transactions

5.2.1 Regular Users

Users can view a list of all the transactions they have performed, they can see a log of all the parts that have been taken out and returned, and when the transaction was done. Users can filter the transaction list by whether the transaction was to return or to take out.

5.2.2 Administrators

Similar to that of a regular user, but administrators can see the transactions of all users.

5.3 Updating Transactions

Users can return a transaction that is marked as "out" in the transaction list. The transaction will then be marked as "in", meaning that the parts in the transaction have been returned. When the transaction is marked as "in", increment the available stock of each part by the quantity in the data store. Of course, administrators should ensure the parts are being returned in real life before users mark a transaction as "in".

6 Miscellaneous

6.1 Theme

Users will be able to switch between a light and dark theme across the website. The light theme will have a nearly white background and dark text. The dark theme will have a very dark navy blue background with nearly white text. The selected theme will be saved in the web browser's cookies so the choice persists between sessions.

6.2 Navigation Bar

The website will feature a navigation bar in every page, allowing the user to traverse the pages. The navigation bar will consist of buttons and menus when opened on a laptop or desktop computer. When the website is opened on a small tablet or mobile phone, the navigation bar will transform to be a list of buttons that will appear in a side pane, which is visible with the press of one button.

Parts Inventory Specification

Ahmed Al-Taiar

November 24, 2023

1 Technologies

- Redwood Web Framework
  - React
  - GraphQL
  - Prisma
  - TypeScript
  - Jest
  - Storybook
- TailwindCSS Utility Classes
- DaisyUI Component Library

2 Database

The website's data will be hosted on a local PostgreSQL server with the exception for the part images, those will be stored on Filestack.

2.1 Schemas

2.1.1 Part

Field	Required	Type	Default Value
ID	Yes	Int	Automatically increment
Name	Yes	String	
Description	No	String	No description provided
Available stock	Yes	Int	0
Image URL	Yes	String	Local placeholder image path
Date of creation	Yes	Date	Now
Transaction ID Relation	No	Int	

2.1.2 User

Field	Required	Type	Default Value
ID	Yes	Int	Automatically increment
First name	Yes	String	
Last name	Yes	String	
Email address	Yes	String	
Encrypted password	Yes	String	
Password salt	Yes	String	
Password reset token	No	String	
Password reset token expiry date	No	Date	
Role	Yes	String	user
Transactions	Yes	Transaction[]	

2.1.3 Transaction

Field	Required	Type	Default Value
ID	Yes	Int	Automatically increment
Transaction Date	Yes	Date	Now
User ID relation	Yes	Int	
Type	Yes	TransactionType	
Parts	Yes	Json[]	

TransactionType is an enum of values “in” or “out”.

3 Account System

Use Redwood's built in authentication system then generate pages (using the Redwood command line interface) for signing up, logging in, forgot password, etc. Change the CSS stylesheets to use DaisyUI components instead, for uniformity across pages.

3.1 Sign Up Page

Modify the sign-up page to include text fields for entering the user's first and last name, then include the values in the payload when creating a new account.

4 Part Management

Generate the pages needed to create, retrieve, update, and delete parts using Redwood's scaffold generator. Pass in the Part schema as input. Change the CSS stylesheets to use DaisyUI components instead, for uniformity across pages.

4.1 Part Form

Modify the form for used for creating or updating parts to include Filestack's image uploader component instead of a text field for inputting the image URL, use the outputted image URL from the uploader component in the payload for creating/updating parts.

4.2 Deleting Parts

After a part is deleted, automatically delete the image from Filestack as well.

4.3 Retrieving Parts

When retrieving a part's image through the image URL, use Filestack's transformation parameters to resize the image to the appropriate size in order to conserve bandwidth. Also create a new GraphQL query, for retrieving parts based on a page and filter, that takes in the following parameters:

Parameter	**Required**	**Type**	**Default Value**
Page	Yes	Int	
Sort Method	Yes	SortMethod	
Sort Order	Yes	SortOrder	
Search query	No	String	Nothing

SortMethod is an enum of any one of these values:

- ID

- Name
- Description
- Available stock
- Date of creation

SortOrder is an enum of any one of these values:

- Ascending
- Descending

5 Basket

5.1 Adding To Basket

When a user adds a part to the basket, save the part as a Json string and also include the quantity. If the same part is already in the basket, increment the quantity instead. Then save the entire basket as a string in the browser's local storage.

5.2 Clearing Basket

Delete the string from the browser's local storage.

5.3 Deleting From Basket

Delete the part & quantity element based on the index.

5.4 Editing Basket

The quantity of each element can be modified directly in the basket page.

6 Transactions

6.1 Creating Transactions

When a user checks out their order, create a transaction in the database. Save the basket and its contents, the user's ID, and date. When the transaction is created, decrement the available stock of each specified part by the quantity. Then clear the basket from the browser's local storage.

6.2 Retrieving Transactions

Administrator accounts get access to a second transactions page. It is the exact same as the normal transactions' page with the exception that it lists all transactions instead of just the user's.

6.3 Returning

Create a new GraphQL mutation which takes the transaction's ID and user's ID, and then marks a transaction's TransactionType to "in", then increment the available stock of each part by the quantity. This mutation can be accessed in the user's transactions page when they want to return parts.

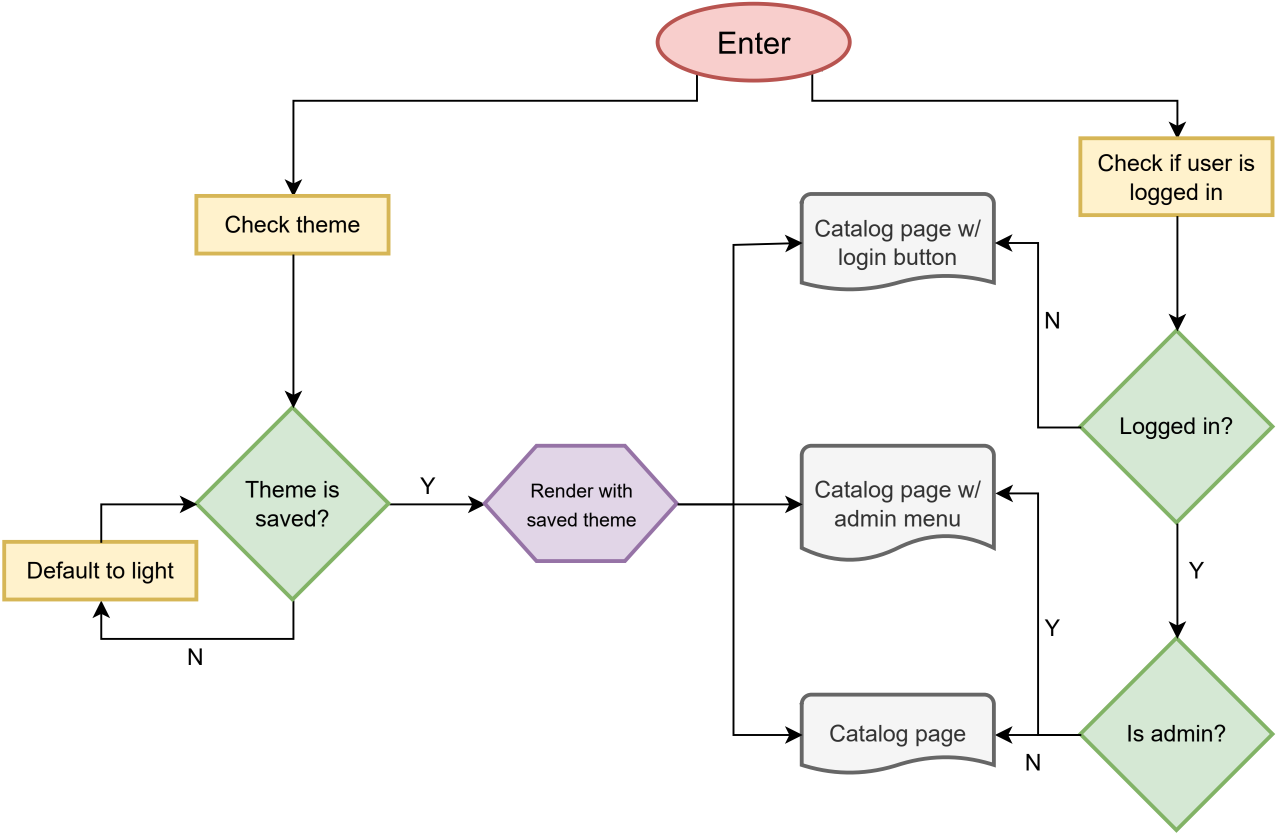
7 Miscellaneous

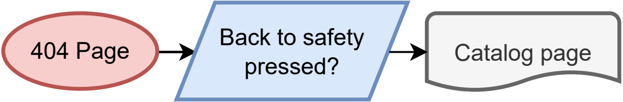
7.1 Theme

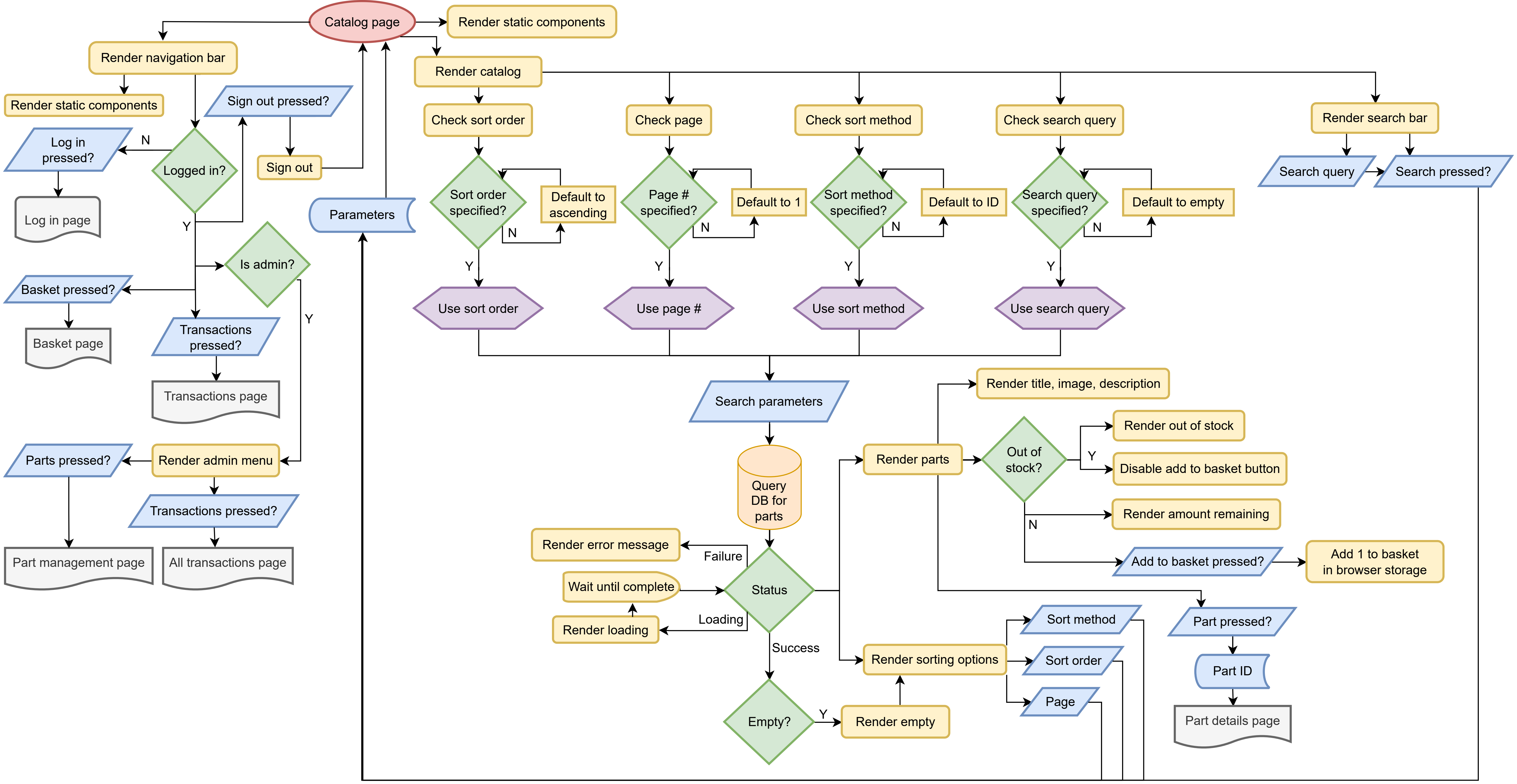
Use DaisyUI's themes and create a theme toggle component in the navigation bar that switches between light and dark theme. The theme should be saved in the browser's local storage so the theme persists across sessions.

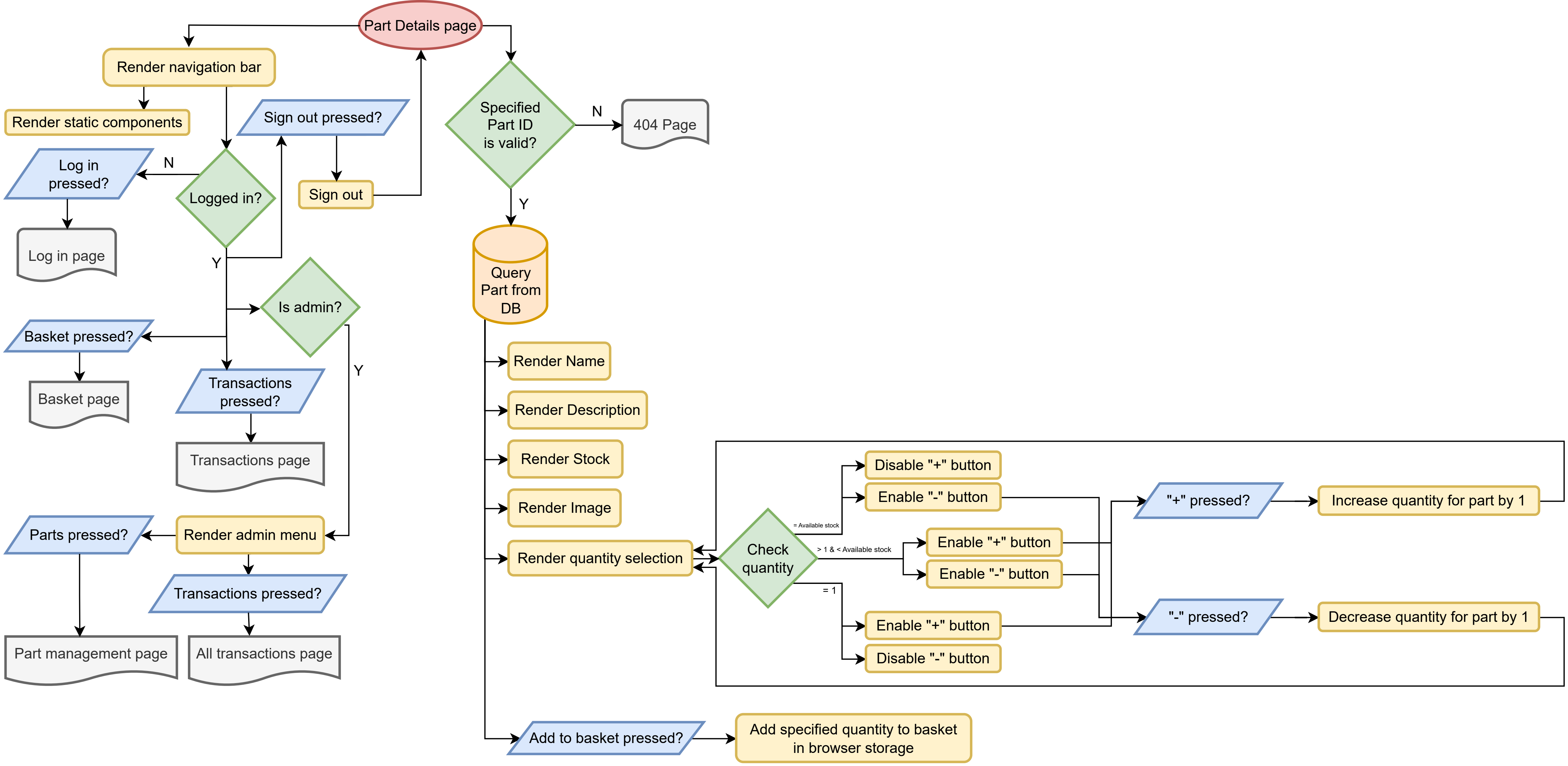
7.2 Navigation Bar

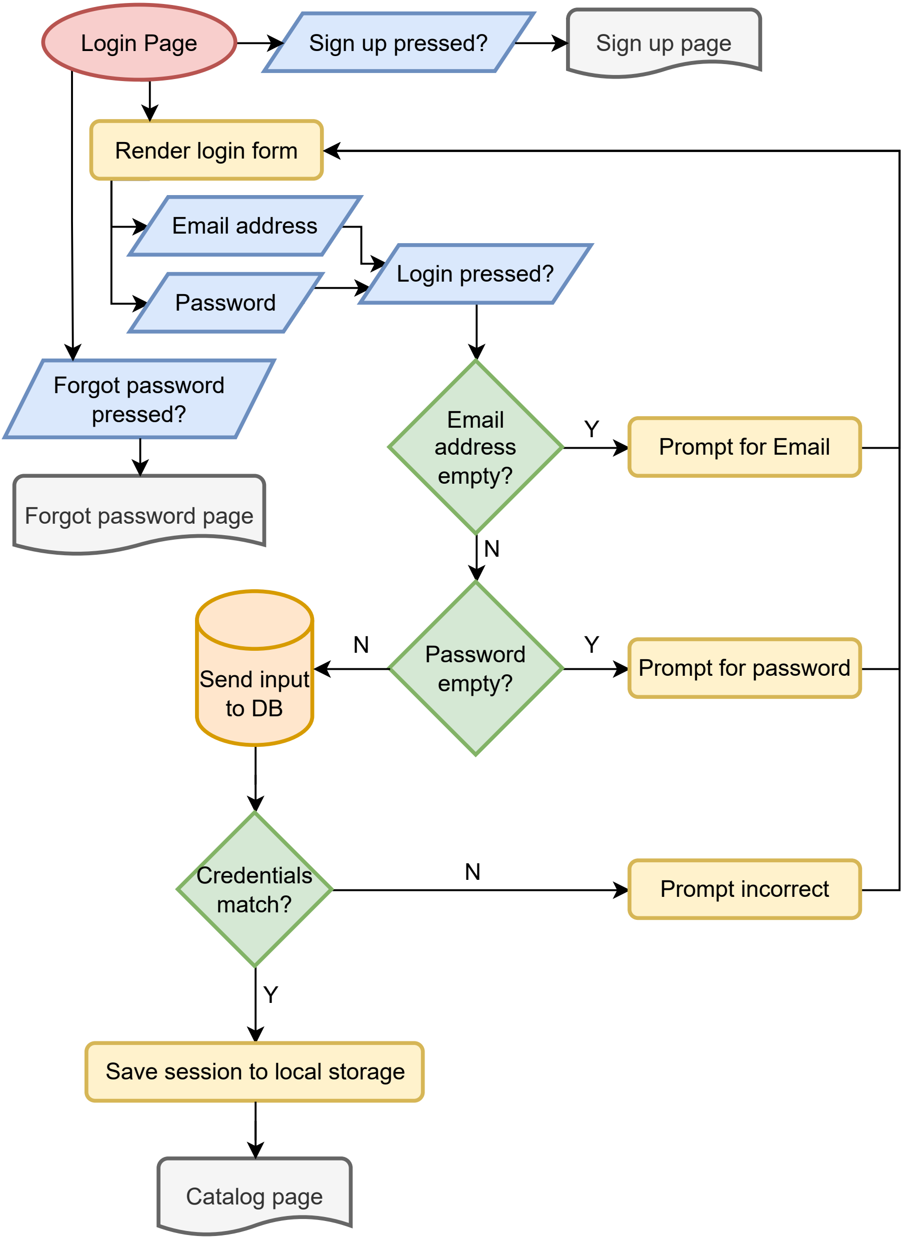
On mobile, hide the links on the navigation bar and put them in a list, that is shown with the press of one button on the navigation bar. It should also have the theme toggle component.

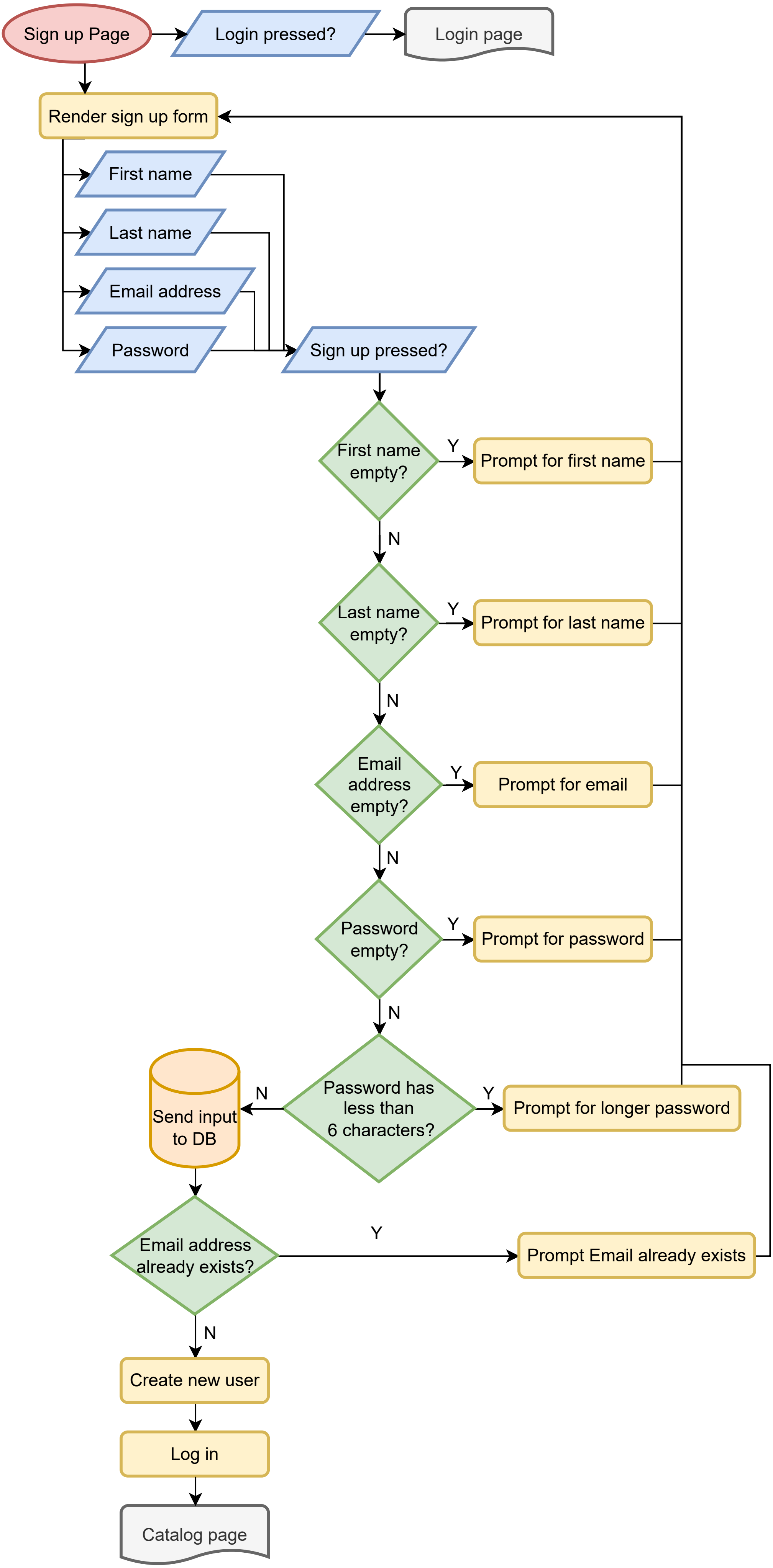

Enter
Check theme
Check if user is logged in
Catalog page w/ login button
N
Logged in?
Theme is saved?
Y
Render with saved theme
Catalog page w/ admin menu
Default to light
Y
N
Y
Catalog page
N
Is admin?

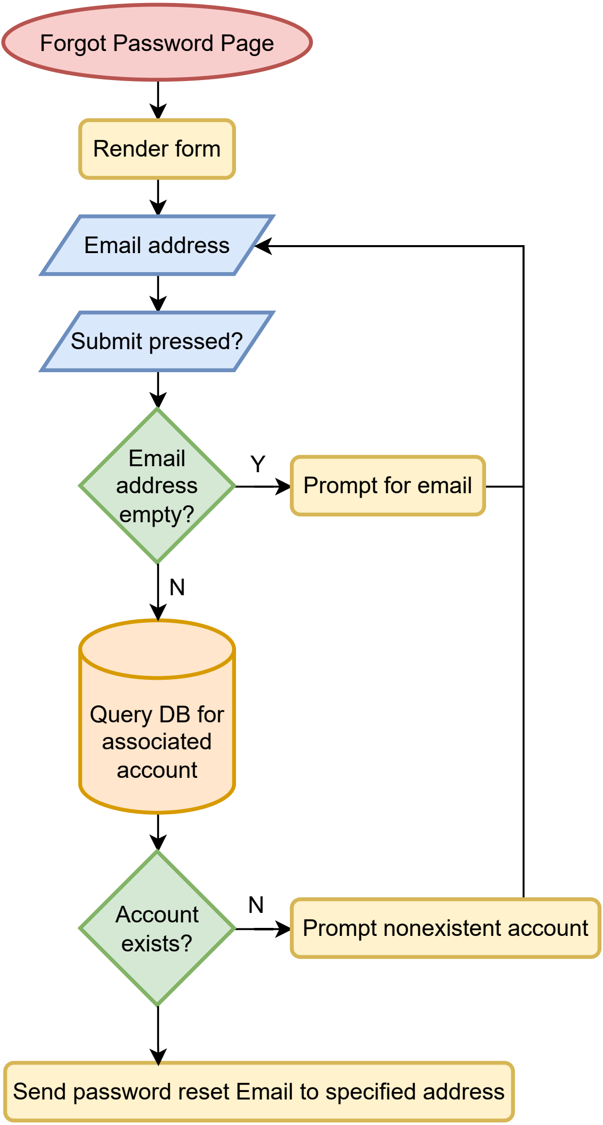

404 Page
Back to safety pressed?
Catalog page

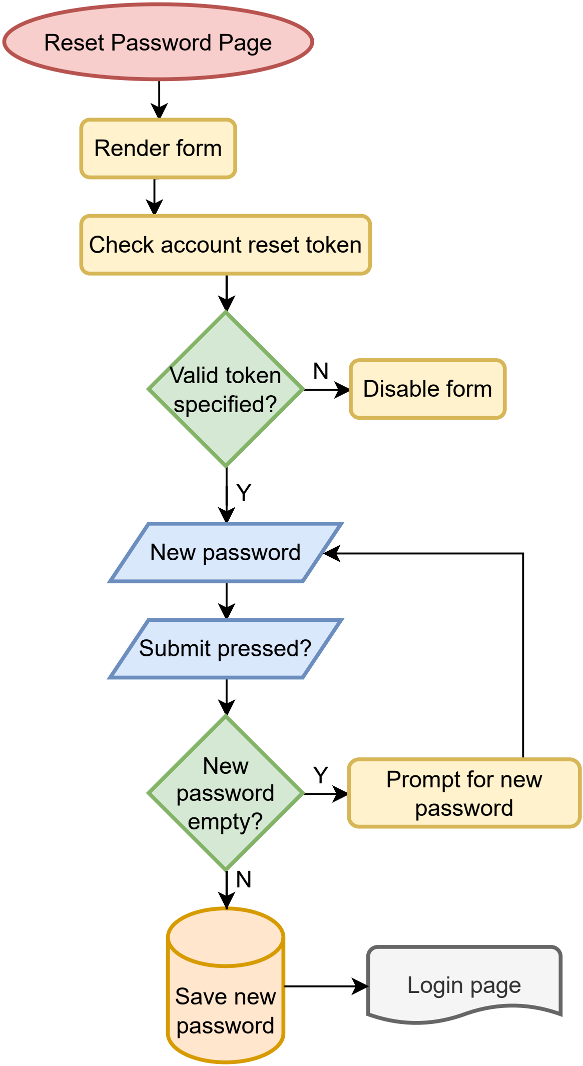

Catalog page
Render static components
Render navigation bar
Render static components
Sign out pressed?
Log in pressed?
N
Logged in?
Sign out
Log in page
Y
Is admin?
Basket pressed?
Basket page
Transactions pressed?
Transactions page
Y
Parts pressed?
Render admin menu
Transactions pressed?
Part management page
All transactions page
Parameters
Render catalog
Check sort order
Sort order specified?
Default to ascending
N
Y
Use sort order
Check page
Page # specified?
Default to 1
N
Y
Use page #
Check sort method
Sort method specified?
Default to ID
N
Y
Use sort method
Check search query
Search query specified?
Default to empty
N
Y
Use search query
Render search bar
Search query
Search pressed?
Search parameters
Query DB for parts
Render title, image, description
Render parts
Out of stock?
Y
Render out of stock
Disable add to basket button
N
Render amount remaining
Add to basket pressed?
Add 1 to basket in browser storage
Render error message
Failure
Wait until complete
Status
Render loading
Loading
Success
Empty?
Y
Render empty
Render sorting options
Sort method
Sort order
Page
Part pressed?
Part ID
Part details page

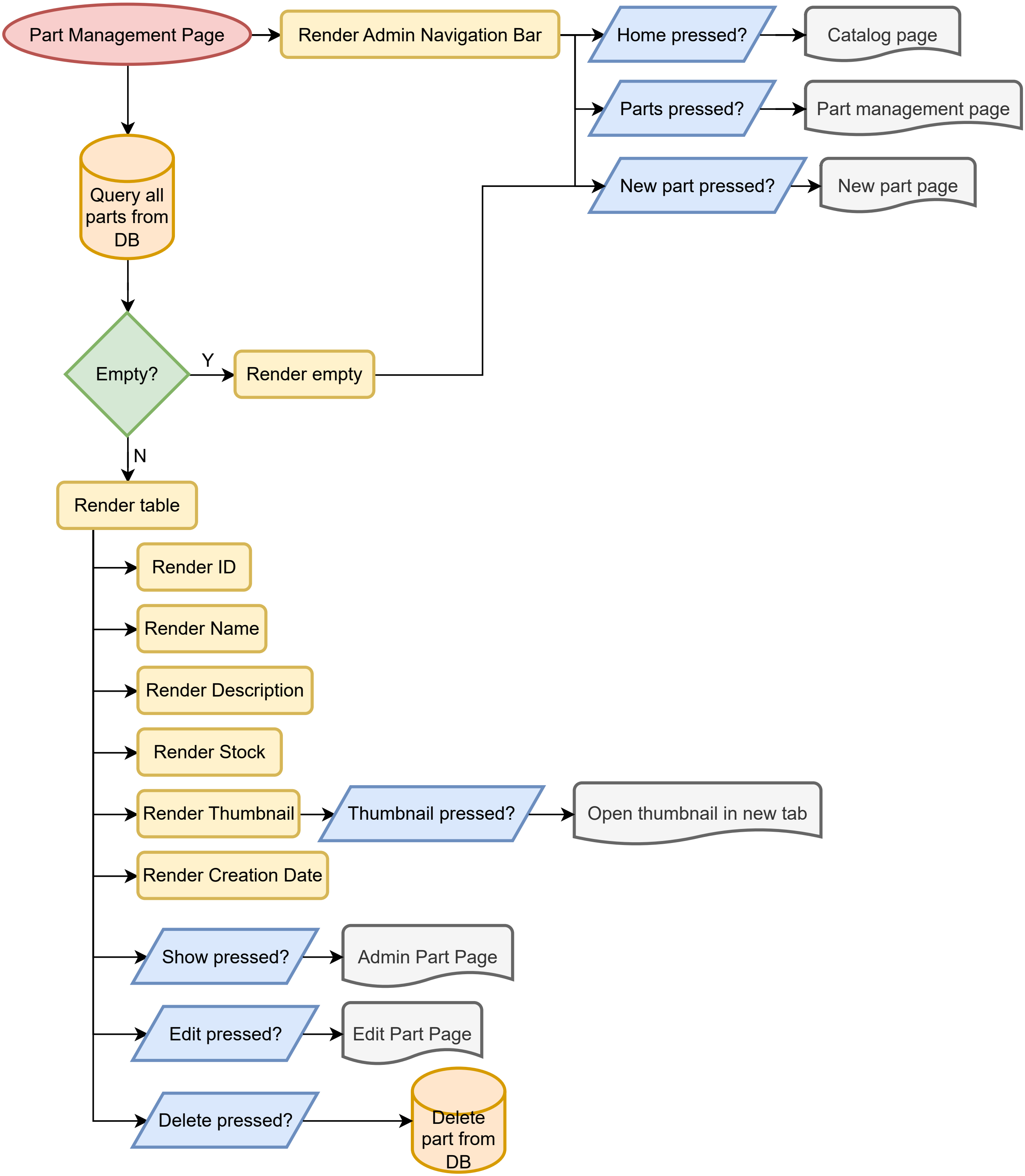

Part Details page
Render navigation bar
Render static components
Sign out pressed?
Log in pressed?
N
Logged in?
Sign out
Log in page
Y
Is admin?
Basket pressed?
Transactions pressed?
Y
Basket page
Transactions page
Parts pressed?
Render admin menu
Transactions pressed?
Part management page
All transactions page
Specified Part ID is valid?
N
404 Page
Y
Query Part from DB
Render Name
Render Description
Render Stock
Render Image
Render quantity selection
Check quantity
= Available stock
> 1 & < Available stock
= 1
Disable "+" button
Enable "-" button
Enable "+" button
Enable "-" button
Enable "+" button
Disable "-" button
"+" pressed?
Increase quantity for part by 1
"-" pressed?
Decrease quantity for part by 1
Add to basket pressed?
Add specified quantity to basket in browser storage

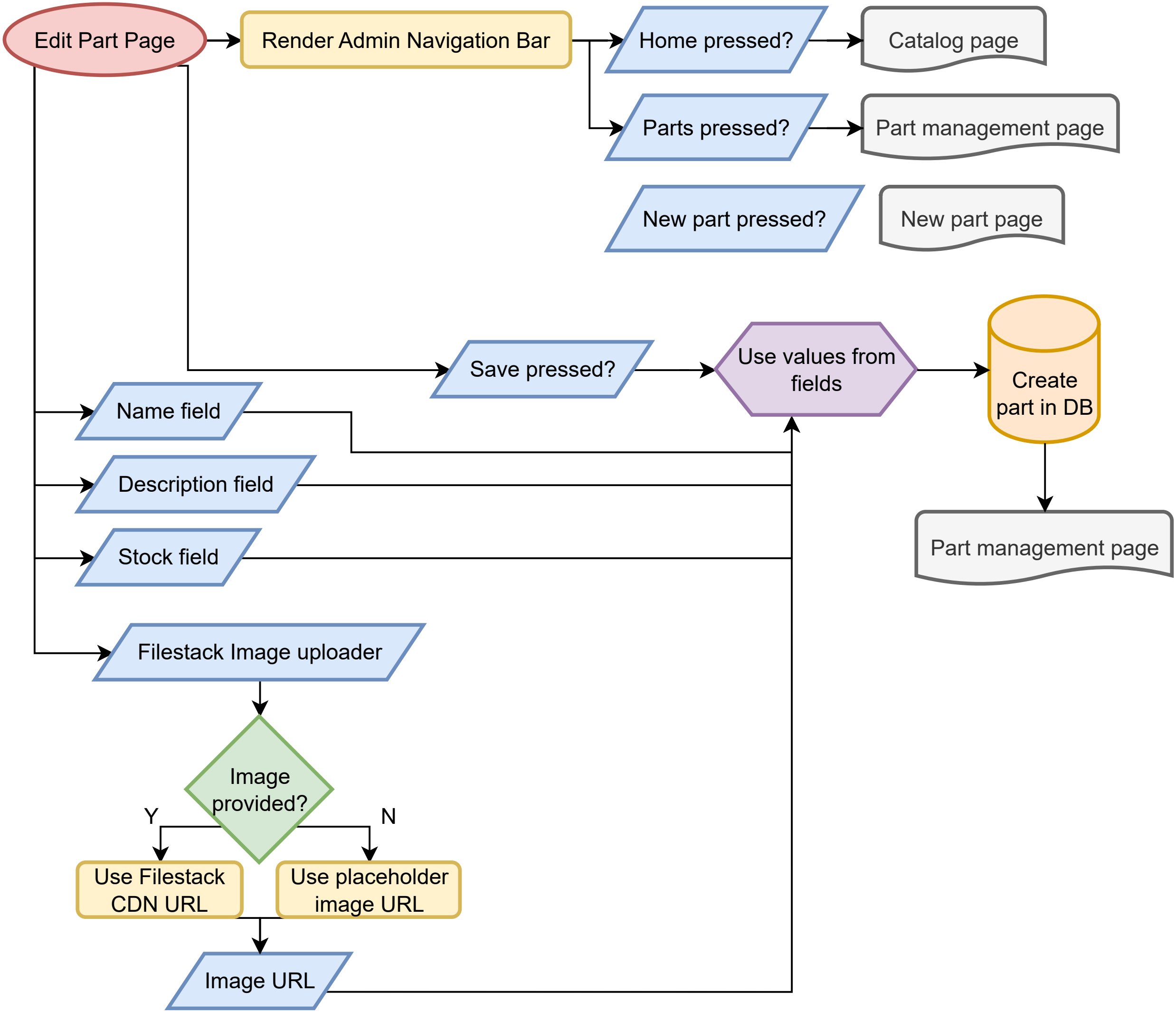

Login Page
Sign up pressed?
Sign up page
Render login form
Email address
Password
Login pressed?
Forgot password pressed?
Forgot password page
Email address empty?
Y
Prompt for Email
N
Password empty?
Y
Prompt for password
N
Send input to DB
Credentials match?
N
Prompt incorrect
Y
Save session to local storage
Catalog page

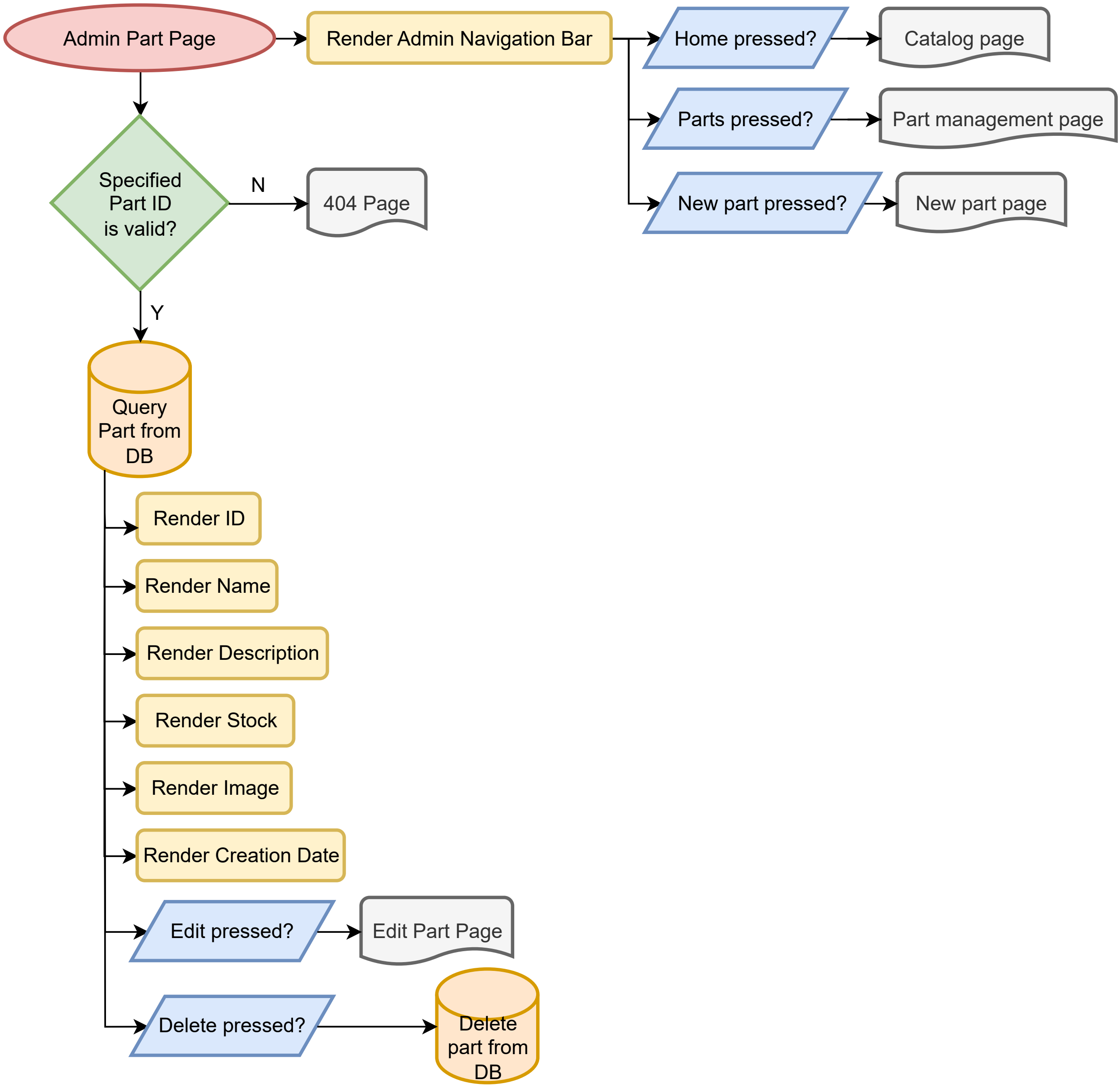

Sign up Page
Login pressed?
Login page
Render sign up form
First name
Last name
Email address
Password
Sign up pressed?
First name empty?
Y
Prompt for first name
N
Last name empty?
Y
Prompt for last name
N
Email address empty?
Y
Prompt for email
N
Password empty?
Y
Prompt for password
N
Password has less than 6 characters?
Y
Prompt for longer password
N
Send input to DB
Email address already exists?
Y
Prompt Email already exists
N
Create new user
Log in
Catalog page

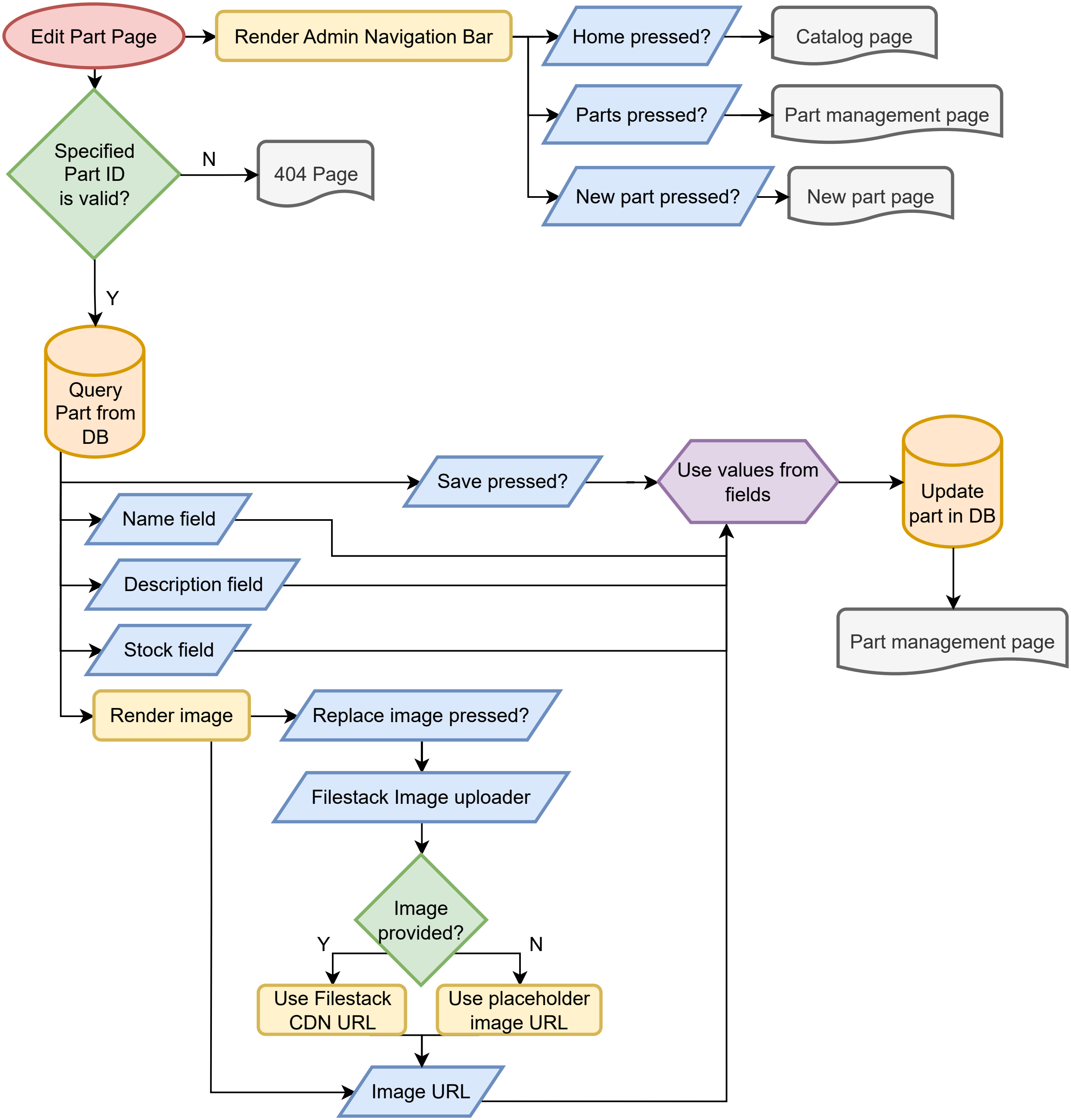

Forgot Password Page
Render form
Email address
Submit pressed?
Email address empty?
Y
Prompt for email
N
Query DB for associated account
Account exists?
N
Prompt nonexistent account
Send password reset Email to specified address

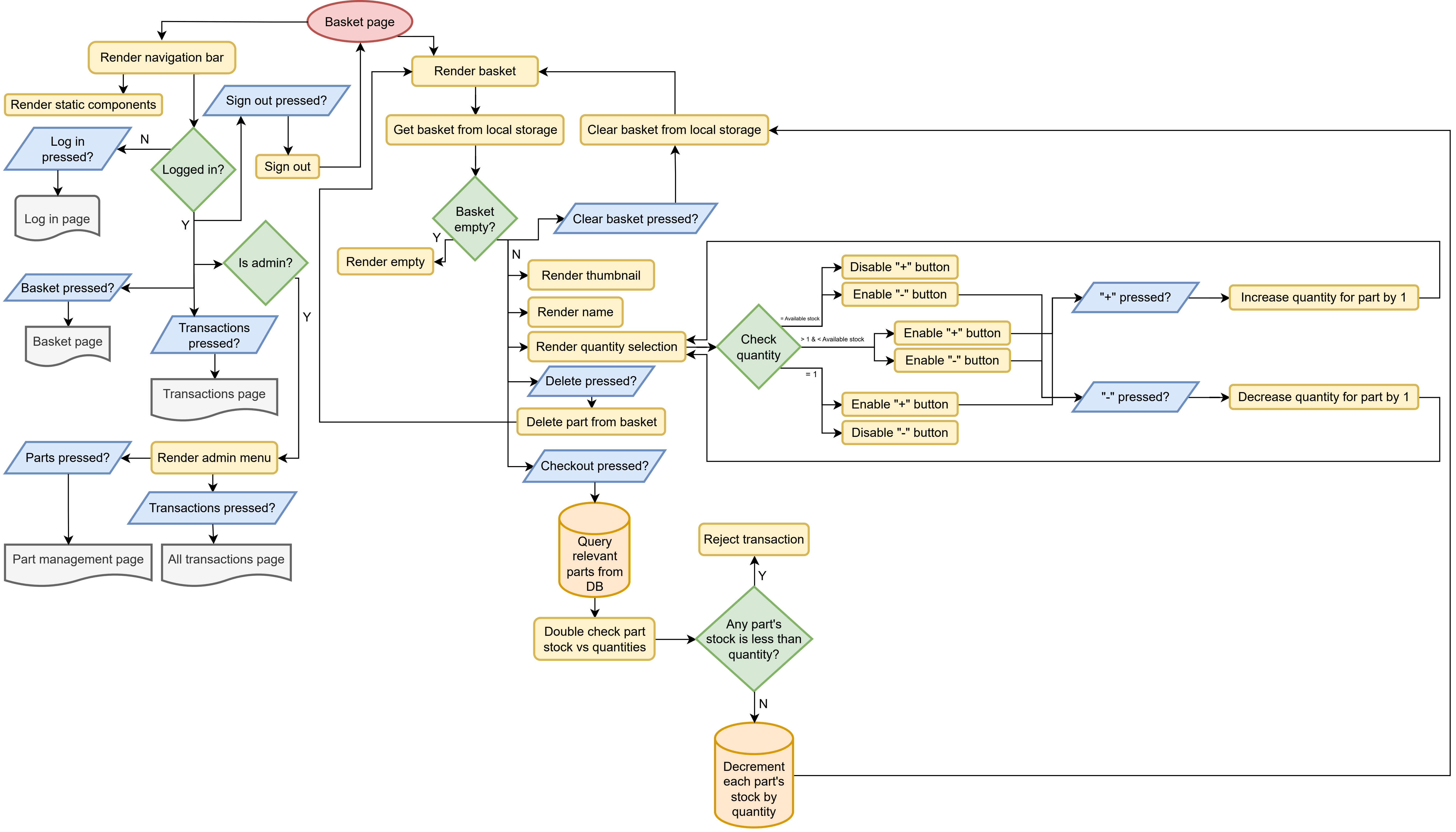

Reset Password Page
Render form
Check account reset token
Valid token specified?
N
Disable form
Y
New password
Submit pressed?
New password empty?
Y
Prompt for new password
N
Save new password
Login page

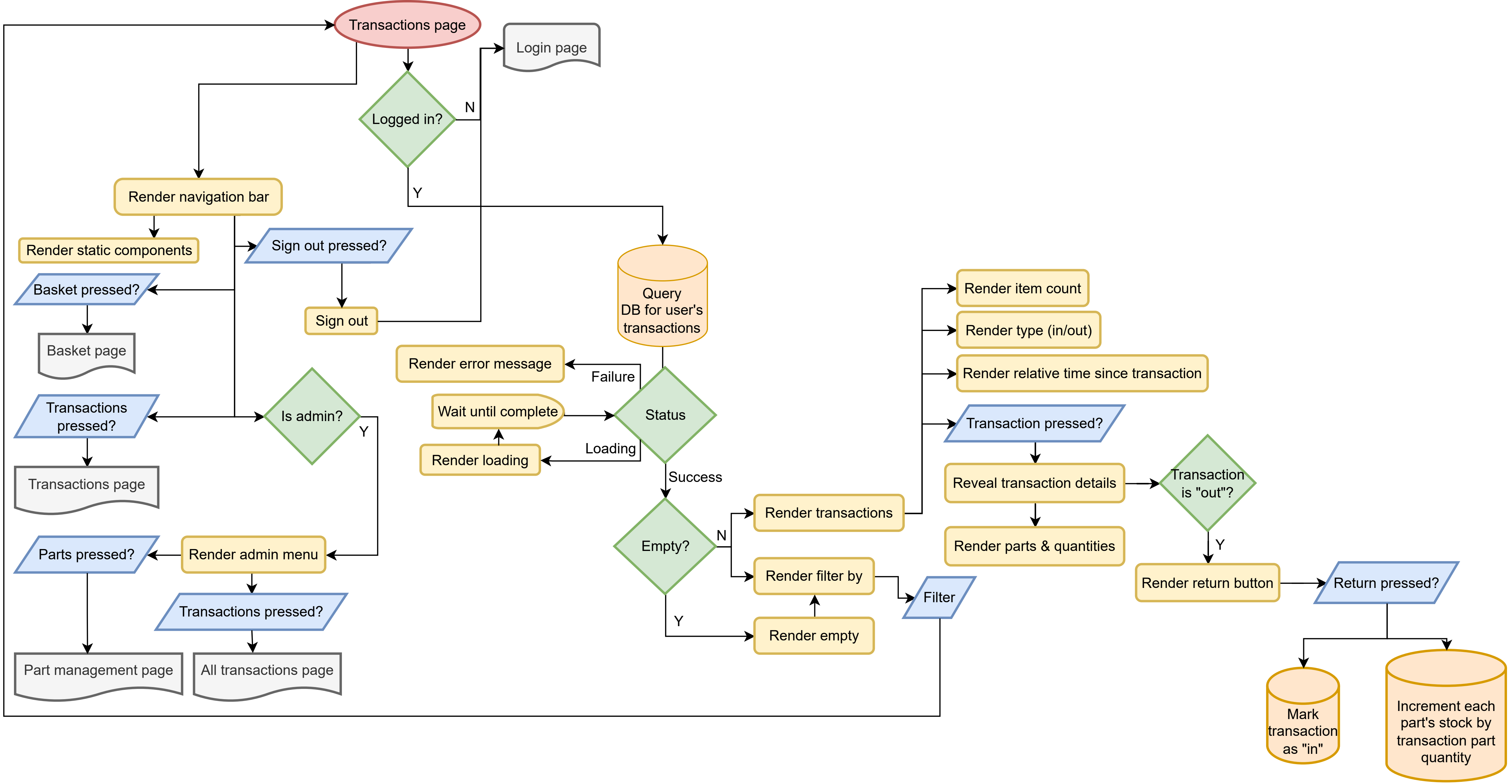

Part Management Page
Render Admin Navigation Bar
Home pressed?
Catalog page
Parts pressed?
Part management page
New part pressed?
New part page
Query all parts from DB
Empty?
Y
Render empty
N
Render table
Render ID
Render Name
Render Description
Render Stock
Render Thumbnail
Thumbnail pressed?
Open thumbnail in new tab
Render Creation Date
Show pressed?
Admin Part Page
Edit pressed?
Edit Part Page
Delete pressed?
Delete part from DB

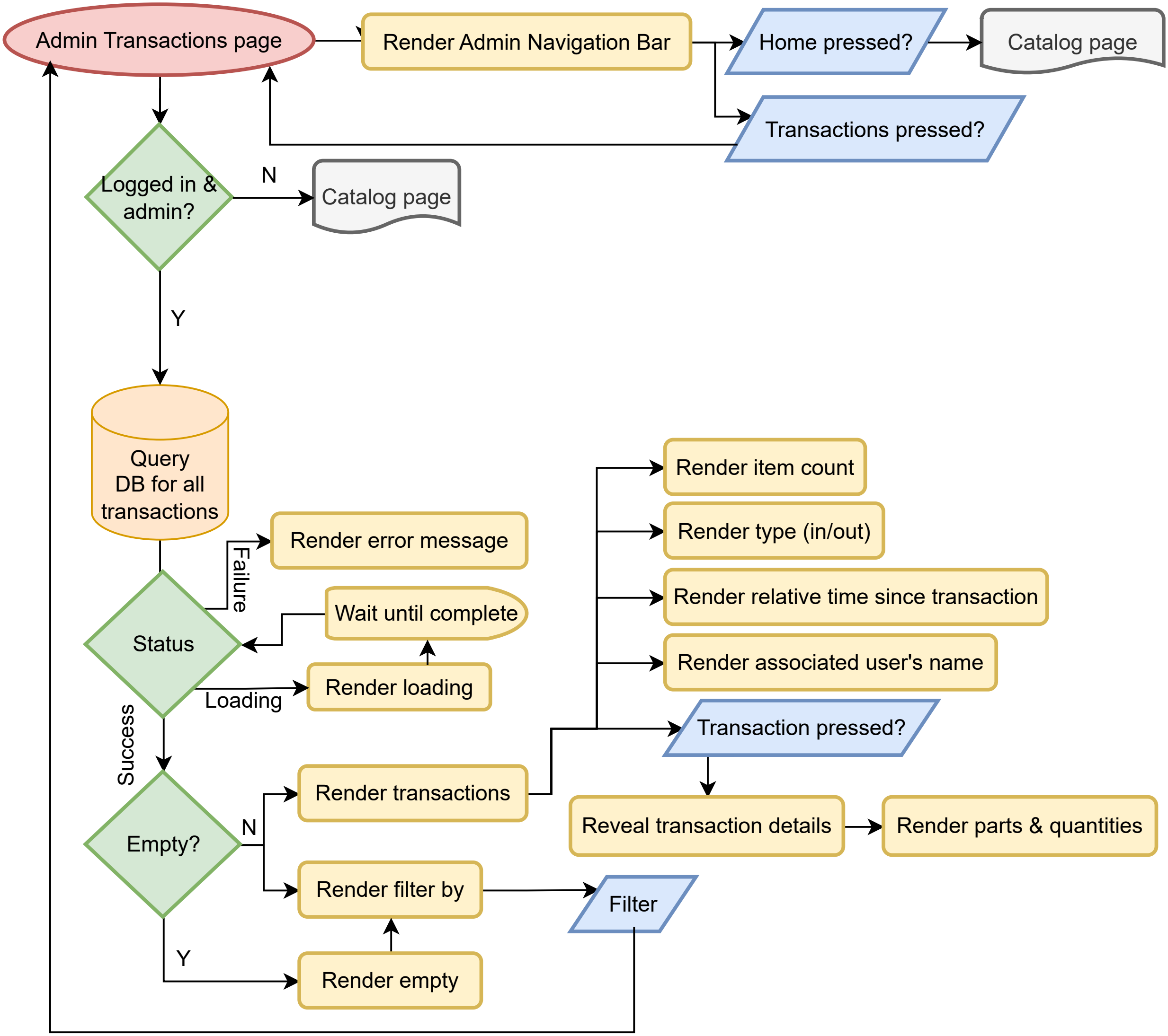

Edit Part Page
Render Admin Navigation Bar
Home pressed?
Catalog page
Parts pressed?
Part management page
New part pressed?
New part page
Save pressed?
Use values from fields
Create part in DB
Part management page
Name field
Description field
Stock field
Filestack Image uploader
Image provided?
Y
N
Use Filestack CDN URL
Use placeholder image URL
Image URL


Admin Part Page
Render Admin Navigation Bar
Home pressed?
Catalog page
Parts pressed?
Part management page
New part pressed?
New part page
Specified Part ID is valid?
N
404 Page
Y
Query Part from DB
Render ID
Render Name
Render Description
Render Stock
Render Image
Render Creation Date
Edit pressed?
Edit Part Page
Delete pressed?
Delete part from DB


Edit Part Page
Render Admin Navigation Bar
Home pressed?
Catalog page
Parts pressed?
Part management page
New part pressed?
New part page
Specified Part ID is valid?
N
404 Page
Y
Query Part from DB
Save pressed?
Use values from fields
Update part in DB
Part management page
Name field
Description field
Stock field
Render image
Replace image pressed?
Filestack Image uploader
Image provided?
Y
N
Use Filestack CDN URL
Use placeholder image URL
Image URL


Basket page
Render navigation bar
Render static components
Sign out pressed?
Log in pressed?
N
Logged in?
Sign out
Log in page
Y
Is admin?
Basket pressed?
Y
Basket page
Transactions pressed?
Transactions page
Parts pressed?
Render admin menu
Transactions pressed?
Part management page
All transactions page
Render basket
Get basket from local storage
Clear basket from local storage
Basket empty?
Clear basket pressed?
Y
Render empty
N
Render thumbnail
Render name
Render quantity selection
Delete pressed?
Delete part from basket
Checkout pressed?
Query relevant parts from DB
Double check part stock vs quantities
Reject transaction
Y
Any part's stock is less than quantity?
N
Decrement each part's stock by quantity
Check quantity
= Available stock
> 1 & < Available stock
= 1
Disable "+" button
Enable "-" button
Enable "+" button
Enable "-" button
Enable "+" button
Disable "-" button
"+" pressed?
"-" pressed?
Increase quantity for part by 1
Decrease quantity for part by 1


Transactions page
Login page
Logged in?
N
Y
Render navigation bar
Render static components
Sign out pressed?
Basket pressed?
Sign out
Basket page
Query DB for user's transactions
Render error message
Failure
Wait until complete
Status
Loading
Render loading
Success
Transactions pressed?
Is admin?
Y
Transactions page
Empty?
N
Y
Render transactions
Render filter by
Render empty
Filter
Parts pressed?
Render admin menu
Transactions pressed?
Part management page
All transactions page
Render item count
Render type (in/out)
Render relative time since transaction
Transaction pressed?
Reveal transaction details
Render parts & quantities
Transaction is "out"?
Y
Render return button
Return pressed?
Mark transaction as "in"
Increment each part's stock by transaction part quantity


Admin Transactions page
Render Admin Navigation Bar
Home pressed?
Catalog page
Transactions pressed?
Logged in & admin?
N
Catalog page
Y
Query DB for all transactions
Render error message
Failure
Wait until complete
Status
Render loading
Loading
Success
Render item count
Render type (in/out)
Render relative time since transaction
Render associated user's name
Transaction pressed?
Render transactions
Reveal transaction details
Render parts & quantities
Empty?
N
Render filter by
Filter
Y
Render empty

Parts Inventory IPO Charts

Ahmed Al-Taiar

December 8, 2023

1 Enter

1.1 Check If User Is Logged In

Input	Process	Output

	Use the destructured `boolean isAuthenticated` variable within Redwood's `useAuth` React hook.	**IF** `isAuthenticated`, `true`
		ELSE, `false`

1.2 Check Theme & Default To Light

Input	Process	Output

	Check the stored `theme` value in the browser's local storage.
	IF `theme` does not exist
	⇒ Set `theme` to "light"
		IF `theme` is "dark", `true`
		ELSE, `false`

2 404

Input	Process	Output
"Back to safety" button		Go to catalog page

3 Navigation Bar

3.1 Render Navigation Bar

Input	Process	Output
Child elements	**IF** screen width is small ⇒ Render menu with all buttons **ELSE** ⇒ Render all buttons on navigation bar Render child elements	Page with navigation bar at the top

3.2 Render All Buttons

Input	Process	Output
1. User is logged in (`boolean`) 2. User is admin (`false` if user is not logged in)	Render "Parts Inventory" button Render "basket" button **IF** user is logged in ⇒ Render "sign out" button ⇒ Render "transactions" button **IF** user is admin ⇒ Render "parts" button ⇒Render (admin) "transactions" button	Buttons

3.3 Sign Out

Input	Process	Output
"Sign out" button	Use the destructured `logOut` function within Redwood's `useAuth` React hook.	Same page the user was on originally

4 Admin Navigation Bar

4.1 Render Admin Navigation Bar

Input	Process	Output
Child elements	Render all buttons on navigation bar Render child elements	Page with admin navigation bar at the top

4.2 Render All Buttons

Input	Process	Output
1. Button label (`string`) 2. Button destination URL (`string`) 3. Title label (`string`) 4. Title destination URL (`string`)	Render home button that goes to the catalog page (5.2) Render title button with specified title and title destination URL Render button with specified button destination URL	Buttons

5 Catalog Page

Input	Process	Output
Any one of: • Redirect • Enter (1) • "Parts Inventory" button (3.2)	Render navigation bar (3.1) Render static components Render catalog	Catalog page

5.1 Static Components

Input	Process	Output
	Render “Parts Inventory” Render “Only take what you need”	Static components

5.2 Catalog

Input	Process	Output
1. Page (default: 1) 2. Sort Method (default: ID) 3. Sort Order (default: ascending) 4. Search query (optional)	Query database for parts with inputted search parameters Render a “loading” placeholder while the database is being queried **IF** no parts found ⇒Render “empty” **ELSE IF** an error occurred ⇒Render “error” **ELSE** ⇒**FOR** each part ⇒⇒Render details ⇒⇒Render “add to basket” button (15.1.2, quantity of 1) ⇒⇒**IF** part’s stock is 0 ⇒⇒⇒Disable button ⇒⇒Redirect to the part’s page if the part is pressed Render dropdown menus to change the search parameters, if any change, repeat 5.2	Catalog

6 Forgot Password Page

6.1 Render Form

Input	Process	Output
	Render email address input Render submit button	Form

6.2 Submit

Input	Process	Output
1. "Submit" button 2. Email address (`string`)	Query database to find account associated with the inputted email address	**IF** Account exists with email address ⇒Send email to account's address with password reset link **ELSE** ⇒Return to form, account doesn't exist (6.1)

7 Login Page

7.1 Render Form

Input	Process	Output
	Render email address input Render password input Render "forgot password" link Render "sign up" link Render "login" button	Form

7.2 Log In

Input	Process	Output
1. "Login" button 2. Email address (`string`) 3. Password (`string`)	Use the destructured `logIn` function within Redwood's `useAuth` React hook **IF** credentials match ⇒ Save session in browser's cookies, so user is still logged in **ELSE** ⇒Reject login	**IF** login successful ⇒ Go to the page the user was previously on

7.3 Forgot Password

Input	Process	Output
"Forgot password" link		Go to forgot password page (6)

7.4 Sign Up

Input	Process	Output
"Sign up" link		Go to sign up page (8)

8 Sign Up Page

8.1 Render Form

Input	Process	Output
	Render first name input Render last name input Render email address input Render password input Render "login" link Render "sign up" button	Form

8.2 Sign Up

Input	Process	Output
1. “Sign up” button 2. First name (`string`) 3. Last name (`string`) 4. Email address (`string`) 5. Password (`string`)	Use the destructured `signUp` function within Redwood’s `useAuth` React hook Save session in browser’s cookies, so user is immediately logged in	Go to the page the user was previously on

8.3 Log In

Input	Process	Output
“Log in” link		Go to login page (7)

9 Reset Password Page

9.1 Render Form

Input	Process	Output
Reset token (`string`)	Render new password input Render “submit” button Match reset token with associated account **IF** reset token is not valid ⇒Disable “submit” button	Form

9.2 Reset Password

Input	Process	Output
1. "Submit" button 2. New password (`string`)	Use the destructured `resetPassword` function within Redwood's `useAuth` React hook	Go to login page (7)

10 Part Management Page

Input	Process	Output
Either: - Redirect - "Parts" button	Render admin navigation bar (4.1) Render part management	Part management page

10.1 Part Management

Input	Process	Output
Database query for parts	Render a “loading” placeholder while the database is being queried **IF** no parts found ⇒Render “empty” **ELSE IF** an error occurred ⇒Render “error” **ELSE** ⇒**FOR** each part ⇒⇒Render part ID, name, description, stock, thumbnail, and creation date in a table row ⇒⇒Render “show” button, that goes to its part details page (11) ⇒⇒Render “edit” button that goes to its edit page (12) ⇒⇒Render “delete” button that deletes the part (10.1.1)	Parts list

10.1.1 Delete Part

Input	Process	Output
- Part (`object`) - “Delete” button	Confirm if the admin wants to delete the part **IF** admin confirms yes ⇒ Delete part	Refresh parts list

11 Admin Part Page

Input	Process	Output
Part “show” button	Render admin navigation bar (4.1) Render admin part	Admin part page

11.1 Admin Part

Input	Process	Output
Part (`object`)	Render complete part ID, name, description, stock, image, and creation date Render “edit” button that goes to its edit page (12) Render “delete button” that deletes the part (10.1.1)	Part details

12 Edit Part Page

Input	Process	Output
Part “edit” button	Render admin navigation bar (4.1) Render edit form	Edit part page

12.1 Edit Part

Input	Process	Output
Part (`object`)	Render part name input (initial value: part's name) Render part description input (initial value: part's description) Render stock number input (initial value: part's stock, must be ≥ 0) Render current image with "replace image" button Render "save" button	Edit form

12.2 Replace Image

Input	Process	Output
"Replace image" button	Render Filestack file upload	**IF** new image file is uploaded ⇒Filestack CDN URL of the new image **ELSE** ⇒Current image URL

12.3 Save

Input	Process	Output
• Part (`object`) • New name (`string`) • New description (`string`) • New stock (`int`) • New image URL (`string`)	Overwrite the part's fields with the new values Update part in database	Go to part management page

13 New Part Page

Input	Process	Output
“New part” button	Render admin navigation bar (4.1) Render new part form	New part page

13.1 New Part Form

Input	Process	Output
	Render part name input Render part description input Render stock number input (initial value: 0, must be ≥ 0) Render Filestack file upload Render “save” button	Form

13.2 Save

Input	Process	Output
• Name (`string`) • Description (`string`) • Stock (`int`) • Image URL (`string`)	Create new part in database	Go to part management page

14 Part Details Page

Input	Process	Output
Catalog part click	Render navigation bar (3.1) Render part details	Part details page

14.1 Part Details

Input	Process	Output
Part (`object`)	Render complete part name, description, stock, and image Render quantity selector (range from 1 to part's stock) Render "add to basket" button with specified quantity (15.1.2)	Part details

15 Basket Page

Input	Process	Output
"Basket" button	Render navigation bar (3.1) Render basket	Basket page

15.1 Basket

Input	Process	Output
Basket entry in browser's local storage (`string` or `null`)	**IF** basket is `null` **OR** empty ⇒Render "empty" **ELSE** ⇒Parse basket to `object[]` ⇒**FOR** each part in basket ⇒⇒Render thumbnail & title ⇒⇒Render quantity selector (range from 1 to part's stock) ⇒⇒Render "delete" button ⇒Render "clear basket" button ⇒Render "checkout" button	Basket

15.1.1 Delete From Basket

Input	Process	Output
• Basket entry in browser's local storage (`string` or `null`) • Basket part • "Delete' button'	Parse basket to an `object[]`, or create a new array if `null` or empty Remove part & quantity from basket array Convert basket array back to string Overwrite basket in browser's local storage with new basket string	New basket

15.1.2 Add To Basket

Input	Process	Output
• Basket entry in browser's local storage (`string` or `null`) • Part (`object`) • Quantity (`int`) • "Add to basket" button	Parse basket to an `object[]`, or create a new array if `null` or empty Add part & quantity to basket array Convert basket array back to string Overwrite basket in browser's local storage with new basket string	New basket

15.1.3 Clear Basket

Input	Process	Output
"Clear basket" button	Delete `basket` entry from browser's local storage	

15.1.4 Checkout

Input	Process	Output
• User is logged in (`boolean`) • Parsed basket entry from browser's local storage (`object[]`) • "Checkout" button	**IF** user is not logged in ⇒Reject transaction **ELSE** ⇒Get user's ID ⇒**FOR** each part in `basket` ⇒⇒Get up-to-date part details from database ⇒⇒**IF** specified quantity for part > part's stock ⇒⇒⇒Reject transaction ⇒⇒**ELSE** ⇒⇒⇒Decrement part's stock by quantity ⇒Create new transaction in database, with basket, user's ID, and "out" as the transaction type ⇒Delete `basket` entry from browser's local storage	Rejection message or transaction

16 Transactions Page

Input	Process	Output
"Transactions" button	Render navigation bar (3.1) Render transactions	Transactions page

16.1 Transactions

Input	Process	Output
• User ID (`int`) • Filter by (`enum`)	Query database for transactions linked to specified user ID and filter Render “filter by” selection **IF** transactions is empty ⇒Render “empty” **ELSE** ⇒**FOR** each transaction ⇒⇒Render item count, relative time, and type (“in”/“out”) ⇒⇒On press, reveal quantity and title of each part ⇒⇒**IF** type is “out” ⇒⇒⇒Render “return” button in revealed portion (16.2)	Transactions

16.2 Return Transaction

Input	Process	Output
• Transaction (`object`) • “Return” button	**FOR** each part & quantity in transaction ⇒Increment part’s stock in database by quantity Update transaction’s type to “in”	Transaction

17 Admin Transactions Page

Input	Process	Output
(Admin) “Transactions” button	Render admin navigation bar (4.1) Render admin transactions	Admin transactions page

17.1 Admin Transactions

Input	Process	Output
Filter by (`enum`)	Query database for all transactions and filter by specified filter Render “filter by” selection **IF** transactions is empty ⇒Render “empty” **ELSE** ⇒**FOR** each transaction ⇒⇒Render item count, relative time, type (“in”/“out”), and user’s full name ⇒⇒On press, reveal quantity and title of each part	Admin transactions

https://github.com/cy1der/arduino-parts-inventory

Root Directory

.editorconfig
.env
.env.defaults
.env.example
.git
.gitignore
.nvmrc
.redwood
.vscode
.yarn
.yarnrc.yml
README.md
api/
documents/
embold.yaml
graphql.config.js
jest.config.js
node_modules/
package.json
prettier.config.js
redwood.toml
scripts/
web/
yarn.lock

api/ tree

```
api
├── db
│   ├── dev.db
│   ├── dev.db-journal
│   ├── migrations
│   │   ├── 20231107150728_move_to_postgres
│   │   │   └── migration.sql
│   │   ├── 20231107152332_transaction
│   │   │   └── migration.sql
│   │   ├── 20231107163803_transaction
│   │   │   └── migration.sql
│   │   ├── 20231107165858_del
│   │   │   └── migration.sql
│   │   ├── 20231107224945_transaction
│   │   │   └── migration.sql
│   │   └── migration_lock.toml
│   └── schema.prisma
├── dist
│   ├── directives
│   │   ├── requireAuth
│   │   │   ├── requireAuth.js
│   │   │   └── requireAuth.js.map
│   │   └── skipAuth
│   │       ├── skipAuth.js
│   │       └── skipAuth.js.map
│   ├── functions
│   │   ├── auth.js
│   │   ├── auth.js.map
│   │   ├── graphql.js
│   │   └── graphql.js.map
│   ├── graphql
│   │   ├── parts.sdl.js
│   │   ├── parts.sdl.js.map
│   │   ├── transactions.sdl.js
│   │   ├── transactions.sdl.js.map
│   │   ├── users.sdl.js
│   │   └── users.sdl.js.map
│   ├── lib
│   │   ├── auth.js
│   │   ├── auth.js.map
│   │   ├── db.js
│   │   ├── db.js.map
│   │   ├── email.js
│   │   ├── email.js.map
│   │   ├── logger.js
│   │   └── logger.js.map
│   └── services
│       ├── parts
│       │   ├── parts.js
│       │   └── parts.js.map
│       ├── transactions
│       │   ├── transactions.js
│       │   └── transactions.js.map
│       └── users
│           ├── users.js
│           └── users.js.map
├── jest.config.js
├── package.json
├── server.config.js
├── src
│   ├── directives
│   │   ├── requireAuth
│   │   │   ├── requireAuth.test.ts
│   │   │   └── requireAuth.ts
│   │   └── skipAuth
│   │       ├── skipAuth.test.ts
│   │       └── skipAuth.ts
│   ├── functions
│   │   ├── auth.ts
│   │   └── graphql.ts
│   ├── graphql
│   │   ├── parts.sdl.ts
│   │   ├── transactions.sdl.ts
│   │   └── users.sdl.ts
│   ├── lib
│   │   ├── auth.ts
│   │   ├── db.ts
│   │   ├── email.ts
│   │   └── logger.ts
│   └── services
│       ├── parts
│       │   ├── parts.scenarios.ts
│       │   ├── parts.test.ts
│       │   └── parts.ts
│       ├── transactions
│       │   ├── transactions.scenarios.ts
│       │   ├── transactions.test.ts
│       │   └── transactions.ts
│       └── users
│           ├── users.scenarios.ts
│           ├── users.test.ts
│           └── users.ts
├── tsconfig.json
└── types
    └── graphql.d.ts
```

https://github.com/cy1der/arduino-parts-inventory

documents/ tree

```
documents
├── code
│   └── Code.odt
│   └── Code.pdf
├── flowchart
│   ├── flowchart.pdf
│   ├── flowchart.xml
│   └── readme.txt
├── ipo
│   ├── ipo.pdf
│   └── ipo.tex
├── requirements
│   ├── requirements.pdf
│   └── requirements.tex
└── specification
    ├── specification.pdf
    └── specification.tex
```

https://github.com/cy1der/arduino-parts-inventory

scripts/ tree

```
scripts
├── seed.ts
└── tsconfig.json
```

https://github.com/cy1der/arduino-parts-inventory

web/ tree

```
web
├── config
│   ├── postcss.config.js
│   └── tailwind.config.js
├── jest.config.js
├── node_modules
├── package.json
├── public
│   ├── README.md
│   ├── favicon.png
│   ├── no_image.png
│   └── robots.txt
├── src
│   ├── App.tsx
│   ├── Routes.tsx
│   ├── auth.ts
│   ├── components
│   │   ├── AdminMenu
│   │   │   ├── AdminMenu.stories.tsx
│   │   │   ├── AdminMenu.test.tsx
│   │   │   └── AdminMenu.tsx
│   │   ├── AdminTransactionsCell
│   │   │   ├── AdminTransactionsCell.mock.ts
│   │   │   ├── AdminTransactionsCell.stories.tsx
│   │   │   ├── AdminTransactionsCell.test.tsx
│   │   │   └── AdminTransactionsCell.tsx
│   │   ├── BasketCell
│   │   │   ├── BasketCell.mock.ts
│   │   │   ├── BasketCell.stories.tsx
│   │   │   ├── BasketCell.test.tsx
│   │   │   └── BasketCell.tsx
│   │   ├── NavbarAccountIcon
│   │   │   ├── NavbarAccountIcon.stories.tsx
│   │   │   ├── NavbarAccountIcon.test.tsx
│   │   │   └── NavbarAccountIcon.tsx
│   │   ├── Part
│   │   │   ├── EditPartCell
│   │   │   │   └── EditPartCell.tsx
│   │   │   ├── NewPart
│   │   │   │   └── NewPart.tsx
│   │   │   ├── Part
│   │   │   │   └── Part.tsx
│   │   │   ├── PartCell
│   │   │   │   └── PartCell.tsx
│   │   │   ├── PartForm
│   │   │   │   └── PartForm.tsx
│   │   │   ├── Parts
│   │   │   │   └── Parts.tsx
│   │   │   └── PartsCell
│   │   │       └── PartsCell.tsx
│   │   ├── PartDetailsCell
│   │   │   ├── PartDetailsCell.mock.ts
│   │   │   ├── PartDetailsCell.stories.tsx
│   │   │   ├── PartDetailsCell.test.tsx
│   │   │   └── PartDetailsCell.tsx
│   │   ├── PartsCell
│   │   │   ├── PartsCell.mock.ts
│   │   │   ├── PartsCell.stories.tsx
│   │   │   ├── PartsCell.test.tsx
│   │   │   └── PartsCell.tsx
│   │   ├── PartsGridUnit
│   │   │   ├── PartsGridUnit.stories.tsx
│   │   │   ├── PartsGridUnit.test.tsx
│   │   │   └── PartsGridUnit.tsx
│   │   ├── ThemeToggle
│   │   │   ├── ThemeToggle.stories.tsx
│   │   │   ├── ThemeToggle.test.tsx
│   │   │   └── ThemeToggle.tsx
│   │   ├── ToastNotification
│   │   │   ├── ToastNotification.stories.tsx
│   │   │   ├── ToastNotification.test.tsx
│   │   │   └── ToastNotification.tsx
│   │   ├── TransactionListItem
│   │   │   ├── TransactionListItem.stories.tsx
│   │   │   ├── TransactionListItem.test.tsx
│   │   │   └── TransactionListItem.tsx
│   │   └── UserTransactionsCell
│   │       ├── UserTransactionsCell.mock.ts
│   │       ├── UserTransactionsCell.stories.tsx
│   │       ├── UserTransactionsCell.test.tsx
│   │       └── UserTransactionsCell.tsx
│   ├── entry.client.tsx
│   ├── index.css
│   ├── index.html
│   ├── layouts
│   │   ├── NavbarLayout
│   │   │   ├── NavbarLayout.stories.tsx
│   │   │   ├── NavbarLayout.test.tsx
│   │   │   └── NavbarLayout.tsx
│   │   └── ScaffoldLayout
│   │       └── ScaffoldLayout.tsx
│   ├── lib
│   │   ├── basket.ts
│   │   ├── formatters.test.tsx
│   │   └── formatters.tsx
│   ├── pages
│   │   ├── AdminTransactionsPage
│   │   │   ├── AdminTransactionsPage.stories.tsx
│   │   │   ├── AdminTransactionsPage.test.tsx
│   │   │   └── AdminTransactionsPage.tsx
│   │   ├── BasketPage
│   │   │   ├── BasketPage.stories.tsx
│   │   │   ├── BasketPage.test.tsx
│   │   │   └── BasketPage.tsx
│   │   ├── FatalErrorPage
│   │   │   └── FatalErrorPage.tsx
│   │   ├── ForgotPasswordPage
│   │   │   └── ForgotPasswordPage.tsx
│   │   ├── HomePage
│   │   │   ├── HomePage.stories.tsx
│   │   │   ├── HomePage.test.tsx
│   │   │   └── HomePage.tsx
│   │   ├── LoginPage
│   │   │   └── LoginPage.tsx
│   │   ├── NotFoundPage
│   │   │   └── NotFoundPage.tsx
│   │   ├── Part
│   │   │   ├── EditPartPage
│   │   │   │   └── EditPartPage.tsx
│   │   │   ├── NewPartPage
│   │   │   │   └── NewPartPage.tsx
│   │   │   ├── PartPage
│   │   │   │   └── PartPage.tsx
│   │   │   └── PartsPage
│   │   │       └── PartsPage.tsx
│   │   ├── PartPage
│   │   │   ├── PartPage.stories.tsx
│   │   │   ├── PartPage.test.tsx
│   │   │   └── PartPage.tsx
│   │   ├── ResetPasswordPage
│   │   │   └── ResetPasswordPage.tsx
│   │   ├── SignupPage
│   │   │   └── SignupPage.tsx
│   │   └── TransactionsPage
│   │       ├── TransactionsPage.stories.tsx
│   │       ├── TransactionsPage.test.tsx
│   │       └── TransactionsPage.tsx
│   └── scaffold.css
├── tsconfig.json
├── types
│   └── graphql.d.ts
└── vite.config.ts
```

.env.example

```
REDWOOD_ENV_FILESTACK_API_KEY=

REDWOOD_ENV_FILESTACK_SECRET=

SESSION_SECRET=

ADMIN_EMAILS=foo@bar.com,fizz@buzz.com,john@example.com

DATABASE_URL=postgresql://user:password@localhost:5432/arduino_parts_inventory

PROD_DOMAIN=https://example.com/

DEV_DOMAIN=http://localhost:8910/

SEND_IN_BLUE_EMAIL=

SEND_IN_BLUE_KEY=
```

.gitignore

```
.idea
.DS_Store
.env
.netlify
.redwood/*
!.redwood/README.md
dev.db*
dist
dist-babel
node_modules
yarn-error.log
web/public/mockServiceWorker.js
web/types/graphql.d.ts
api/types/graphql.d.ts
api/src/lib/generateGraphiQLHeader.*
.pnp.*
.yarn/*
!.yarn/patches
!.yarn/plugins
!.yarn/releases
!.yarn/sdks
!.yarn/versions

*.aux
*.fdb_latexmk
*.fls
*.log
*.synctex.gz
```

api/db/schema.prisma

```
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

generator client {
  provider      = "prisma-client-js"
  binaryTargets = "native"
}

model Part {
  id             Int      @id @default(autoincrement())
  name           String
  description    String?  @default("No description provided")
  availableStock Int      @default(0)
  imageUrl       String   @default("/no_image.png")
  createdAt      DateTime @default(now())
  transactionId  Int?
}

model User {
  id                  Int           @id @default(autoincrement())
  firstName           String
  lastName            String
  email               String        @unique
  hashedPassword      String
  salt                String
  resetToken          String?
  resetTokenExpiresAt DateTime?
  roles               String        @default("user")
  transactions        Transaction[]
}

enum TransactionType {
  in
  out
}

model Transaction {
  id     Int              @id @default(autoincrement())
  date   DateTime         @default(now())
  user   User             @relation(fields: [userId], references: [id])
  userId Int
  type   TransactionType
  parts  Json[] // { part: Part, quantity: Int }[]
}
```

api/package.json

```
{
  "name": "api",
  "version": "0.0.0",
  "private": true,
  "dependencies": {
    "@redwoodjs/api": "6.4.2",
    "@redwoodjs/auth-dbauth-api": "6.4.2",
    "@redwoodjs/graphql-server": "6.4.2",
    "filestack-js": "^3.27.0",
    "nodemailer": "^6.9.7"
  },
  "devDependencies": {
    "@types/nodemailer": "^6.4.14"
  }
}
```

api/src/directives/requireAuth/requireAuth.ts

```
import gql from 'graphql-tag'

import type { ValidatorDirectiveFunc } from '@redwoodjs/graphql-server'
import { createValidatorDirective } from '@redwoodjs/graphql-server'

import { requireAuth as applicationRequireAuth } from 'src/lib/auth'

export const schema = gql`
  """
  Use to check whether or not a user is authenticated and is associated
  with an optional set of roles.
  """
  directive @requireAuth(roles: [String]) on FIELD_DEFINITION
`

type RequireAuthValidate = ValidatorDirectiveFunc<{ roles?: string[] }>

const validate: RequireAuthValidate = ({ directiveArgs }) => {
  const { roles } = directiveArgs
  applicationRequireAuth({ roles })
}

const requireAuth = createValidatorDirective(schema, validate)

export default requireAuth
```

api/src/directives/skipAuth/skipAuth.ts

```
import gql from 'graphql-tag'

import { createValidatorDirective } from '@redwoodjs/graphql-server'

export const schema = gql`
  """
  Use to skip authentication checks and allow public access.
  """
  directive @skipAuth on FIELD_DEFINITION
`

const skipAuth = createValidatorDirective(schema, () => {
  return
})

export default skipAuth
```

api/src/functions/auth.ts

```
import type { APIGatewayProxyEvent, Context } from 'aws-lambda'

import {
  DbAuthHandler,
  DbAuthHandlerOptions,
  PasswordValidationError,
} from '@redwoodjs/auth-dbauth-api'

import { db } from 'src/lib/db'
import { sendEmail } from 'src/lib/email'

export const handler = async (
  event: APIGatewayProxyEvent,
  context: Context
) => {
  const forgotPasswordOptions: DbAuthHandlerOptions['forgotPassword'] = {
    // handler() is invoked after verifying that a user was found with the given
    // username. This is where you can send the user an email with a link to
    // reset their password. With the default dbAuth routes and field names, the
    // URL to reset the password will be:
    //
    // https://example.com/reset-password?resetToken=${user.resetToken}
    //
    // Whatever is returned from this function will be returned from
    // the `forgotPassword()` function that is destructured from `useAuth()`
    // You could use this return value to, for example, show the email
    // address in a toast message so the user will know it worked and where
    // to look for the email.
    handler: async (user) => {
      const env = process.env.NODE_ENV || 'development'
      const domain =
        env == 'production' ? process.env.PROD_DOMAIN :
process.env.DEV_DOMAIN

      const text = `If this wasn't you, please disregard this email.\n\n\
nHello ${user.firstName},\n\nYou are receiving this email because a
password reset was requested.\nEnter the following URL to begin resetting
your password:\n\n${domain}reset-password?resetToken=${user.resetToken}
      `

      const html = text.replaceAll('\n', '<br />')

      const subject = 'Password Reset Request'

      await sendEmail({ to: user.email, subject, text, html })

      return user
    },

    // How long the resetToken is valid for, in seconds (default is 24
hours)
    expires: 60 * 60 * 24,

    errors: {
      // for security reasons you may want to be vague here rather than
expose
      // the fact that the email address wasn't found (prevents fishing for
      // valid email addresses)
      usernameNotFound: 'Email not found',
      // if the user somehow gets around client validation
      usernameRequired: 'Email is required',
    },
  }

  const loginOptions: DbAuthHandlerOptions['login'] = {
    // handler() is called after finding the user that matches the
    // username/password provided at login, but before actually considering
them
    // logged in. The `user` argument will be the user in the database that
    // matched the username/password.
    //
    // If you want to allow this user to log in simply return the user.
    //
    // If you want to prevent someone logging in for another reason (maybe
they
    // didn't validate their email yet), throw an error and it will be
returned
    // by the `logIn()` function from `useAuth()` in the form of:
    // `{ message: 'Error message' }`
    handler: (user) => {
      return user
    },

    errors: {
      usernameOrPasswordMissing: 'Both email and password are required',
      usernameNotFound: 'Email ${username} not found',
      // For security reasons you may want to make this the same as the
      // usernameNotFound error so that a malicious user can't use the
error
      // to narrow down if it's the username or password that's incorrect
      incorrectPassword: 'Incorrect password for ${username}',
    },

    // How long a user will remain logged in, in seconds
    expires: 60 * 60 * 24 * 365 * 10,
  }

  const resetPasswordOptions: DbAuthHandlerOptions['resetPassword'] = {
    // handler() is invoked after the password has been successfully updated in
    // the database. Returning anything truthy will automatically log the user
    // in. Return `false` otherwise, and in the Reset Password page redirect the
    // user to the login page.
    handler: (_user) => {
      return true
    },

    // If `false` then the new password MUST be different from the current one
    allowReusedPassword: false,

    errors: {
      // the resetToken is valid, but expired
      resetTokenExpired: 'resetToken is expired',
      // no user was found with the given resetToken
      resetTokenInvalid: 'resetToken is invalid',
      // the resetToken was not present in the URL
      resetTokenRequired: 'resetToken is required',
      // new password is the same as the old password (apparently they did not forget it)
      reusedPassword: 'Must choose a new password',
    },
  }


  const signupOptions: DbAuthHandlerOptions['signup'] = {
    // Whatever you want to happen to your data on new user signup. Redwood will
    // check for duplicate usernames before calling this handler. At a minimum
    // you need to save the `username`, `hashedPassword` and `salt` to your
    // user table. `userAttributes` contains any additional object members that
    // were included in the object given to the `signUp()` function you got
    // from `useAuth()`.
    //
    // If you want the user to be immediately logged in, return the user that
    // was created.
    //
    // If this handler throws an error, it will be returned by the `signUp()`
    // function in the form of: `{ error: 'Error message' }`.
    //
    // If this returns anything else, it will be returned by the
    // `signUp()` function in the form of: `{ message: 'String here' }`.
    handler: ({ username, hashedPassword, salt, userAttributes }) => {
      const adminEmails: string[] = process.env.ADMIN_EMAILS.split(',')

      let role = 'user'
      const email = username.toLowerCase()

      if (adminEmails.includes(email)) role = 'admin'
      return db.user.create({
        data: {
          email: email,
          hashedPassword: hashedPassword,
          salt: salt,
          firstName: userAttributes.firstName,
          lastName: userAttributes.lastName,
          roles: role,
        },
      })
    },

    // Include any format checks for password here. Return `true` if the
    // password is valid, otherwise throw a `PasswordValidationError`.
    // Import the error along with `DbAuthHandler` from `@redwoodjs/api` above.
    passwordValidation: (password) => {
      if (password.length < 6)
        throw new PasswordValidationError(
          'Password must be at least 6 characters'
        )
      else return true
    },

    errors: {
      // `field` will be either "username" or "password"
      fieldMissing: '${field} is required',
      usernameTaken: 'Email `${username}` already in use',
    },
  }
  const authHandler = new DbAuthHandler(event, context, {
    // Provide prisma db client
    db: db,

    // The name of the property you'd call on `db` to access your user
table.
    // i.e. if your Prisma model is named `User` this value would be
`user`, as in `db.user`
    authModelAccessor: 'user',

    // A map of what dbAuth calls a field to what your database calls it.
    // `id` is whatever column you use to uniquely identify a user
(probably
    // something like `id` or `userId` or even `email`)
    authFields: {
      id: 'id',
      username: 'email',
      hashedPassword: 'hashedPassword',
      salt: 'salt',
      resetToken: 'resetToken',
      resetTokenExpiresAt: 'resetTokenExpiresAt',
    },

    // Specifies attributes on the cookie that dbAuth sets in order to
remember
    // who is logged in. See
https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#restrict_access_t
o_cookies
    cookie: {
      HttpOnly: true,
      Path: '/',
      SameSite: 'Strict',

      Secure: process.env.NODE_ENV !== 'development',


      // If you need to allow other domains (besides the api side) access
to

      // the dbAuth session cookie:

      // Domain: 'example.com',

    },


    forgotPassword: forgotPasswordOptions,

    login: loginOptions,

    resetPassword: resetPasswordOptions,

    signup: signupOptions,

  })


  return await authHandler.invoke()

}
```

api/src/functions/graphql.ts

```
import { authDecoder } from '@redwoodjs/auth-dbauth-api'
import { createGraphQLHandler } from '@redwoodjs/graphql-server'

import directives from 'src/directives/**/*.{js,ts}'
import sdls from 'src/graphql/**/*.sdl.{js,ts}'
import services from 'src/services/**/*.{js,ts}'

import { getCurrentUser } from 'src/lib/auth'
import { db } from 'src/lib/db'
import { logger } from 'src/lib/logger'

export const handler = createGraphQLHandler({
  authDecoder,
  getCurrentUser,
  loggerConfig: { logger, options: {} },
  directives,
  sdls,
  services,
  onException: () => {
    // Disconnect from your database with an unhandled exception.
    db.$disconnect()
  },
})
```

api/src/graphql/parts.sdl.ts

```
export const schema = gql`
  type Part {
    id: Int!
    name: String!
    description: String
    availableStock: Int!
    imageUrl: String!
    createdAt: DateTime!
  }

  type PartPage {
    parts: [Part!]!
    count: Int!
    page: Int!
    sort: SortMethod!
    order: SortOrder!
    search: String
  }

  enum SortMethod {
    id
    name
    description
    stock
    createdAt
  }

  enum SortOrder {
    ascending
    descending
  }

  type Query {
    partPage(
      page: Int!
      sort: SortMethod!
      order: SortOrder!
      searchQuery: String
    ): PartPage @skipAuth
    parts: [Part!]! @skipAuth
    part(id: Int!): Part @skipAuth
  }

  input CreatePartInput {
    name: String!
    description: String
    availableStock: Int!
    imageUrl: String!
  }

  input UpdatePartInput {
    name: String
    description: String
    availableStock: Int
    imageUrl: String
  }

  type Mutation {
    createPart(input: CreatePartInput!): Part! @requireAuth
    updatePart(id: Int!, input: UpdatePartInput!): Part!
      @requireAuth(roles: "admin")
    deletePart(id: Int!): Part! @requireAuth(roles: "admin")
  }
`
```

api/src/graphql/transactions.sdl.ts

```
export const schema = gql`
  type Transaction {
    id: Int!
    date: DateTime!
    user: User!
    userId: Int!
    type: TransactionType!
    parts: [JSON]!
  }

  enum TransactionType {
    in
    out
  }

  enum FilterTransactionsByType {
    in
    out
    both
  }

  type UserTransactions {
    transactions: [Transaction!]!
    filter: FilterTransactionsByType!
  }

  type Query {
    transactions(filter: FilterTransactionsByType!): UserTransactions!
      @requireAuth(roles: "admin")
    transaction(id: Int!): Transaction @requireAuth(roles: "admin")
    userTransactions(
      userId: Int!
      filter: FilterTransactionsByType!
    ): UserTransactions! @requireAuth
  }

  input CreateTransactionInput {
    date: DateTime!
    userId: Int!
    type: TransactionType!
    parts: [JSON]!
  }

  input UpdateTransactionInput {
    date: DateTime
    userId: Int
    type: TransactionType
    parts: [JSON]!
  }

  type Mutation {
    createTransaction(input: CreateTransactionInput!): Transaction!
@requireAuth
    returnTransaction(id: Int!, userId: Int!): Transaction! @requireAuth
    updateTransaction(id: Int!, input: UpdateTransactionInput!):
Transaction!
      @requireAuth(roles: "admin")
    deleteTransaction(id: Int!): Transaction! @requireAuth(roles: "admin")
  }
`
```

api/src/graphql/users.sdl.ts

```
export const schema = gql`
  type User {
    id: Int!
    firstName: String!
    lastName: String!
    email: String!
    hashedPassword: String!
    salt: String!
    resetToken: String
    resetTokenExpiresAt: DateTime
    roles: String!
    transactions: [Transaction]!
  }

  type Query {
    users: [User!]! @requireAuth(roles: "admin")
    user(id: Int!): User @requireAuth(roles: "admin")
  }

  input CreateUserInput {
    firstName: String!
    lastName: String!
    email: String!
    hashedPassword: String!
    salt: String!
    resetToken: String
    resetTokenExpiresAt: DateTime
    roles: String!
  }

  input UpdateUserInput {
    firstName: String
    lastName: String
    email: String
    hashedPassword: String
    salt: String
    resetToken: String
    resetTokenExpiresAt: DateTime
    roles: String
  }

  type Mutation {
    createUser(input: CreateUserInput!): User! @requireAuth(roles: "admin")
    updateUser(id: Int!, input: UpdateUserInput!): User!
      @requireAuth(roles: "admin")
    deleteUser(id: Int!): User! @requireAuth(roles: "admin")
  }
`
```

api/src/lib/auth.ts

```
import type { Decoded } from '@redwoodjs/api'
import { AuthenticationError, ForbiddenError } from '@redwoodjs/graphql-server'

import { db } from './db'

/**
 * The session object sent in as the first argument to getCurrentUser() will
 * have a single key `id` containing the unique ID of the logged in user
 * (whatever field you set as `authFields.id` in your auth function config).
 * You'll need to update the call to `db` below if you use a different model
 * name or unique field name, for example:
 *
 *   return await db.profile.findUnique({ where: { email: session.id } })
 *                   ───┬───                       ──┬──
 *      model accessor ─┘      unique id field name ─┘
 *
 * !! BEWARE !! Anything returned from this function will be available to the
 * client--it becomes the content of `currentUser` on the web side (as well as
 * `context.currentUser` on the api side). You should carefully add additional
 * fields to the `select` object below once you've decided they are safe to be
 * seen if someone were to open the Web Inspector in their browser.
 */
export const getCurrentUser = async (session: Decoded) => {
  if (!session || typeof session.id !== 'number') {
    throw new Error('Invalid session')
  }
  return await db.user.findUnique({
    where: { id: session.id },
    select: { id: true, firstName: true, roles: true, transactions: true },
  })
}

/**
 * The user is authenticated if there is a currentUser in the context
 *
 * @returns {boolean} - If the currentUser is authenticated
 */
export const isAuthenticated = (): boolean => {
  return !!context.currentUser
}

/**
 * When checking role membership, roles can be a single value, a list, or none.
 * You can use Prisma enums too (if you're using them for roles), just import your enum type from `@prisma/client`
 */
type AllowedRoles = string | string[] | undefined

/**
 * Checks if the currentUser is authenticated (and assigned one of the given roles)
 *
 * @param roles: {@link AllowedRoles} - Checks if the currentUser is assigned one of these roles
 *
 * @returns {boolean} - Returns true if the currentUser is logged in and assigned one of the given roles,
 * or when no roles are provided to check against. Otherwise returns false.
 */
```

```
export const hasRole = (roles: AllowedRoles): boolean => {
  if (!isAuthenticated()) {
    return false
  }

  // If your User model includes roles, uncomment the role checks on
currentUser
  if (roles) {
    if (Array.isArray(roles)) {
      // the line below has changed
      if (context.currentUser.roles)
        return context.currentUser.roles
          .split(',')
          .some((role) => roles.includes(role))
    }

    if (typeof roles === 'string') {
      // the line below has changed
      if (context.currentUser.roles)
        return context.currentUser.roles.split(',').includes(roles)
    }

    // roles not found
    return false
  }

  return true
}

/**
 * Use requireAuth in your services to check that a user is logged in,
 * whether or not they are assigned a role, and optionally raise an
 * error if they're not.
 *
 * @param roles: {@link AllowedRoles} - When checking role membership,
these roles grant access.
 *
 * @returns - If the currentUser is authenticated (and assigned one of the
given roles)
 *
 * @throws {@link AuthenticationError} - If the currentUser is not
authenticated
 * @throws {@link ForbiddenError} If the currentUser is not allowed due to
role permissions
 *
 * @see https://github.com/redwoodjs/redwood/tree/main/packages/auth for
examples
 */
export const requireAuth = ({ roles }: { roles?: AllowedRoles } = {}) => {
  if (!isAuthenticated()) {
    throw new AuthenticationError("You don't have permission to do that.")
  }

  if (roles && !hasRole(roles)) {
    throw new ForbiddenError("You don't have access to do that.")
  }
}
```

api/src/lib/db.ts

```
// See https://www.prisma.io/docs/reference/tools-and-interfaces/prisma-client/constructor
// for options.

import { PrismaClient } from '@prisma/client'

import { emitLogLevels, handlePrismaLogging } from '@redwoodjs/api/logger'

import { logger } from './logger'

/*
 * Instance of the Prisma Client
 */
export const db = new PrismaClient({
  log: emitLogLevels(['info', 'warn', 'error']),
})

handlePrismaLogging({
  db,
  logger,
  logLevels: ['info', 'warn', 'error'],
})
```

api/src/lib/email.ts

```
import * as nodemailer from 'nodemailer'

interface Options {
  to: string | string[]
  subject: string
  html: string
  text: string
}

export async function sendEmail({ to, subject, text, html }: Options) {
  const transporter = nodemailer.createTransport({
    host: 'smtp-relay.brevo.com',
    port: 587,
    secure: false,
    auth: {
      user: process.env.SEND_IN_BLUE_EMAIL,
      pass: process.env.SEND_IN_BLUE_KEY,
    },
  })

  const info = await transporter.sendMail({
    from: `"Parts Inventory (noreply)" \<${process.env.SEND_IN_BLUE_EMAIL}>`,
    to: Array.isArray(to) ? to : [to],
    subject,
    text,
    html,
  })

  return info
}
```

https://github.com/cy1der/arduino-parts-inventory

api/src/services/parts/parts.ts

```
import * as Filestack from 'filestack-js'
import type { QueryResolvers, MutationResolvers } from 'types/graphql'

import { db } from 'src/lib/db'

const PARTS_PER_PAGE = 8

const removeEnding = (input: string): string =>
  input.endsWith('ending') ? input.slice(0, -6) : input

export const parts: QueryResolvers['parts'] = () => {
  return db.part.findMany()
}

export const part: QueryResolvers['part'] = ({ id }) => {
  return db.part.findUnique({
    where: { id },
  })
}

export const partPage: QueryResolvers['partPage'] = async ({
  page = 1,
  sort = 'id',
  order = 'ascending',
  searchQuery,
}) => {
  const offset = (page - 1) * PARTS_PER_PAGE
  let orderByCase
  switch (sort) {
    case 'id':
      orderByCase = { id: removeEnding(order) }
      break

    case 'name':
      orderByCase = { name: removeEnding(order) }
      break

    case 'createdAt':
      orderByCase = { createdAt: removeEnding(order) }
      break

    case 'description':
      orderByCase = { description: removeEnding(order) }
      break

    case 'stock':
      orderByCase = {
        availableStock: removeEnding(order),
      }
      break

    default:
      orderByCase = { id: removeEnding(order) }
      break
  }

  if (searchQuery && searchQuery.length > 0)
    return {
      parts: await db.part.findMany({
        where: {
          name: {
            contains: searchQuery,
          },
        },
        take: PARTS_PER_PAGE,
        skip: offset,
        orderBy: orderByCase,
      }),
      count: await db.part.count({
        where: {
          name: {
            contains: searchQuery,
          },
        },
      }),
      page,
      sort,
      order,
      search: searchQuery,
    }
  else
    return {
      parts: await db.part.findMany({
        take: PARTS_PER_PAGE,
        skip: offset,
        orderBy: orderByCase,
      }),
      count: await db.part.count(),
      page,
      sort,
      order,
    }
}

export const createPart: MutationResolvers['createPart'] = ({ input }) => {
  input.description =
    input.description.length == 0
      ? 'No description provided'
      : input.description

  return db.part.create({
    data: input,
  })
}

export const updatePart: MutationResolvers['updatePart'] = ({ id, input })
=> {
  input.description =
    input.description.length == 0
      ? 'No description provided'
      : input.description

  return db.part.update({
    data: input,
    where: { id },
  })
}
```

https://github.com/cy1der/arduino-parts-inventory

```
export const deletePart: MutationResolvers['deletePart'] = async ({ id })
=> {
  const client = Filestack.init(process.env.REDWOOD_ENV_FILESTACK_API_KEY)
  const part = await db.part.findUnique({ where: { id } })

  if (!part.imageUrl.includes('no_image.png')) {
    const handle = part.imageUrl.split('/').pop()

    const security = Filestack.getSecurity(
      {
        expiry: new Date().getTime() + 5 * 60 * 1000,
        handle,
        call: ['remove'],
      },
      process.env.REDWOOD_ENV_FILESTACK_SECRET
    )

    await client.remove(handle, security)
  }

  return db.part.delete({
    where: { id },
  })
}
```

api/src/services/transactions/transactions.ts

```
import type {
  QueryResolvers,
  MutationResolvers,
  TransactionRelationResolvers,
  Part,
} from 'types/graphql'

import { UserInputError } from '@redwoodjs/graphql-server'

import { db } from 'src/lib/db'

export const transactions: QueryResolvers['transactions'] = async ({
  filter,
}) => {
  const transactions =
    filter == 'both'
      ? await db.transaction.findMany({
          orderBy: {
            date: 'desc',
          },
        })
      : await db.transaction.findMany({
          where: {
            type: filter,
          },
          orderBy: {
            date: 'desc',
          },
        })
  return {
    transactions,
    filter,
  }
}

export const userTransactions: QueryResolvers['userTransactions'] = async
({
  userId,
  filter,
}) => {
  return {
    transactions:
      filter == 'both'
        ? await db.transaction.findMany({
            where: {
              userId,
            },
            orderBy: {
              date: 'desc',
            },
          })
        : await db.transaction.findMany({
            where: {
              AND: {
                userId,
                type: filter,
              },
            },
            orderBy: {
              date: 'desc',
            },
          }),
    filter,
  }
}

export const transaction: QueryResolvers['transaction'] = ({ id }) => {
  return db.transaction.findUnique({
    where: { id },
  })
}

export const returnTransaction: MutationResolvers['returnTransaction'] =
  async ({ id, userId }) => {
    const transaction = await db.transaction.findUnique({ where: { id } })

    if (transaction.type == 'out' && userId == transaction.userId) {
      for (const partRaw of transaction.parts) {
        const transactionPart = JSON.parse(partRaw['part']) as Part
        const part = await db.part.findUnique({
          where: { id: transactionPart.id },
        })

        await db.part.update({
          where: { id: part.id },
          data: {
            availableStock: {
              increment: partRaw['quantity'],
            },
          },
        })
      }

      return await db.transaction.update({
        where: { id: transaction.id },
        data: { type: 'in' },
      })
    } else return transaction
  }

export const createTransaction: MutationResolvers['createTransaction'] =
  async ({ input }) => {
    const basket = input.parts.map((item) => {
      const part: Part = JSON.parse(item['part']) as Part
      return { part: part.id, quantity: item['quantity'] }
    })

    for (const item of basket) {
      const part = await db.part.findUnique({ where: { id: item.part } })

      if (!part) throw new UserInputError(`Part ${item.part} does not
exist`)

      if (input.type == 'out') {
        if (part.availableStock < item.quantity)
          throw new UserInputError(
            `Cannot take out more than available stock ($
{part.availableStock} available, ${item.quantity} requested)`
          )
        await db.part.update({
          where: { id: item.part },
          data: { availableStock: { decrement: item.quantity } },
        })
      } else if (input.type == 'in') {
        await db.part.update({
          where: { id: item.part },
          data: { availableStock: { increment: item.quantity } },
        })
      }
    }

    return db.transaction.create({
      data: input,
    })
  }

export const updateTransaction: MutationResolvers['updateTransaction'] = ({
  id,
  input,
}) => {
  return db.transaction.update({
    data: input,
    where: { id },
  })
}

export const deleteTransaction: MutationResolvers['deleteTransaction'] = ({
  id,
}) => {
  return db.transaction.delete({
    where: { id },
  })
}

export const Transaction: TransactionRelationResolvers = {
  user: (_obj, { root }) => {
    return db.transaction.findUnique({ where: { id: root?.id } }).user()
  },
}
```

api/src/services/users/users.ts

```
import type {
  QueryResolvers,
  MutationResolvers,
  UserRelationResolvers,
} from 'types/graphql'

import { db } from 'src/lib/db'

export const users: QueryResolvers['users'] = () => {
  return db.user.findMany()
}

export const user: QueryResolvers['user'] = ({ id }) => {
  return db.user.findUnique({
    where: { id },
  })
}

export const createUser: MutationResolvers['createUser'] = ({ input }) => {
  return db.user.create({
    data: input,
  })
}

export const updateUser: MutationResolvers['updateUser'] = ({ id, input })
=> {
  return db.user.update({
    data: input,
    where: { id },
  })
}

export const deleteUser: MutationResolvers['deleteUser'] = ({ id }) => {
  return db.user.delete({
    where: { id },
  })
}

export const User: UserRelationResolvers = {
  transactions: (_obj, { root }) => {
    return db.user.findUnique({ where: { id: root?.id } }).transactions()
  },
}
```

web/config/tailwind.config.js

```
/** @type {import('tailwindcss').Config} */
export const content = ['src/**/*.{js,jsx,ts,tsx}']
export const theme = {
  extend: {
    fontFamily: {
      inter: ['Inter', 'sans-serif'],
    },
    colors: {
      logo: {
        DEFAULT: '#246EB9',
        hover: '#1f5d9d',
      },
    },
  },
}
export const plugins = [require('daisyui'), require('tailwindcss-animate')]
export const daisyui = {
  themes: ['light', 'dark'],
}
```

web/package.json

```
{
  "name": "web",
  "version": "0.0.0",
  "private": true,
  "browserslist": {
    "development": [
      "last 1 version"
    ],
    "production": [
      "defaults"
    ]
  },
  "dependencies": {
    "@mdi/js": "^7.3.67",
    "@mdi/react": "^1.6.1",
    "@redwoodjs/auth-d  bauth-web": "6.4.2",
    "@redwoodjs/forms": "6.4.2",
    "@redwoodjs/router": "6.4.2",
    "@redwoodjs/web": "6.4.2",
    "dayjs": "^1.11.10",
    "filestack-react": "^4.0.1",
    "humanize-string": "2.1.0",
    "prop-types": "15.8.1",
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "theme-change": "^2.5.0"
  },
  "devDependencies": {
    "@redwoodjs/vite": "6.4.2",
    "@types/filestack-react": "^4.0.3",
    "@types/node": "^20.8.9",
    "@types/react": "18.2.14",
    "@types/react-dom": "18.2.6",
    "autoprefixer": "^10.4.16",
    "daisyui": "^3.9.3",
    "postcss": "^8.4.31",
    "postcss-loader": "^7.3.3",
    "tailwindcss": "^3.3.3",
    "tailwindcss-animate": "^1.0.7"
  }
}
```

web/src/App.tsx

```
import { FatalErrorBoundary, RedwoodProvider } from '@redwoodjs/web'
import { RedwoodApolloProvider } from '@redwoodjs/web/apollo'

import FatalErrorPage from 'src/pages/FatalErrorPage'
import Routes from 'src/Routes'

import { AuthProvider, useAuth } from './auth'

import './scaffold.css'
import './index.css'

const App = () => (
  <FatalErrorBoundary page={FatalErrorPage}>
    <RedwoodProvider titleTemplate="%PageTitle | %AppTitle">
      <AuthProvider>
        <RedwoodApolloProvider useAuth={useAuth}>
          <Routes />
        </RedwoodApolloProvider>
      </AuthProvider>
    </RedwoodProvider>
  </FatalErrorBoundary>
)

export default App
```

web/src/Routes.tsx

```
import { Router, Route, Set, PrivateSet } from '@redwoodjs/router'

import NavbarLayout from 'src/layouts/NavbarLayout'
import ScaffoldLayout from 'src/layouts/ScaffoldLayout'

import { useAuth } from './auth'

const Routes = () => {
  return (
    <Router useAuth={useAuth}>
      <Route path="/login" page={LoginPage} name="login" />
      <Route path="/signup" page={SignupPage} name="signup" />
      <Route path="/forgot-password" page={ForgotPasswordPage} name="forgotPassword" />
      <Route path="/reset-password" page={ResetPasswordPage} name="resetPassword" />

      <PrivateSet unauthenticated="home" roles="admin">
        <Set wrap={ScaffoldLayout} title="Parts" titleTo="parts" buttonLabel="New Part" buttonTo="newPart">
          <Route path="/admin/parts/new" page={PartNewPartPage} name="newPart" />
          <Route path="/admin/parts/{id:Int}/edit" page={PartEditPartPage} name="editPart" />
          <Route path="/admin/parts/{id:Int}" page={PartPartPage} name="part" />
          <Route path="/admin/parts" page={PartPartsPage} name="parts" />
        </Set>
        <Set wrap={ScaffoldLayout} title="Transactions" titleTo="adminTransactions">
```

```
          <Route path="/admin/transactions" page={AdminTransactionsPage}
name="adminTransactions" />
        </Set>
      </PrivateSet>

      <Set wrap={NavbarLayout}>
        <Route path="/" page={HomePage} name="home" />
        <Route path="/part/{id:Int}" page={PartPage} name="partDetails" />
        <Route path="/basket" page={BasketPage} name="basket" />
        <PrivateSet unauthenticated="login">
          <Route path="/transactions" page={TransactionsPage}
name="userTransactions" />
        </PrivateSet>
      </Set>

      <Route notfound page={NotFoundPage} />
    </Router>
  )
}

export default Routes
```

web/src/auth.ts

```
import { createDbAuthClient, createAuth } from '@redwoodjs/auth-dbauth-web'

const dbAuthClient = createDbAuthClient()

export const { AuthProvider, useAuth } = createAuth(dbAuthClient)
```

web/src/components/AdminMenu/AdminMenu.tsx

```
import { mdiWrench } from '@mdi/js'
import Icon from '@mdi/react'

import { Link, routes } from '@redwoodjs/router'

import { useAuth } from 'src/auth'

interface Props {
  mobile: boolean
  className?: string
}

const AdminMenu = ({ mobile, className }: Props) => {
  const { isAuthenticated, hasRole } = useAuth()

  return isAuthenticated && hasRole('admin') ? (
    <div className={className}>
      <details
        className={`dropdown ${
          mobile ? 'dropdown-start space-y-2' : 'dropdown-end space-y-4'
        }`}
      >
        <summary className="btn btn-ghost swap swap-rotate w-12 hover:shadow-lg">
          <Icon path={mdiWrench} className="h-8 w-8 text-base-content" />
        </summary>
        <div className="dropdown-content flex flex-col items-center space-y-3 rounded-xl bg-base-100 p-3 shadow-lg">
          <ul className="space-y-3 bg-base-100 text-base-content">
```

```
            <li>
              <Link
                to={routes.parts()}
                className="btn btn-ghost w-full font-inter hover:shadow-lg"
              >
                Parts
              </Link>
            </li>
            <li>
              <Link
                to={routes.adminTransactions()}
                className="btn btn-ghost w-full hover:shadow-lg"
              >
                <p className="font-inter">Transactions</p>
              </Link>
            </li>
          </ul>
        </div>
      </details>
    </div>
  ) : (
    <></>
  )
}

export default AdminMenu
```

web/src/components/AdminTransactionsCell/AdminTransactionsCell.tsx

```
/* eslint-disable jsx-a11y/no-noninteractive-tabindex */
import { mdiAlert } from '@mdi/js'
import { Icon } from '@mdi/react'
import type { TransactionsQuery } from 'types/graphql'

import { Link, routes } from '@redwoodjs/router'
import type { CellSuccessProps, CellFailureProps } from '@redwoodjs/web'

import TransactionListItem from
'../TransactionListItem/TransactionListItem'

export const beforeQuery = ({ filter }) => {
  filter = filter && ['both', 'in', 'out'].includes(filter) ? filter :
'both'

  return { variables: { filter } }
}

export const QUERY = gql`
  query TransactionsQuery($filter: FilterTransactionsByType!) {
    transactions(filter: $filter) {
      transactions {
        id
        date
        parts
        type
        user {
          firstName
          lastName
        }
      }
      filter
    }
  }
`

export const Loading = () => (
  <div className="flex w-auto justify-center">
    <p className="loading loading-bars loading-lg" />
  </div>
)

export const Empty = () => (
  <div className="flex">
    <div className="alert w-auto">
      <p className="text-center font-inter">It&#39;s empty in here...</p>
    </div>
  </div>
)

export const Failure = ({ error }: CellFailureProps) => (
  <div className="flex w-auto justify-center">
    <div className="alert alert-error w-auto">
      <Icon path={mdiAlert} className="h-6 w-6" />
      <p className="font-inter">Error! {error?.message}</p>
    </div>
  </div>
)
```

https://github.com/cy1der/arduino-parts-inventory

```
export const Success = ({
  transactions,
}: CellSuccessProps<TransactionsQuery>) => {
  if (transactions.transactions.length == 0) return Empty()
  return (
    <div className="flex flex-col space-y-6 font-inter">
      <div className="dropdown">
        <label tabIndex={0} className="btn m-1 normal-case">
          Filter:{' '}
          {transactions.filter == 'both'
            ? 'None'
            : transactions.filter.replace(
                transactions.filter[0],
                transactions.filter[0].toUpperCase()
              )}
        </label>
        <ul
          tabIndex={0}
          className="dropdown-content rounded-box z-10 mt-3 w-auto space-y-3 bg-base-100 p-3 shadow"
        >
          {['None', 'In', 'Out'].map((filter) => (
            <li key={filter}>
              <Link
                className="btn btn-ghost w-full normal-case"
                to={routes.adminTransactions({
                  filter: filter === 'None' ? 'both' : filter.toLowerCase(),
                })}
              >
```

```
                {filter}
              </Link>
            </li>
          ))}
        </ul>
      </div>
      {transactions.transactions.map((item, i) => {
        return (
          <TransactionListItem
            transaction={item}
            key={i}
            returnButton={false}
            admin
          />
        )
      })}
    </div>
  )
}
```

web/src/components/BasketCell/BasketCell.tsx

```
import { useState } from 'react'

import { mdiMinus, mdiPlus, mdiDelete } from '@mdi/js'
import Icon from '@mdi/react'
import type { CreateTransactionInput } from 'types/graphql'

import { useMutation } from '@redwoodjs/web'
import { toast } from '@redwoodjs/web/dist/toast'

import { useAuth } from 'src/auth'
import {
  getBasket,
  setBasket,
  removeFromBasket,
  clearBasket,
} from 'src/lib/basket'

import ToastNotification from '../ToastNotification'

export const CREATE_TRANSACTION_MUTATION = gql`
  mutation CreateTransactionMutation($input: CreateTransactionInput!) {
    createTransaction(input: $input) {
      id
    }
  }
`

const thumbnail = (url: string) => {
  if (url.includes('no_image.png')) return url
  const parts = url.split('/')
  parts.splice(3, 0, 'resize=width:160')
  return parts.join('/')
}

export const BasketCell = () => {
  const { isAuthenticated, currentUser } = useAuth()
  const [basket, setBasketState] = useState(getBasket())

  const [createTransaction, { loading }] = useMutation(
    CREATE_TRANSACTION_MUTATION,
    {
      onCompleted: () => {
        toast.custom((t) => (
          <ToastNotification
            toast={t}
            type="success"
            message="Transaction complete"
          />
        ))

        setBasketState(clearBasket())
      },
      onError: (error) => {
        toast.custom((t) => (
          <ToastNotification toast={t} type="error" message={error.message}
/>
        ))
      },
    }
  )
  const submitBasket = (input: CreateTransactionInput) => {
    createTransaction({ variables: { input } })
  }

  return (
    <div className="space-y-3">
      {basket.length > 0 ? (
        basket.map((item, i) => (
          <div
            key={i}
            className="flex max-w-5xl items-center rounded-xl bg-base-200
shadow-xl"
          >
            <img
              alt={item.part.name}
              className="hidden h-20 w-20 rounded-l-xl object-cover
sm:flex"
              src={thumbnail(item.part.imageUrl)}
            />
            <div className="m-3 w-full items-center justify-between space-
y-3 sm:flex sm:space-y-0">
              <p className="overflow-hidden text-ellipsis whitespace-nowrap
font-inter text-lg font-bold">
                {item.part.name}
              </p>
              <div className="flex justify-between space-x-3">
                <div className="join">
                  <button
                    className={`btn join-item ${
                      item.quantity <= 1 ? 'btn-disabled' : ''
                    }`}
                    onClick={() => {
                      const newBasket = basket
                      newBasket[i].quantity -= 1
                      setBasketState(setBasket(newBasket))
                    }}
                  >
                    <Icon path={mdiMinus} className="h-6 w-6" />
                  </button>
                  <p className="btn join-item items-center font-inter text-lg">
                    {item.quantity}
                  </p>
                  <button
                    className={`btn join-item ${
                      item.quantity >= item.part.availableStock
                        ? 'btn-disabled'
                        : ''
                    }`}
                    onClick={() => {
                      const newBasket = basket
                      newBasket[i].quantity += 1
                      setBasketState(setBasket(newBasket))
                    }}
                  >
                    <Icon path={mdiPlus} className="h-6 w-6" />
                  </button>
                </div>
                <button
                  className="btn btn-ghost hover:shadow-lg"
                  onClick={() => {
                    const newBasket = removeFromBasket(i)
                    if (typeof newBasket == 'string')
                      toast.custom((t) => (
                        <ToastNotification
                          toast={t}
                          type="error"
                          message={newBasket}
                        />
                      ))
                    else {
                      setBasketState(newBasket)
                      toast.custom((t) => (
                        <ToastNotification
                          toast={t}
                          type="success"
                          message={`Removed ${item.part.name} from basket`}
                        />
                      ))
                    }
                  }}
                >
                  <Icon
                    path={mdiDelete}
                    className="h-8 w-8 text-base-content"
                  />
                </button>
              </div>
            </div>
          </div>
        ))
```

```
      ) : (
        <div className="flex">
          <div className="alert w-auto shadow-lg">
            <p className="text-center font-inter">It&#39;s empty in
here...</p>
          </div>
        </div>
      )}
      {basket.length > 0 ? (
        <div className="flex space-x-3 pt-3">
          <button
            onClick={() => {
              setBasketState(clearBasket())
              toast.custom((t) => (
                <ToastNotification
                  toast={t}
                  type="success"
                  message="Basket cleared"
                />
              ))
            }}
            className="btn font-inter"
          >
            Clear basket
          </button>
          <button
            disabled={loading}
            onClick={() => {
              if (!isAuthenticated)
                toast.custom((t) => (
                  <ToastNotification
```

```
                    toast={t}
                    type="error"
                    message="You must be logged in to do that"
                  />
                ))
              else {
                submitBasket({
                  date: new Date().toISOString(),
                  type: 'out',
                  userId: currentUser.id,
                  parts: basket.map((item) => ({
                    part: JSON.stringify(item.part),
                    quantity: item.quantity,
                  })),
                })
              }
            }}
            className={`btn btn-primary font-inter ${
              loading ? 'btn-disabled' : ''
            }`}
          >
            Checkout
          </button>
        </div>
      ) : (
        <></>
      )}
    </div>
  )
}
```

web/src/components/NavbarAccountIcon/NavbarAccountIcon.tsx

```
import { mdiAccount, mdiLogout, mdiLogin } from '@mdi/js'
import Icon from '@mdi/react'

import { Link, routes } from '@redwoodjs/router'

import { useAuth } from 'src/auth'

interface Props {
  mobile: boolean
  className?: string
}

const NavbarAccountIcon = ({ mobile, className }: Props) => {
  const { isAuthenticated, currentUser, logOut } = useAuth()

  return isAuthenticated ? (
    <div className={className}>
      <details
        className={`dropdown-end dropdown ${
          mobile ? 'space-y-2' : 'space-y-4'
        }`}
      >
        <summary className="btn btn-ghost swap swap-rotate w-12
hover:shadow-lg">
          <Icon path={mdiAccount} className="h-8 w-8 text-base-content" />
        </summary>
        <div className="dropdown-content flex w-auto flex-row items-center
space-x-3 rounded-xl bg-base-100 p-3 shadow-lg">
```

```
          <p className="whitespace-nowrap font-inter text-lg">
            {currentUser ? `Hello, ${currentUser.firstName}!` : ``}
          </p>
          <button className="btn btn-ghost" type="button" onClick={logOut}>
            <Icon path={mdiLogout} className="h-7 w-7 text-base-content" />
          </button>
        </div>
      </details>
    </div>
  ) : (
    <div className={className}>
      <Link to={routes.login()}>
        <button className="btn btn-ghost" type="button" onClick={logOut}>
          <Icon path={mdiLogin} className="h-8 w-8 text-base-content" />
        </button>
      </Link>
    </div>
  )
}

export default NavbarAccountIcon
```

web/src/components/Part/EditPartCell/EditPartCell.tsx

```
import type { EditPartById, UpdatePartInput } from 'types/graphql'

import { navigate, routes } from '@redwoodjs/router'
import type { CellSuccessProps, CellFailureProps } from '@redwoodjs/web'
import { useMutation } from '@redwoodjs/web'
import { toast } from '@redwoodjs/web/toast'

import PartForm from 'src/components/Part/PartForm'
import ToastNotification from 'src/components/ToastNotification'

export const QUERY = gql`
  query EditPartById($id: Int!) {
    part: part(id: $id) {
      id
      name
      description
      availableStock
      imageUrl
      createdAt
    }
  }
`
const UPDATE_PART_MUTATION = gql`
  mutation UpdatePartMutation($id: Int!, $input: UpdatePartInput!) {
    updatePart(id: $id, input: $input) {
      id
      name
      description
      availableStock
      imageUrl
      createdAt
    }
  }
`


export const Loading = () => <div>Loading...</div>


export const Failure = ({ error }: CellFailureProps) => (
  <div className="rw-cell-error">{error?.message}</div>
)


export const Success = ({ part }: CellSuccessProps<EditPartById>) => {
  const [updatePart, { loading, error }] =
useMutation(UPDATE_PART_MUTATION, {
    onCompleted: () => {
      toast.custom((t) => (
        <ToastNotification toast={t} type="success" message="Part
updated" />
      ))
      navigate(routes.parts())
    },
    onError: (error) => {
      toast.custom((t) => (
        <ToastNotification toast={t} type="error"
message={error.message} />
      ))
    },
  })


  const onSave = (input: UpdatePartInput, id: EditPartById['part']['id'])
=> {
    updatePart({ variables: { id, input } })
  }

  return (
    <div className="rw-segment">
      <header className="rw-segment-header">
        <h2 className="rw-heading rw-heading-secondary">
          Edit Part {part?.id}
        </h2>
      </header>
      <div className="rw-segment-main">
        <PartForm part={part} onSave={onSave} error={error}
loading={loading} />
      </div>
    </div>
  )
}
```

web/src/components/Part/NewPart/NewPart.tsx

```
import type { CreatePartInput } from 'types/graphql'

import { navigate, routes } from '@redwoodjs/router'
import { useMutation } from '@redwoodjs/web'
import { toast } from '@redwoodjs/web/toast'

import PartForm from 'src/components/Part/PartForm'
import ToastNotification from 'src/components/ToastNotification'

const CREATE_PART_MUTATION = gql`
  mutation CreatePartMutation($input: CreatePartInput!) {
    createPart(input: $input) {
      id
    }
  }
`


const NewPart = () => {
  const [createPart, { loading, error }] = useMutation(CREATE_PART_MUTATION, {
    onCompleted: () => {
      toast.custom((t) => (
        <ToastNotification toast={t} type="success" message="Part created" />
      ))
      navigate(routes.parts())
    },
    onError: (error) => {
      toast.custom((t) => (
        <ToastNotification toast={t} type="error" 
message={error.message} />
      ))
    },
  })

  const onSave = (input: CreatePartInput) => {
    createPart({ variables: { input } })
  }

  return (
    <div className="rw-segment">
      <header className="rw-segment-header">
        <h2 className="rw-heading rw-heading-secondary">New Part</h2>
      </header>
      <div className="rw-segment-main">
        <PartForm onSave={onSave} loading={loading} error={error} />
      </div>
    </div>
  )
}

export default NewPart
```

web/src/components/Part/Part/Part.tsx

```
import type { DeletePartMutationVariables, FindPartById } from
'types/graphql'

import { Link, routes, navigate } from '@redwoodjs/router'
import { useMutation } from '@redwoodjs/web'
import { toast } from '@redwoodjs/web/toast'

import ToastNotification from 'src/components/ToastNotification'
import { timeTag } from 'src/lib/formatters'

const DELETE_PART_MUTATION = gql`
  mutation DeletePartMutation($id: Int!) {
    deletePart(id: $id) {
      id
    }
  }
`

interface Props {
  part: NonNullable<FindPartById['part']>
}

const Part = ({ part }: Props) => {
  const [deletePart] = useMutation(DELETE_PART_MUTATION, {
    onCompleted: () => {
      toast.custom((t) => (
        <ToastNotification toast={t} type="success" message="Part
deleted" />
      ))
      navigate(routes.parts())
    },
    onError: (error) => {
      toast.custom((t) => (
        <ToastNotification toast={t} type="error" 
message={error.message} />
      ))
    },
  })

  const onDeleteClick = (id: DeletePartMutationVariables['id']) => {
    if (confirm('Are you sure you want to delete part ' + id + '?')) {
      deletePart({ variables: { id } })
    }
  }

  const preview = (url: string) => {
    if (url.includes('no_image.png')) return url
    const parts = url.split('/')
    parts.splice(3, 0, 'resize=height:500')
    return parts.join('/')
  }

  return (
    <>
      <div className="rw-segment font-inter">
        <header className="rw-segment-header">
          <h2 className="rw-heading rw-heading-secondary">
            Part {part.id} details
          </h2>
        </header>
        <table className="rw-table">
          <tbody>
            <tr>
              <th>ID</th>
              <td>{part.id}</td>
            </tr>
            <tr>
              <th>Name</th>
              <td>{part.name}</td>
            </tr>
            <tr>
              <th>Description</th>
              <td>{part.description}</td>
            </tr>
            <tr>
              <th>Available stock</th>
              <td>{part.availableStock}</td>
            </tr>
            <tr>
              <th>Image</th>
              <td>
                <img
                  alt=""
                  src={preview(part.imageUrl)}
                  style={{ display: 'block', margin: '2rem 0' }}
                />
              </td>
            </tr>
            <tr>
              <th>Created at</th>
              <td>{timeTag(part.createdAt)}</td>
            </tr>
          </tbody>
        </table>
      </div>
      <nav className="rw-button-group">
        <Link
          to={routes.editPart({ id: part.id })}
          className="rw-button btn-primary"
        >
          Edit
        </Link>
        <button
          type="button"
          className="rw-button btn-error"
          onClick={() => onDeleteClick(part.id)}
        >
          Delete
        </button>
      </nav>
    </>
  )
}

export default Part
```

web/src/components/Part/PartCell/PartCell.tsx

```
import type { FindPartById } from 'types/graphql'

import type { CellSuccessProps, CellFailureProps } from '@redwoodjs/web'

import Part from 'src/components/Part/Part'

export const QUERY = gql`
  query FindPartById($id: Int!) {
    part: part(id: $id) {
      id
      name
      description
      availableStock
      imageUrl
      createdAt
    }
  }
`

export const Loading = () => <div>Loading...</div>

export const Empty = () => <div>Part not found</div>

export const Failure = ({ error }: CellFailureProps) => (
  <div className="rw-cell-error">{error?.message}</div>
)

export const Success = ({ part }: CellSuccessProps<FindPartById>) => {
  return <Part part={part} />
```

https://github.com/cy1der/arduino-parts-inventory

```
}
```

web/src/components/Part/PartForm/PartForm.tsx

```
import { useState } from 'react'

import { PickerInline } from 'filestack-react'
import type { EditPartById, UpdatePartInput } from 'types/graphql'

import {
  Form,
  FormError,
  FieldError,
  Label,
  TextField,
  NumberField,
  Submit,
  TextAreaField,
} from '@redwoodjs/forms'
import type { RWGqlError } from '@redwoodjs/forms'

type FormPart = NonNullable<EditPartById['part']>

interface PartFormProps {
  part?: EditPartById['part']
  onSave: (data: UpdatePartInput, id?: FormPart['id']) => void
  error: RWGqlError
  loading: boolean
}

const PartForm = (props: PartFormProps) => {
  const [imageUrl, setImageUrl] = useState(props?.part?.imageUrl)
  const onSubmit = (data: FormPart) => {
    const dataUpdated = Object.assign(data, {
      imageUrl: imageUrl ?? '/no_image.png',
    })

    props.onSave(dataUpdated, props?.part?.id)
  }

  const onImageUpload = (response) => {
    setImageUrl(response.filesUploaded[0].url)
  }

  const preview = (url: string) => {
    if (url.includes('no_image.png')) return url
    const parts = url.split('/')
    parts.splice(3, 0, 'resize=height:500')
    return parts.join('/')
  }

  return (
    <div className="rw-form-wrapper">
      <Form<FormPart> onSubmit={onSubmit} error={props.error}>
        <FormError
          error={props.error}
          wrapperClassName="rw-form-error-wrapper"
          titleClassName="rw-form-error-title"
          listClassName="rw-form-error-list"
        />

        <TextField
```

```
          name="name"
          placeholder="Name"
          defaultValue={props.part?.name}
          className="rw-input mb-3 min-w-full"
          errorClassName="rw-input rw-input-error min-w-full"
          validation={{ required: true }}
        />

        <FieldError name="name" className="rw-field-error pb-3" />

        <TextAreaField
          name="description"
          placeholder="Description"
          defaultValue={props.part?.description}
          className="rw-textarea mb-1 min-w-full"
          errorClassName="rw-input rw-input-error"
        />

        <FieldError name="description" className="rw-field-error pb-3" />

        <NumberField
          name="availableStock"
          defaultValue={props.part?.availableStock ?? 0}
          className="rw-input min-w-full"
          errorClassName="rw-input rw-input-error min-w-full"
          validation={{ required: true, min: 0 }}
          min={0}
          max={2147483647}
        />
        <FieldError name="availableStock" className="rw-field-error pb-3" />

        <Label
          name="imageUrl"
          className="rw-label"
          errorClassName="rw-label rw-label-error"
        >
          Image
        </Label>

        {!imageUrl && (
          <div style={{ height: '500px' }}>
            <PickerInline
              onSuccess={onImageUpload}
              pickerOptions={{ accept: 'image/*' }}
              apikey={process.env.REDWOOD_ENV_FILESTACK_API_KEY}
            />
          </div>
        )}

        {imageUrl && (
          <div>
            <img
              alt=""
              src={preview(imageUrl)}
              style={{ display: 'block', margin: '2rem 0' }}
            />
            <button
              onClick={() => setImageUrl(null)}
              className="rw-button btn-primary"
            >
              Replace Image
            </button>
          </div>
        )}

        <FieldError name="imageUrl" className="rw-field-error" />

        <div className="rw-button-group">
          <Submit disabled={props.loading} className="rw-button btn-primary">
            Save
          </Submit>
        </div>
      </Form>
    </div>
  )
}

export default PartForm
```

web/src/components/Part/Parts/Parts.tsx

```
import type { DeletePartMutationVariables, FindParts } from 'types/graphql'

import { Link, routes } from '@redwoodjs/router'
import { useMutation } from '@redwoodjs/web'
import { toast } from '@redwoodjs/web/toast'

import { QUERY } from 'src/components/Part/PartsCell'
import ToastNotification from 'src/components/ToastNotification'
import { timeTag, truncate } from 'src/lib/formatters'

const DELETE_PART_MUTATION = gql`
  mutation DeletePartMutation($id: Int!) {
    deletePart(id: $id) {
      id
    }
  }
`

const PartsList = ({ parts }: FindParts) => {
  const [deletePart] = useMutation(DELETE_PART_MUTATION, {
    onCompleted: () => {
      toast.custom((t) => (
        <ToastNotification toast={t} type="success" message="Part deleted" />
      ))
    },
    onError: (error) => {
      toast.custom((t) => (
        <ToastNotification toast={t} type="error"
message={error.message} />
      ))
    },
    // This refetches the query on the list page. Read more about other
ways to
    // update the cache over here:
    // https://www.apollographql.com/docs/react/data/mutations/#making-all-
other-cache-updates
    refetchQueries: [{ query: QUERY }],
    awaitRefetchQueries: true,
  })

  const onDeleteClick = (id: DeletePartMutationVariables['id']) => {
    if (confirm('Are you sure you want to delete part ' + id + '?')) {
      deletePart({ variables: { id } })
    }
  }

  const thumbnail = (url: string) => {
    if (url.includes('no_image.png')) return url
    const parts = url.split('/')
    parts.splice(3, 0, 'resize=width:100')
    return parts.join('/')
  }

  return (
    <div className="rw-segment rw-table-wrapper-responsive font-inter">
      <table className="rw-table">
        <thead>
          <tr>
```

```
            <th>Id</th>
            <th>Name</th>
            <th>Description</th>
            <th>Available stock</th>
            <th>Image</th>
            <th>Created at</th>
            <th>&nbsp;</th>
          </tr>
        </thead>
        <tbody>
          {parts.map((part) => (
            <tr key={part.id}>
              <td>{truncate(part.id)}</td>
              <td>{truncate(part.name)}</td>
              <td>{truncate(part.description)}</td>
              <td>{truncate(part.availableStock)}</td>
              <td>
                <a href={part.imageUrl} target="_blank" rel="noreferrer">
                  <img
                    alt={`${part.name} thumbnail`}
                    src={thumbnail(part.imageUrl)}
                    style={{ maxWidth: '50px' }}
                  />
                </a>
              </td>
              <td>{timeTag(part.createdAt)}</td>
              <td>
                <nav className="rw-table-actions">
                  <Link
                    to={routes.part({ id: part.id })}
                    title={'Show part ' + part.id + ' detail'}
                    className="rw-button rw-button-small"
                  >
                    Show
                  </Link>
                  <Link
                    to={routes.editPart({ id: part.id })}
                    title={'Edit part ' + part.id}
                    className="rw-button rw-button-small btn-primary"
                  >
                    Edit
                  </Link>
                  <button
                    type="button"
                    title={'Delete part ' + part.id}
                    className="rw-button rw-button-small btn-error"
                    onClick={() => onDeleteClick(part.id)}
                  >
                    Delete
                  </button>
                </nav>
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  )
}
```

```
export default PartsList
```

web/src/components/Part/PartsCell/PartsCell.tsx

```
import { mdiAlert } from '@mdi/js'
import Icon from '@mdi/react'
import type { FindParts } from 'types/graphql'

import type { CellSuccessProps, CellFailureProps } from '@redwoodjs/web'

import Parts from 'src/components/Part/Parts'

export const QUERY = gql`
  query FindParts {
    parts {
      id
      name
      description
      availableStock
      imageUrl
      createdAt
    }
  }
`

export const Loading = () => (
  <div className="flex w-auto justify-center">
    <p className="loading loading-bars loading-lg" />
  </div>
)

export const Empty = () => (
  <div className="flex justify-center">
```

```
    <div className="alert w-auto">
      <p className="text-center font-inter">It&#39;s empty in here...</p>
    </div>
  </div>
)

export const Failure = ({ error }: CellFailureProps) => (
  <div className="flex w-auto justify-center">
    <div className="alert alert-error w-auto">
      <Icon path={mdiAlert} className="h-6 w-6" />
      <p className="font-inter">Error! {error?.message}</p>
    </div>
  </div>
)

export const Success = ({ parts }: CellSuccessProps<FindParts>) => {
  return <Parts parts={parts} />
}
```

web/src/components/PartDetailsCell/PartDetailsCell.tsx

```
import { useState } from 'react'

import { mdiAlert, mdiPlus, mdiMinus } from '@mdi/js'
import { Icon } from '@mdi/react'
import type { FindPartDetailsById } from 'types/graphql'

import type { CellSuccessProps, CellFailureProps } from '@redwoodjs/web'
import { toast } from '@redwoodjs/web/toast'

import { addToBasket } from 'src/lib/basket'

import ToastNotification from '../ToastNotification'

export const QUERY = gql`
  query FindPartDetailsById($id: Int!) {
    part: part(id: $id) {
      id
      name
      description
      availableStock
      imageUrl
      createdAt
    }
  }
`

export const Loading = () => (
  <div className="flex w-auto justify-center">
    <p className="loading loading-bars loading-lg" />
  </div>
)

export const Empty = () => (
  <div className="flex justify-center">
    <div className="alert w-auto">
      <p className="text-center font-inter">It&#39;s empty in here...</p>
    </div>
  </div>
)

export const Failure = ({ error }: CellFailureProps) => (
  <div className="flex w-auto justify-center">
    <div className="alert alert-error w-auto">
      <Icon path={mdiAlert} className="h-6 w-6" />
      <p className="font-inter">Error! {error?.message}</p>
    </div>
  </div>
)

const image = (url: string, size: number) => {
  if (url.includes('no_image.png')) return url
  const parts = url.split('/')
  parts.splice(3, 0, `resize=height:${size}`)
  return parts.join('/')
}

export const Success = ({ part }: CellSuccessProps<FindPartDetailsById>) =>
{
  const [toTake, setToTake] = useState(part.availableStock > 0 ? 1 : 0)
  return (
    <div className="grid grid-cols-1 gap-8 sm:grid-cols-2">
      <div className="col-span-2">
        <h1 className="font-inter text-4xl font-bold">{part.name}</h1>
      </div>

      <div className="order-first col-span-2 sm:order-2 sm:col-span-1">
        <div className="-z-10 flex sm:sticky sm:top-28">
          <div className="h-0 w-full bg-base-100" />
          <img
            alt={part.name}
            src={image(part.imageUrl, 640)}
            className="mb-8 w-full rounded-3xl object-cover sm:mb-0 sm:w-64
md:w-80 lg:w-96 xl:w-[32rem] 2xl:w-[40rem]"
          />
        </div>
      </div>
      <div className="col-span-2 space-y-8 font-inter sm:col-span-1">
        <p className="break-words text-lg">{part.description}</p>
        <div className="divider" />
        <p className="text-xl">
          <strong>Current stock:</strong> {part.availableStock}
        </p>
        <div className="flex space-x-5">
          <div className="join">
            <button
              className={`btn join-item ${
                toTake == 1 || part.availableStock == 0 ? 'btn-disabled' :
''
              }`}
              onClick={() => setToTake(toTake - 1)}
            >
```

```
              <Icon path={mdiMinus} className="h-6 w-6" />
            </button>
            <p className="btn join-item items-center font-inter text-lg">
              {toTake}
            </p>
            <button
              className={`btn join-item ${
                toTake == part.availableStock ? 'btn-disabled' : ''
              }`}
              onClick={() => setToTake(toTake + 1)}
            >
              <Icon path={mdiPlus} className="h-6 w-6" />
            </button>
          </div>
          <button
            className={`btn btn-primary ${toTake == 0 ? 'btn-disabled' :
''}`}
            onClick={() => {
              const newBasket = addToBasket(part, toTake)

              if (typeof newBasket == 'string')
                toast.custom((t) => (
                  <ToastNotification
                    toast={t}
                    type="error"
                    message={newBasket}
                  />
                ))
              else
                toast.custom((t) => (
                  <ToastNotification
```

```
                    toast={t}
                    type="success"
                    message={`Added ${toTake} ${part.name} to basket`}
                  />
                ))
            }}
          >
            Add to basket
          </button>
        </div>
      </div>
    </div>
  )
}
```

web/src/components/PartsCell/PartsCell.tsx

```
/* eslint-disable jsx-a11y/no-noninteractive-tabindex */
import { useState } from 'react'

import {
  mdiAlert,
  mdiChevronRight,
  mdiChevronLeft,
  mdiChevronDoubleRight,
  mdiChevronDoubleLeft,
} from '@mdi/js'
import { Icon } from '@mdi/react'
import type { PartsQuery, SortMethod, SortOrder } from 'types/graphql'

import { Link, routes } from '@redwoodjs/router'
import type { CellSuccessProps, CellFailureProps } from '@redwoodjs/web'

import PartsGridUnit from '../PartsGridUnit/PartsGridUnit'

const POSTS_PER_PAGE = 8

export const beforeQuery = ({ page, sort, order, search }) => {
  page = page ? parseInt(page, 10) : 1
  sort =
    sort &&
    (['createdAt', 'description', 'id', 'name', 'stock'].includes(sort)
      ? sort
      : 'id')
  order =
    order && (['ascending', 'descending'].includes(order) ? order :
'ascending')

  return { variables: { page, sort, order, search } }
}

export const QUERY = gql`
  query PartsQuery(
    $page: Int!
    $sort: SortMethod!
    $order: SortOrder!
    $search: String
  ) {
    partPage(page: $page, sort: $sort, order: $order, searchQuery: $search)
{
      parts {
        id
        name
        description
        availableStock
        imageUrl
        createdAt
      }
      count
      page
      sort
      order
      search
    }
  }
`
```

https://github.com/cy1der/arduino-parts-inventory

```
export const Loading = () => (
  <div className="flex w-auto justify-center">
    <p className="loading loading-bars loading-lg" />
  </div>
)

export const Empty = (
  search: string,
  setSearch: {
    (value: React.SetStateAction<string>): void
    (arg0: string): void
  }
) => (
  <div className="mx-auto w-fit flex-col justify-center space-y-3">
    <div className="flex space-x-3 font-inter">
      <input
        type="search"
        autoComplete="off"
        spellCheck="false"
        placeholder="Search"
        className="input input-bordered w-full max-w-xs"
        value={search}
        onChange={(e) => setSearch(e.target.value)}
      />
      <Link
        className="btn"
        to={routes.home({
          search: search,
        })}
      >
        Search
      </Link>
    </div>
    <div className="alert w-auto">
      <p className="text-center font-inter">It&#39;s empty in here...</p>
    </div>
  </div>
)

export const Failure = ({ error }: CellFailureProps) => (
  <div className="flex w-auto justify-center">
    <div className="alert alert-error w-auto">
      <Icon path={mdiAlert} className="h-6 w-6" />
      <p className="font-inter">Error! {error?.message}</p>
    </div>
  </div>
)

export const Success = ({ partPage }: CellSuccessProps<PartsQuery>) => {
  const sortMethodToText = (sortByText: string) => {
    switch (sortByText as SortMethod) {
      case 'createdAt':
        sortByText = 'Created at'
        break

      case 'description':
        sortByText = 'Description'
        break
      case 'id':
        sortByText = 'ID'
        break

      case 'name':
        sortByText = 'Name'
        break

      case 'stock':
        sortByText = 'Stock'
        break
    }

    return sortByText
  }

  const sortOrderToText = (orderText: string) => {
    switch (orderText as SortOrder) {
      case 'ascending':
        orderText = 'Ascending'
        break

      case 'descending':
        orderText = 'Descending'
        break
    }

    return orderText
  }
```

```
  const [search, setSearch] = useState(partPage.search ?? '')

  if (partPage.count == 0) return Empty(search, setSearch)
  else {
    const sortByText: string = sortMethodToText(partPage.sort)
    const orderText: string = sortOrderToText(partPage.order)

    return (
      <div className="flex flex-col items-center space-y-6">
        <div className="flex space-x-3 font-inter">
          <input
            type="search"
            autoComplete="off"
            spellCheck="false"
            placeholder="Search (case sensitive)"
            className="input input-bordered w-full max-w-xs"
            value={search}
            onChange={(e) => setSearch(e.target.value)}
          />
          <Link
            className="btn"
            to={routes.home({
              page: partPage.page,
              sort: partPage.sort,
              order: partPage.order,
              search: search,
            })}
          >
            Search
          </Link>
        </div>
        <div className="flex space-x-3 font-inter">
          <div className="dropdown">
            <label tabIndex={0} className="btn m-1 normal-case">
              Sort: {sortByText}
            </label>
            <ul
              tabIndex={0}
              className="dropdown-content rounded-box z-[1] mt-3 w-auto space-y-3 bg-base-100 p-3 shadow"
            >
              {['id', 'createdAt', 'description', 'name', 'stock'].map(
                (sort) => (
                  <li key={sort}>
                    <Link
                      className="btn btn-ghost w-full normal-case"
                      to={
                        partPage.search
                          ? routes.home({
                              page: partPage.page,
                              sort,
                              order: partPage.order,
                              search: partPage.search,
                            })
                          : routes.home({
                              page: partPage.page,
                              sort,
                              order: partPage.order,
                            })
                      }
                    >
```

```
                      {sortMethodToText(sort)}
                    </Link>
                  </li>
                )
              )}
            </ul>
          </div>
          <div className="dropdown">
            <label tabIndex={0} className="btn m-1 normal-case">
              Order: {orderText}
            </label>
            <ul
              tabIndex={0}
              className="dropdown-content rounded-box z-[1] mt-3 w-auto
space-y-3 bg-base-100 p-4 shadow"
            >
              {['ascending', 'descending'].map((order) => (
                <li key={order}>
                  <Link
                    className="btn btn-ghost w-full normal-case"
                    to={
                      partPage.search
                        ? routes.home({
                            page: partPage.page,
                            sort: partPage.sort,
                            order,
                            search: partPage.search,
                          })
                        : routes.home({
                            page: partPage.page,
                            sort: partPage.sort,
                            order,
                          })
                    }
                  >
                    {sortOrderToText(order)}
                  </Link>
                </li>
              ))}
            </ul>
          </div>
        </div>
        <div className="grid place-items-center gap-x-6 gap-y-6 md:grid-cols-1 lg:grid-cols-2 xl:grid-cols-3 2xl:grid-cols-4">
          {partPage.parts.map((part) => (
            <PartsGridUnit key={part.id} part={part} />
          ))}
        </div>
        <div className="join">
          <Link
            className={`btn join-item ${
              partPage.page == 1 ? 'btn-disabled' : ''
            }`}
            to={routes.home({
              page: 1,
              sort: partPage.sort,
              order: partPage.order,
            })}
          >
            <Icon path={mdiChevronDoubleLeft} className="h-6 w-6" />
          </Link>
          <Link
```

```
            className={`btn join-item ${
              partPage.page == 1 ? 'btn-disabled' : ''
            }`}
            to={routes.home({
              page: partPage.page - 1,
              sort: partPage.sort,
              order: partPage.order,
            })}
          >
            <Icon path={mdiChevronLeft} className="h-6 w-6" />
          </Link>
          <p className="btn join-item items-center font-inter normal-case">
            Page {partPage.page} of {Math.ceil(partPage.count /
POSTS_PER_PAGE)}
          </p>
          <Link
            className={`btn join-item ${
              partPage.page == Math.ceil(partPage.count / POSTS_PER_PAGE)
                ? 'btn-disabled'
                : ''
            }`}
            to={routes.home({
              page: partPage.page + 1,
              sort: partPage.sort,
              order: partPage.order,
            })}
          >
            <Icon path={mdiChevronRight} className="h-6 w-6" />
          </Link>
          <Link
            className={`btn join-item ${
              partPage.page == Math.ceil(partPage.count / POSTS_PER_PAGE)
                ? 'btn-disabled'
                : ''
            }`}
            to={routes.home({
              page: Math.ceil(partPage.count / POSTS_PER_PAGE),
              sort: partPage.sort,
              order: partPage.order,
            })}
          >
            <Icon path={mdiChevronDoubleRight} className="h-6 w-6" />
          </Link>
        </div>
      </div>
    )
  }
}
```

web/src/components/PartsGridUnit/PartsGridUnit.tsx

```
import type { Part } from 'types/graphql'

import { Link, routes } from '@redwoodjs/router'
import { toast } from '@redwoodjs/web/toast'

import { addToBasket } from 'src/lib/basket'

import ToastNotification from '../ToastNotification'

interface Props {
  part: Part
}

const thumbnail = (url: string) => {
  if (url.includes('no_image.png')) return url
  const parts = url.split('/')
  parts.splice(3, 0, 'resize=width:384')
  return parts.join('/')
}

const PartsGridUnit = ({ part }: Props) => {
  return (
    <Link to={routes.partDetails({ id: part.id })}>
      <div className="card-compact card w-72 space-y-3 bg-base-100 font-inter shadow-xl transition-all duration-200 hover:-translate-y-2 hover:shadow-2xl sm:w-96">
        <figure>
          <img
            className="h-48 object-cover"
            src={thumbnail(part.imageUrl)}
            width={384}
            height={128}
            alt={part.name + ' image'}
          />
        </figure>
        <div className="card-body">
          <h2 className="card-title justify-between">
            {part.name}
            {part.availableStock == 0 ? (
              <div className="badge badge-error">Out of stock</div>
            ) : (
              <div className="badge badge-ghost whitespace-nowrap">
                {part.availableStock + ' left'}
              </div>
            )}
          </h2>
          <p className="overflow-hidden text-ellipsis whitespace-nowrap">
            {part.description}
          </p>
          <div className="card-actions justify-end">
            <button
              className={`btn btn-primary ${
                part.availableStock == 0 ? 'btn-disabled' : ''
              }`}
              onClick={(event) => {
                event.stopPropagation()
                event.preventDefault()

                const newBasket = addToBasket(part, 1)
                if (typeof newBasket == 'string')
                  toast.custom((t) => (
                    <ToastNotification
                      toast={t}
                      type="error"
                      message={newBasket}
                    />
                  ))
                else
                  toast.custom((t) => (
                    <ToastNotification
                      toast={t}
                      type="success"
                      message={`Added 1 ${part.name} to basket`}
                    />
                  ))
              }}
            >
              Add to basket
            </button>
          </div>
        </div>
      </div>
    </Link>
  )
}

export default PartsGridUnit
```

web/src/components/ThemeToggle/ThemeToggle.tsx

```
import { useEffect } from 'react'

import { mdiWeatherSunny, mdiWeatherNight } from '@mdi/js'
import Icon from '@mdi/react'
import { themeChange } from 'theme-change'

const ThemeToggle = () => {
  let isDark = false

  if (typeof window !== 'undefined')
    isDark = localStorage.getItem('theme') === 'dark'

  useEffect(() => {
    themeChange(false)
    return () => {
      themeChange(true)
    }
  }, [])

  return (
    <label className="swap-rotate btn btn-ghost swap w-12 hover:shadow-lg">
      <input
        type="checkbox"
        defaultChecked={isDark}
        data-toggle-theme="light,dark"
      />
      <Icon
        path={mdiWeatherSunny}
        className="swap-off h-8 w-8  text-yellow-500"
      />
      <Icon path={mdiWeatherNight} className="swap-on h-8 w-8 text-blue-500" />
    </label>
  )
}

export default ThemeToggle
```

web/src/components/ToastNotification/ToastNotification.tsx

```
import { mdiCloseCircle, mdiInformation, mdiCheckCircle } from '@mdi/js'
import { Icon } from '@mdi/react'

import { Toast } from '@redwoodjs/web/toast'

type NotificationType = 'success' | 'error' | 'info'

interface Props {
  type: NotificationType
  message: string
  toast: Toast
}

const ToastNotification = ({ type, message, toast }: Props) => (
  <div
    className={`${
      toast.visible
        ? 'duration-200 animate-in slide-in-from-top'
        : 'duration-200 animate-out slide-out-to-top'
    } pointer-events-auto flex w-full max-w-sm items-center rounded-2xl bg-base-100 shadow-lg`}
  >
    <Icon
      className={`m-3 h-8 w-8 ${
        type == 'success'
          ? 'text-success'
          : type == 'error'
          ? 'text-error'
          : 'text-info'
      }`}
      path={
        type == 'success'
          ? mdiCheckCircle
          : type == 'error'
          ? mdiCloseCircle
          : mdiInformation
      }
    />
    <p className="m-3 ml-0 font-inter">{message}</p>
  </div>
)

export default ToastNotification
```

web/src/components/TransactionListItem/TransactionListItem.tsx

```
import { Prisma } from '@prisma/client'
import dayjs from 'dayjs'
import relativeTime from 'dayjs/plugin/relativeTime'
import type { Part, Transaction, TransactionType } from 'types/graphql'

import { useMutation } from '@redwoodjs/web'
import { toast } from '@redwoodjs/web/dist/toast'

import { useAuth } from 'src/auth'

import ToastNotification from '../ToastNotification'

interface Props {
  transaction:
    | {
        id: number
        date: string
        parts: Prisma.JsonValue[]
        type: TransactionType
      }
    | Transaction
  returnButton?: boolean
  admin?: boolean
}

dayjs.extend(relativeTime)
```

https://github.com/cy1der/arduino-parts-inventory

```
export const UPDATE_TRANSACTION_MUTATION = gql`
  mutation UpdateTransactionMutation($id: Int!, $userId: Int!) {
    returnTransaction(id: $id, userId: $userId) {
      id
    }
  }
`

const TransactionListItem = ({
  transaction,
  returnButton = true,
  admin = false,
}: Props) => {
  const elapsedTime = dayjs(transaction.date).fromNow()

  const [returnTransaction, { loading }] = useMutation(
    UPDATE_TRANSACTION_MUTATION,
    {
      onCompleted: () => {
        toast.custom((t) => (
          <ToastNotification
            toast={t}
            type="success"
            message="Transaction updated"
          />
        ))
      },
      onError: (error) => {
        toast.custom((t) => (
          <ToastNotification toast={t} type="error" message={error.message}
/>
```

```
        ))
      },
    }
  )

  const { currentUser } = useAuth()

  return (
    <div className="collapse collapse-arrow max-w-5xl bg-base-200 font-inter">
      <input type="checkbox" />
      <div className="collapse-title text-xl font-medium">
        <div className="items-center justify-between space-x-3 space-y-3 sm:flex sm:space-y-0">
          <p className="overflow-hidden text-ellipsis whitespace-nowrap">
            {transaction.parts.length} items
          </p>
          <div className="w-fit flex-col space-x-3">
            {admin ? (
              <div className="badge badge-primary whitespace-nowrap sm:badge-lg">{`${
                (transaction as Transaction).user?.firstName
              } ${(transaction as Transaction).user?.lastName}`}</div>
            ) : (
              <></>
            )}
            <div
              className={`badge sm:badge-lg ${
                transaction.type == 'out' ? 'badge-error' : 'badge-success'
              }`}
            >
```

```
              {transaction.type.replace(
                transaction.type[0],
                transaction.type[0].toUpperCase()
              )}
            </div>
            <div className="badge whitespace-nowrap bg-base-300 sm:badge-lg">
              {elapsedTime.replace(
                elapsedTime[0],
                elapsedTime[0].toUpperCase()
              )}
            </div>
          </div>
        </div>
      </div>
      <div className="collapse-content space-y-3">
        <ul className="mx-4 list-disc space-y-2">
          {transaction.parts.map((raw) => {
            const part = JSON.parse(raw['part']) as Part
            const quantity = raw['quantity']
            return (
              <li className="list-item" key={part.id}>
                <div className="flex justify-between space-x-3">
                  <p>{part.name}</p>
                  <div className="badge badge-info">
                    <p>Quantity: {quantity}</p>
                  </div>
                </div>
              </li>
            )
          })}
        </ul>
        {transaction.type == 'out' && returnButton ? (
          <button
            className={`btn btn-primary ${loading ? 'btn-disabled' : ''}`}
            disabled={loading}
            onClick={() =>
              returnTransaction({
                variables: { id: transaction.id, userId: currentUser.id },
              })
            }
          >
            Return
          </button>
        ) : (
          <></>
        )}
      </div>
    </div>
  )
}

export default TransactionListItem
```

web/src/components/UserTransactionsCell/UserTransactionsCell.tsx

```
/* eslint-disable jsx-a11y/no-noninteractive-tabindex */
import { mdiAlert } from '@mdi/js'
import { Icon } from '@mdi/react'
import type { UserTransactionsQuery } from 'types/graphql'

import { Link, routes } from '@redwoodjs/router'
import type { CellSuccessProps, CellFailureProps } from '@redwoodjs/web'

import TransactionListItem from
'../TransactionListItem/TransactionListItem'

export const beforeQuery = ({ filter, userId }) => {
  userId = userId ?? null
  filter = filter && ['both', 'in', 'out'].includes(filter) ? filter :
'both'

  return { variables: { filter, userId } }
}

export const QUERY = gql`
  query UserTransactionsQuery(
    $userId: Int!
    $filter: FilterTransactionsByType!
  ) {
    userTransactions(userId: $userId, filter: $filter) {
      transactions {
        id
        date
        parts
        type
      }
      filter
    }
  }
`

export const Loading = () => (
  <div className="flex w-auto justify-center">
    <p className="loading loading-bars loading-lg" />
  </div>
)

export const Empty = () => (
  <div className="flex">
    <div className="alert w-auto">
      <p className="text-center font-inter">It&#39;s empty in here...</p>
    </div>
  </div>
)

export const Failure = ({ error }: CellFailureProps) => (
  <div className="flex w-auto justify-center">
    <div className="alert alert-error w-auto">
      <Icon path={mdiAlert} className="h-6 w-6" />
      <p className="font-inter">Error! {error?.message}</p>
    </div>
  </div>
)
```

```
export const Success = ({
  userTransactions,
}: CellSuccessProps<UserTransactionsQuery>) => {
  if (userTransactions.transactions.length == 0) return Empty()
  return (
    <div className="flex flex-col space-y-6 font-inter">
      <div className="dropdown">
        <label tabIndex={0} className="btn m-1 normal-case">
          Filter:{' '}
          {userTransactions.filter == 'both'
            ? 'None'
            : userTransactions.filter.replace(
                userTransactions.filter[0],
                userTransactions.filter[0].toUpperCase()
              )}
        </label>
        <ul
          tabIndex={0}
          className="dropdown-content rounded-box z-10 mt-3 w-auto space-y-3 bg-base-100 p-3 shadow"
        >
          {['None', 'In', 'Out'].map((filter) => (
            <li key={filter}>
              <Link
                className="btn btn-ghost w-full normal-case"
                to={routes.userTransactions({
                  filter: filter === 'None' ? 'both' : filter.toLowerCase(),
                })}
              >
```

```
                {filter}
              </Link>
            </li>
          ))}
        </ul>
      </div>
      {userTransactions.transactions.map((item, i) => {
        return <TransactionListItem transaction={item} key={i} />
      })}
    </div>
  )
}
```

web/src/index.css

```css
@tailwind base;
@tailwind components;
@tailwind utilities;

.no-scrollbar::-webkit-scrollbar {
  display: none;
}

.no-scrollbar {
  -ms-overflow-style: none;
  scrollbar-width: none;
}
```

web/src/index.html

```
<!DOCTYPE html>
<html data-theme="light" lang="en">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="icon" type="image/png" href="/favicon.png" />

  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link href="https://fonts.googleapis.com/css2?family=Inter:wght@100;200;300;400;500;600;700;800;900&family=Open+Sans:wght@300;400;500;600;700;800&display=swap" rel="stylesheet">
</head>

<body class="no-scrollbar">
  <!-- Please keep this div empty -->
  <div id="redwood-app"></div>
</body>

</html>
```

web/src/layouts/NavbarLayout/NavbarLayout.tsx

```
import { useState } from 'react'

import { mdiChip, mdiMenu, mdiBasket } from '@mdi/js'
import Icon from '@mdi/react'

import { Link, routes } from '@redwoodjs/router'
import { Toaster } from '@redwoodjs/web/toast'

import { useAuth } from 'src/auth'
import AdminMenu from 'src/components/AdminMenu/AdminMenu'
import NavbarAccountIcon from 'src/components/NavbarAccountIcon/NavbarAccountIcon'
import ThemeToggle from 'src/components/ThemeToggle/ThemeToggle'
import { getBasket } from 'src/lib/basket'

type NavBarLayoutProps = {
  children?: React.ReactNode
}

const NavBarLayout = ({ children }: NavBarLayoutProps) => {
  const { hasRole, isAuthenticated } = useAuth()
  const [basket] = useState(getBasket())

  return (
    <>
      <Toaster />
      <div className="navbar sticky top-0 z-50 bg-base-100 shadow-lg">
        <div className="justify-start space-x-3">
          <Icon
```

```
            path={mdiChip}
            className="ml-3 hidden h-10 text-logo md:block"
          />
          <Link
            to={routes.home()}
            className="btn btn-ghost items-center hover:shadow-lg"
          >
            <p className="font-inter text-xl normal-case tracking-tight">
              Parts Inventory
            </p>
          </Link>
        </div>
        <div className="ml-auto justify-end space-x-3">
          <ul className="relative hidden items-center space-x-3 lg:flex">
            {isAuthenticated ? (
              <li>
                <Link
                  to={routes.userTransactions()}
                  className="btn btn-ghost font-inter hover:shadow-lg"
                >
                  Transactions
                </Link>
              </li>
            ) : (
              <></>
            )}
          </ul>
          <ThemeToggle />
          <Link
            to={routes.basket()}
            className="items-center btn btn-ghost hidden hover:shadow-lg lg:flex"
          >
            <div className="indicator">
              {basket.length > 0 ? (
                <span className="badge indicator-item badge-primary font-inter">
                  {basket.length}
                </span>
              ) : (
                <></>
              )}

              <Icon path={mdiBasket} className="h-8 w-8 text-base-content" />
            </div>
          </Link>
          <NavbarAccountIcon mobile={false} className="hidden lg:block" />
          <AdminMenu mobile={false} className="hidden lg:block" />
          <div className="lg:hidden">
            <input
              id="mobile-menu-drawer"
              type="checkbox"
              className="drawer-toggle"
              defaultChecked={false}
            />
            <div className="drawer-content">
              <label
                htmlFor="mobile-menu-drawer"
                className="btn btn-ghost drawer-button hover:shadow-lg"
              >
```

```
                <Icon path={mdiMenu} className="h-8 w-8" />
              </label>
            </div>
            <div className="drawer-side z-50">
              <label htmlFor="mobile-menu-drawer" className="drawer-
overlay">
                <br />
              </label>
              <ul className="min-h-full w-80 space-y-3 bg-base-100 p-3
text-base-content shadow-lg">
                <li>
                  <div className="flex items-center justify-between">
                    {hasRole('admin') ? (
                      <AdminMenu mobile={true} />
                    ) : (
                      <Icon path={mdiChip} className="ml-3 h-10 text-
logo" />
                    )}
                    <Link
                      to={routes.home()}
                      className="btn btn-ghost items-center hover:shadow-
lg"
                    >
                      <p className="font-inter text-xl normal-case
tracking-tight">
                        Parts Inventory
                      </p>
                    </Link>
                    <NavbarAccountIcon mobile={true} />
                  </div>
                </li>
                <li>
```

```
                  <Link
                    to={routes.basket()}
                    className="btn btn-ghost w-full hover:shadow-lg"
                  >
                    <p className="font-inter text-base">Basket</p>
                  </Link>
                </li>
                {isAuthenticated ? (
                  <li>
                    <Link
                      to={routes.userTransactions()}
                      className="btn btn-ghost w-full hover:shadow-lg"
                    >
                      <p className="font-inter text-base">Transactions</p>
                    </Link>
                  </li>
                ) : (
                  <></>
                )}
              </ul>
            </div>
          </div>
        </div>
      </div>
      <main className="m-3">{children}</main>
    </>
  )
}

export default NavBarLayout
```

web/src/layouts/ScaffoldLayout/ScaffoldLayout.tsx

```
import { mdiHome } from '@mdi/js'
import Icon from '@mdi/react'

import { Link, routes } from '@redwoodjs/router'
import { Toaster } from '@redwoodjs/web/toast'

type LayoutProps = {
  title: string
  titleTo: string
  buttonLabel?: string
  buttonTo?: string
  children: React.ReactNode
}

const ScaffoldLayout = ({
  title,
  titleTo,
  buttonLabel,
  buttonTo,
  children,
}: LayoutProps) => {
  return (
    <div className="rw-scaffold">
      <Toaster />
      <header className="rw-header">
        <div className="space-x-3">
          <Link to={routes.home()} className="btn btn-ghost hover:shadow-lg">
            <Icon path={mdiHome} className="h-8 w-8" />
          </Link>
          <h1 className="rw-heading rw-heading-primary rw-button btn-ghost
normal-case">
            <Link to={routes[titleTo]()}>{title}</Link>
          </h1>
        </div>
        {buttonLabel && buttonTo ? (
          <Link to={routes[buttonTo]()} className="rw-button btn-success">
            <div className="rw-button-icon">+</div> {buttonLabel}
          </Link>
        ) : (
          <></>
        )}
      </header>
      <main className="rw-main">{children}</main>
    </div>
  )
}

export default ScaffoldLayout
```

web/src/lib/basket.ts

```
import { Part } from 'types/graphql'

const BASKET_KEY = 'basket'

export interface BasketItem {
  part: Part
  quantity: number
}

export const getBasket = (): BasketItem[] => {
  const basketRaw = localStorage.getItem(BASKET_KEY)
  const basket: BasketItem[] = basketRaw ? JSON.parse(basketRaw) : []

  return basket
}

export const setBasket = (newBasket: BasketItem[]): BasketItem[] => {
  localStorage.setItem(BASKET_KEY, JSON.stringify(newBasket))
  return getBasket()
}

export const addToBasket = (
  part: Part,
  quantity: number
): BasketItem[] | string => {
  const basket = getBasket()
  const existingPartIndex = basket.findIndex((item) => item.part.id == part.id)
  if (existingPartIndex !== -1) {
    const basketPart = basket[existingPartIndex]

    if (basketPart.quantity + quantity <= basketPart.part.availableStock)
      basket[existingPartIndex].quantity += quantity
    else return `Cannot exceed number of items left ($
{part.availableStock})`
  } else basket.push({ part, quantity })

  localStorage.setItem(BASKET_KEY, JSON.stringify(basket))

  return basket
}

export const removeFromBasket = (index: number): BasketItem[] | string => {
  const basket = getBasket()

  if (index >= 0 && index < basket.length) basket.splice(index, 1)
  else return 'Error: index out of bounds'

  localStorage.setItem(BASKET_KEY, JSON.stringify(basket))

  return basket
}

export const clearBasket = (): BasketItem[] => {
  localStorage.removeItem(BASKET_KEY)

  return []
}
```

web/src/lib/formatters.tsx

```
import React from 'react'

import humanize from 'humanize-string'

const MAX_STRING_LENGTH = 150

export const formatEnum = (values: string | string[] | null | undefined) => {
  let output = ''

  if (Array.isArray(values)) {
    const humanizedValues = values.map((value) => humanize(value))
    output = humanizedValues.join(', ')
  } else if (typeof values === 'string') {
    output = humanize(values)
  }

  return output
}

export const jsonDisplay = (obj: unknown) => {
  return (
    <pre>
      <code>{JSON.stringify(obj, null, 2)}</code>
    </pre>
  )
}

export const truncate = (value: string | number) => {
  let output = value?.toString() ?? ''

  if (output.length > MAX_STRING_LENGTH) {
    output = output.substring(0, MAX_STRING_LENGTH) + '...'
  }

  return output
}

export const jsonTruncate = (obj: unknown) => {
  return truncate(JSON.stringify(obj, null, 2))
}

export const timeTag = (dateTime?: string) => {
  let output: string | JSX.Element = ''

  if (dateTime) {
    output = (
      <time dateTime={dateTime} title={dateTime}>
        {new Date(dateTime).toUTCString()}
      </time>
    )
  }

  return output
}

export const checkboxInputTag = (checked: boolean) => {
  return <input type="checkbox" checked={checked} disabled />
}
```

web/src/pages/AdminTransactionsPage/AdminTransactionsPage.tsx

```
import { mdiAlert } from '@mdi/js'
import Icon from '@mdi/react'
import { FilterTransactionsByType } from 'types/graphql'

import { MetaTags } from '@redwoodjs/web'

import { useAuth } from 'src/auth'
import AdminTransactionsCell from 'src/components/AdminTransactionsCell'

interface Props {
  filter: FilterTransactionsByType
}

const AdminTransactionsPage = ({ filter = 'both' }: Props) => {
  const { isAuthenticated, hasRole } = useAuth()

  return (
    <>
      <MetaTags
        title="Admin Transactions"
        description="Admin Transactions page"
      />

      <div className="m-8">
        <p className="mb-8 font-inter text-3xl font-bold">Transactions</p>
        {isAuthenticated ? (
          hasRole('admin') ? (
            <AdminTransactionsCell filter={filter} />
          ) : (
            <div className="flex w-auto justify-center">
              <div className="alert alert-error w-auto">
                <Icon path={mdiAlert} className="h-6 w-6" />
                <p className="font-inter">You must be admin to do that.</p>
              </div>
            </div>
          )
        ) : (
          <div className="flex w-auto justify-center">
            <div className="alert alert-error w-auto">
              <Icon path={mdiAlert} className="h-6 w-6" />
              <p className="font-inter">You must be logged in to do
that.</p>
            </div>
          </div>
        )}
      </div>
    </>
  )
}

export default AdminTransactionsPage
```

web/src/pages/BasketPage/BasketPage.tsx

```
import { MetaTags } from '@redwoodjs/web'

import { BasketCell } from 'src/components/BasketCell/BasketCell'

const BasketPage = () => {
  return (
    <>
      <MetaTags title="Basket" description="Basket page" />

      <div className="m-8">
        <h1 className="mb-8 font-inter text-3xl font-bold">Basket</h1>
        <BasketCell />
      </div>
    </>
  )
}

export default BasketPage
```

web/src/pages/FatalErrorPage/FatalErrorPage.tsx

```
// This import will be automatically removed when building for production
import { DevFatalErrorPage } from '@redwoodjs/web/dist/components/DevFatalErrorPage'

export default DevFatalErrorPage ||
  (() => (
    <main>
      <style
        dangerouslySetInnerHTML={{
          __html: `
              html, body {
                margin: 0;
              }
              html * {
                box-sizing: border-box;
              }
              main {
                display: flex;
                align-items: center;
                font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto, "Helvetica Neue", Arial, "Noto Sans", sans-serif;
                text-align: center;
                background-color: #E2E8F0;
                height: 100vh;
              }
              section {
                background-color: white;
                border-radius: 0.25rem;
                width: 32rem;
                padding: 1rem;
                margin: 0 auto;
                box-shadow: 0 1px 3px 0 rgba(0, 0, 0, 0.1), 0 1px 2px 0 rgba(0, 0, 0, 0.06);
              }
              h1 {
                font-size: 2rem;
                margin: 0;
                font-weight: 500;
                line-height: 1;
                color: #2D3748;
              }
            `,
        }}
      />
      <section>
        <h1>
          <span>Something went wrong</span>
        </h1>
      </section>
    </main>
  ))
```

web/src/pages/ForgotPasswordPage/ForgotPasswordPage.tsx

```
import { useEffect, useRef } from 'react'

import { Form, TextField, Submit, FieldError } from '@redwoodjs/forms'
import { navigate, routes } from '@redwoodjs/router'
import { MetaTags } from '@redwoodjs/web'
import { toast, Toaster } from '@redwoodjs/web/toast'

import { useAuth } from 'src/auth'
import ToastNotification from 'src/components/ToastNotification'

const ForgotPasswordPage = () => {
  const { isAuthenticated, forgotPassword } = useAuth()

  useEffect(() => {
    if (isAuthenticated) {
      navigate(routes.home())
    }
  }, [isAuthenticated])

  const emailRef = useRef<HTMLInputElement>(null)
  useEffect(() => {
    emailRef?.current?.focus()
  }, [])

  const onSubmit = async (data: { email: string }) => {
    const response = await forgotPassword(data.email.toLowerCase())
    if (response.error) {
      toast.custom((t) => (
        <ToastNotification toast={t} type="error"
message={response.error} />
      ))
    } else {
      // The function `forgotPassword.handler` in api/src/functions/auth.js
has
      // been invoked, let the user know how to get the link to reset their
      // password (sent in email, perhaps?)
      toast.custom((t) => (
        <ToastNotification
          toast={t}
          type="success"
          message={`A link to reset your password was sent to $
{response.email}`}
        />
      ))
      navigate(routes.login())
    }
  }

  return (
    <>
      <MetaTags title="Forgot Password" />

      <main className="rw-main">
        <Toaster toastOptions={{ className: 'rw-toast', duration:
6000 }} />
        <div className="rw-scaffold rw-login-container">
          <div className="rw-segment">
            <header className="rw-segment-header">
```

```
              <h2 className="rw-heading rw-heading-primary">Forgot
Password</h2>
            </header>

            <div className="rw-segment-main">
              <div className="rw-form-wrapper">
                <Form onSubmit={onSubmit} className="rw-form-wrapper">
                  <div className="text-left">
                    <TextField
                      name="email"
                      placeholder="Email"
                      className="rw-input mb-3 min-w-full"
                      errorClassName="rw-input rw-input-error min-w-full"
                      ref={emailRef}
                      validation={{
                        required: {
                          value: true,
                          message: 'Required',
                        },
                        pattern: {
                          value: new RegExp(
                            /^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-]+)(\.[a-
zA-Z]{2,5}){1,2}$/
                          ),
                          message: 'Email is not valid',
                        },
                      }}
                    />

                    <FieldError name="email" className="rw-field-error pb-
3" />
```

```
                  </div>

                  <div className="rw-button-group">
                    <Submit className="rw-button
btn-primary">Submit</Submit>
                  </div>
                </Form>
              </div>
            </div>
          </div>
        </div>
      </main>
    </>
  )
}

export default ForgotPasswordPage
```

web/src/pages/HomePage/HomePage.tsx

```
import { SortMethod, SortOrder } from 'types/graphql'

import { MetaTags } from '@redwoodjs/web'

import PartsCell from 'src/components/PartsCell'

interface Props {
  page: number
  sort: SortMethod
  order: SortOrder
  search?: string
}

const HomePage = ({
  page = 1,
  sort = 'id',
  order = 'ascending',
  search,
}: Props) => {
  return (
    <>
      <MetaTags title="Home" description="Home page" />

      <div className="my-16 text-center font-inter md:my-24">
        <h1 className="text-4xl font-extrabold tracking-wide md:text-6xl">
          Parts Inventory
        </h1>
        <p className="pt-4 text-xl">Only take what you need</p>
      </div>
      <PartsCell page={page} sort={sort} order={order} search={search} />
    </>
  )
}

export default HomePage
```

web/src/pages/LoginPage/LoginPage.tsx

```
import { useRef, useEffect } from 'react'

import {
  Form,
  TextField,
  PasswordField,
  Submit,
  FieldError,
} from '@redwoodjs/forms'
import { Link, navigate, routes } from '@redwoodjs/router'
import { MetaTags } from '@redwoodjs/web'
import { toast, Toaster } from '@redwoodjs/web/toast'

import { useAuth } from 'src/auth'
import ToastNotification from 'src/components/ToastNotification'

const LoginPage = () => {
  const { isAuthenticated, logIn } = useAuth()

  useEffect(() => {
    if (isAuthenticated) {
      navigate(routes.home())
    }
  }, [isAuthenticated])

  const emailRef = useRef<HTMLInputElement>(null)
  useEffect(() => {
    emailRef.current?.focus()
  }, [])
```

https://github.com/cy1der/arduino-parts-inventory

```
  const onSubmit = async (data: Record<string, string>) => {
    const response = await logIn({
      username: data.email.toLowerCase(),
      password: data.password,
    })

    if (response.message) {
      toast(response.message)
    } else if (response.error) {
      toast.custom((t) => (
        <ToastNotification toast={t} type="error"
message={response.error} />
      ))
    } else {
      toast.custom((t) => (
        <ToastNotification toast={t} type="success" message="Welcome back!"
/>
      ))
    }
  }

  return (
    <>
      <MetaTags title="Login" />

      <main className="rw-main">
        <Toaster toastOptions={{ className: 'rw-toast', duration:
6000 }} />
        <div className="rw-scaffold rw-login-container">
          <div className="rw-segment">
```

```
            <header className="rw-segment-header">
              <h2 className="rw-heading rw-heading-primary">Login</h2>
            </header>

            <div className="rw-segment-main">
              <div className="rw-form-wrapper">
                <Form onSubmit={onSubmit} className="rw-form-wrapper">
                  <TextField
                    name="email"
                    placeholder="Email"
                    className="rw-input mb-3 min-w-full"
                    errorClassName="rw-input rw-input-error min-w-full"
                    ref={emailRef}
                    validation={{
                      required: {
                        value: true,
                        message: 'Required',
                      },
                    }}
                  />

                  <FieldError name="email" className="rw-field-error pb-3" />

                  <PasswordField
                    name="password"
                    placeholder="Password"
                    className="rw-input mb-3 min-w-full"
                    errorClassName="rw-input rw-input-error min-w-full"
                    autoComplete="current-password"
                    validation={{
                      required: {
                        value: true,
                        message: 'Required',
                      },
                    }}
                  />

                  <FieldError name="password" className="rw-field-error" />

                  <div className="rw-forgot-link">
                    <Link
                      to={routes.forgotPassword()}
                      className="rw-forgot-link"
                    >
                      Forgot Password?
                    </Link>
                  </div>

                  <div className="rw-button-group">
                    <Submit className="rw-button btn-primary
">Login</Submit>
                  </div>
                </Form>
              </div>
            </div>
          </div>
          <div className="rw-login-link">
            <span className="font-inter text-base-content">
              Don&apos;t have an account?
            </span>{' '}
            <Link to={routes.signup()} className="rw-link">
```

```
              Sign up!
            </Link>
          </div>
        </div>
      </main>
    </>
  )
}

export default LoginPage
```

web/src/pages/NotFoundPage/NotFoundPage.tsx

```
import { Link, routes } from '@redwoodjs/router'

export default () => (
  <div className="flex min-h-screen min-w-max items-center justify-center
p-3">
    <div className="mockup-phone font-inter">
      <div className="camera"></div>
      <div className="display">
        <div className="artboard phone-1 bg-base-100 p-2">
          <div className="pt-6">{message1('W-what, where am I?')}</div>
          {message2('Leave, now.')}
          {message1('What, why?')}
          {message2("You're being watched, this is the 404 zone.")}
          {message1('I better get going then...')}
          <div className="flex h-56 items-center justify-center">
            <Link to={routes.home()} className="btn">
              Back to safety
            </Link>
          </div>
        </div>
      </div>
    </div>
  </div>
)

const message1 = (message: string) => (
  <div className="chat chat-end">
    <div className="chat-bubble chat-bubble-info">{message}</div>
  </div>
```

https://github.com/cy1der/arduino-parts-inventory

```
)


const message2 = (message: string) => (
  <div className="chat chat-start">
    <div className="chat-bubble bg-gray-200 text-base-content">{message}</div>
  </div>
)
```

web/src/pages/Part/EditPartPage/EditPartPage.tsx

```
import EditPartCell from 'src/components/Part/EditPartCell'

type PartPageProps = {
  id: number
}

const EditPartPage = ({ id }: PartPageProps) => {
  return <EditPartCell id={id} />
}

export default EditPartPage
```

web/src/pages/Part/NewPartPage/NewPartPage.tsx

```
import NewPart from 'src/components/Part/NewPart'

const NewPartPage = () => {
  return <NewPart />
}

export default NewPartPage
```

web/src/pages/Part/PartPage/PartPage.tsx

```
import PartCell from 'src/components/Part/PartCell'

type PartPageProps = {
  id: number
}

const PartPage = ({ id }: PartPageProps) => {
  return <PartCell id={id} />
}

export default PartPage
```

web/src/pages/Part/PartsPage/PartsPage.tsx

```
import PartsCell from 'src/components/Part/PartsCell'

const PartsPage = () => {
  return <PartsCell />
}

export default PartsPage
```

web/src/pages/PartPage/PartPage.tsx

```
import { MetaTags } from '@redwoodjs/web'

import PartDetailsCell from 'src/components/PartDetailsCell'

interface Props {
  id: number
}

const PartPage = ({ id }: Props) => {
  return (
    <>
      <MetaTags title="Part" description="Part page" />

      <div className="m-8">
        <PartDetailsCell id={id} />
      </div>
    </>
  )
}

export default PartPage
```

web/src/pages/ResetPasswordPage/ResetPasswordPage.tsx

```
import { useEffect, useRef, useState } from 'react'

import { Form, PasswordField, Submit, FieldError } from '@redwoodjs/forms'
import { navigate, routes } from '@redwoodjs/router'
import { MetaTags } from '@redwoodjs/web'
import { toast, Toaster } from '@redwoodjs/web/toast'

import { useAuth } from 'src/auth'
import ToastNotification from 'src/components/ToastNotification'

const ResetPasswordPage = ({ resetToken }: { resetToken: string }) => {
  const { isAuthenticated, reauthenticate, validateResetToken,
resetPassword } =
    useAuth()
  const [enabled, setEnabled] = useState(true)

  useEffect(() => {
    if (isAuthenticated) {
      navigate(routes.home())
    }
  }, [isAuthenticated])

  useEffect(() => {
    const validateToken = async () => {
      const response = await validateResetToken(resetToken)
      if (response.error) {
        setEnabled(false)
        toast.custom((t) => (
          <ToastNotification toast={t} type="error"
message={response.error} />
        ))
      } else {
        setEnabled(true)
      }
    }
    validateToken()
  }, [resetToken, validateResetToken])

  const passwordRef = useRef<HTMLInputElement>(null)
  useEffect(() => {
    passwordRef.current?.focus()
  }, [])

  const onSubmit = async (data: Record<string, string>) => {
    const response = await resetPassword({
      resetToken,
      password: data.password,
    })

    if (response.error) {
      toast.custom((t) => (
        <ToastNotification toast={t} type="error"
message={response.error} />
      ))
    } else {
      toast.custom((t) => (
        <ToastNotification
          toast={t}
          type="success"
          message="Password changed!"
        />
      ))
      await reauthenticate()
      navigate(routes.login())
    }
  }

  return (
    <>
      <MetaTags title="Reset Password" />

      <main className="rw-main">
        <Toaster toastOptions={{ className: 'rw-toast', duration:
6000 }} />
        <div className="rw-scaffold rw-login-container">
          <div className="rw-segment">
            <header className="rw-segment-header">
              <h2 className="rw-heading rw-heading-primary">Reset
Password</h2>
            </header>

            <div className="rw-segment-main">
              <div className="rw-form-wrapper">
                <Form onSubmit={onSubmit} className="rw-form-wrapper">
                  <div className="text-left">
                    <PasswordField
                      name="password"
                      placeholder="New password"
                      autoComplete="new-password"
                      className="rw-input mb-3 min-w-full"
                      errorClassName="rw-input rw-input-error min-w-full"
                      disabled={!enabled}
                      ref={passwordRef}
                      validation={{
                        required: {
                          value: true,
                          message: 'Required',
                        },
                      }}
                    />

                    <FieldError
                      name="password"
                      className="rw-field-error pb-3"
                    />
                  </div>

                  <div className="rw-button-group">
                    <Submit
                      className="rw-button btn-primary"
                      disabled={!enabled}
                    >
                      Submit
                    </Submit>
                  </div>
                </Form>
              </div>
            </div>
          </div>
        </div>
```

```
      </main>
    </>
  )
}

export default ResetPasswordPage
```

web/src/pages/SignupPage/SignupPage.tsx

```
import { useRef } from 'react'
import { useEffect } from 'react'

import {
  Form,
  TextField,
  PasswordField,
  FieldError,
  Submit,
} from '@redwoodjs/forms'
import { Link, navigate, routes } from '@redwoodjs/router'
import { MetaTags } from '@redwoodjs/web'
import { toast, Toaster } from '@redwoodjs/web/toast'

import { useAuth } from 'src/auth'
import ToastNotification from 'src/components/ToastNotification'

const SignupPage = () => {
  const { isAuthenticated, signUp } = useAuth()

  useEffect(() => {
    if (isAuthenticated) {
      navigate(routes.home())
    }
  }, [isAuthenticated])

  // focus on name box on page load
  const nameRef = useRef<HTMLInputElement>(null)
  useEffect(() => {
    nameRef.current?.focus()
  }, [])

  const onSubmit = async (data: Record<string, string>) => {
    const response = await signUp({
      username: data.email.toLowerCase(),
      password: data.password,
      firstName: data.firstName,
      lastName: data.lastName,
    })

    if (response.message) toast(response.message)
    else if (response.error)
      toast.custom((t) => (
        <ToastNotification toast={t} type="error"
message={response.error} />
      ))
    // user is signed in automatically
    else
      toast.custom((t) => (
        <ToastNotification toast={t} type="success" message="Welcome!" />
      ))
  }

  return (
    <>
      <MetaTags title="Signup" />

      <main>
        <Toaster toastOptions={{ className: 'rw-toast', duration:
6000 }} />
```

```
        <div className="rw-scaffold rw-login-container">
          <div className="rw-segment">
            <header className="rw-segment-header">
              <h2 className="rw-heading rw-heading-primary">Sign up</h2>
            </header>

            <div className="rw-segment-main">
              <div className="rw-form-wrapper">
                <Form onSubmit={onSubmit} className="rw-form-wrapper">
                  <div className="mb-3 flex justify-between space-x-3">
                    <div>
                      <TextField
                        placeholder="First Name"
                        name="firstName"
                        className="rw-input"
                        errorClassName="rw-input rw-input-error"
                        ref={nameRef}
                        validation={{
                          required: {
                            value: true,
                            message: 'Required',
                          },
                        }}
                      />
                      <FieldError name="firstName" className="rw-field-
error" />
                    </div>

                    <div>
                      <TextField
                        name="lastName"
                        placeholder="Last Name"
                        className="rw-input"
                        errorClassName="rw-input rw-input-error"
                        validation={{
                          required: {
                            value: true,
                            message: 'Required',
                          },
                        }}
                      />
                      <FieldError name="lastName" className="rw-field-error" />
                    </div>
                  </div>

                  <TextField
                    name="email"
                    placeholder="Email"
                    className="rw-input mb-3 min-w-full"
                    errorClassName="rw-input rw-input-error min-w-full"
                    validation={{
                      required: {
                        value: true,
                        message: 'Required',
                      },
                      pattern: {
                        value: new RegExp(
                          /^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-]+)(\.[a-zA-Z]{2,5}){1,2}$/
                        ),
                        message: 'Email is not valid',
                      },
                    }}
                    inputMode="email"
                  />
                  <FieldError name="email" className="rw-field-error pb-3" />

                  <PasswordField
                    name="password"
                    placeholder="Password"
                    className="rw-input min-w-full"
                    errorClassName="rw-input rw-input-error min-w-full"
                    autoComplete="current-password"
                    validation={{
                      required: {
                        value: true,
                        message: 'Required',
                      },
                    }}
                  />
                  <FieldError name="password" className="rw-field-error" />

                  <div className="rw-button-group">
                    <Submit className="rw-button btn-primary">Sign Up</Submit>
                  </div>
                </Form>
              </div>
            </div>
          </div>
          <div className="rw-login-link">
```

```
            <span className="font-inter text-base-content">
              Already have an account?
            </span>{' '}
            <Link to={routes.login()} className="rw-link">
              Log in!
            </Link>
          </div>
        </div>
      </main>
    </>
  )
}

export default SignupPage
```

web/src/pages/TransactionsPage/TransactionsPage.tsx

```
import { mdiAlert } from '@mdi/js'
import Icon from '@mdi/react'
import { FilterTransactionsByType } from 'types/graphql'

import { MetaTags } from '@redwoodjs/web'

import { useAuth } from 'src/auth'
import UserTransactionsCell from 'src/components/UserTransactionsCell'

interface Props {
  filter: FilterTransactionsByType
}

const TransactionsPage = ({ filter = 'both' }: Props) => {
  const { isAuthenticated, currentUser } = useAuth()
  return (
    <>
      <MetaTags title="Transactions" description="Transactions page" />

      <div className="m-8">
        <h1 className="mb-8 font-inter text-3xl font-bold">
          Transaction History
        </h1>
        {isAuthenticated ? (
          <UserTransactionsCell
            filter={filter}
            userId={currentUser ? currentUser.id : -1}
          />
        ) : (
          <div className="flex w-auto justify-center">
            <div className="alert alert-error w-auto">
              <Icon path={mdiAlert} className="h-6 w-6" />
              <p className="font-inter">You must be logged in to do that.</p>
            </div>
          </div>
        )}
      </div>
    </>
  )
}

export default TransactionsPage
```

web/src/scaffold.css

```
.rw-scaffold h1,
.rw-scaffold h2 {
  @apply m-0;
}
.rw-header {
  @apply navbar items-center justify-between space-x-3 bg-base-100 shadow-lg;
}
.rw-main {
  @apply m-4;
}
.rw-segment {
  @apply w-full overflow-hidden rounded-lg ;
  scrollbar-color: theme('colors.zinc.400') transparent;
}
.rw-segment::-webkit-scrollbar {
  height: initial;
}
.rw-segment::-webkit-scrollbar-track {
  @apply rounded-b-[10px] rounded-t-none border-0 border-t border-solid border-gray-200 bg-transparent p-[2px];
}
.rw-segment::-webkit-scrollbar-thumb {
  @apply rounded-full border-[3px] border-solid border-transparent bg-zinc-400 bg-clip-content;
}
.rw-segment-header {
  @apply bg-base-300 px-4 py-3;
}
```

https://github.com/cy1der/arduino-parts-inventory

```
.rw-segment-main {
  @apply bg-base-200 p-4;
}
.rw-link {
  @apply font-inter link link-hover text-base-content;
}
.rw-forgot-link {
  @apply text-right font-inter text-xs text-base-content link link-hover;
}
.rw-heading {
  @apply font-inter font-semibold text-base-content;
}
.rw-heading.rw-heading-primary {
  @apply text-xl;
}
.rw-heading.rw-heading-secondary {
  @apply text-sm;
}
.rw-heading .rw-link {
  @apply text-gray-600 no-underline;
}
.rw-heading .rw-link:hover {
  @apply text-gray-900 underline;
}
.rw-cell-error {
  @apply text-sm font-semibold;
}
.rw-form-wrapper {
  @apply text-sm;
}
```

```
.rw-cell-error,
.rw-form-error-wrapper {
  @apply my-4 rounded border border-red-100 bg-red-50 p-4 text-red-600;
}
.rw-form-error-title {
  @apply m-0 font-semibold;
}
.rw-form-error-list {
  @apply mt-2 list-inside list-disc;
}
.rw-button {
  @apply btn font-inter hover:shadow-lg;
}
.rw-button.rw-button-small {
  @apply btn-sm;
}
.rw-button-icon {
  @apply mr-1 text-xl leading-5;
}
.rw-button-group {
  @apply mx-2 my-3 flex justify-center;
}
.rw-button-group .rw-button {
  @apply mx-1;
}
.rw-form-wrapper .rw-button-group {
  @apply mt-8;
}
.rw-label {
  @apply mt-6 block text-left font-inter font-semibold text-gray-600;
```

```
}
.rw-label.rw-label-error {
  @apply text-red-600;
}
.rw-input {
  @apply input input-bordered w-full max-w-xs font-inter text-base-content;
}
.rw-textarea {
  @apply textarea textarea-bordered text-base font-inter max-w-xs w-full text-base-content;
}
.rw-check-radio-items {
  @apply flex justify-items-center;
}
.rw-check-radio-item-none {
  @apply text-gray-600;
}
.rw-input[type='checkbox'],
.rw-input[type='radio'] {
  @apply ml-0 mr-1 mt-1 inline w-4;
}
.rw-input-error {
  @apply input-error;
}
.rw-input-error:focus {
  @apply border-red-600 outline-none;
  box-shadow: 0 0 5px #c53030;
}
.rw-field-error {
  @apply my-3 block text-left font-inter text-xs font-semibold text-red-600;
```

```
}
.rw-table-wrapper-responsive {
  @apply overflow-x-auto;
}
.rw-table-wrapper-responsive .rw-table {
  min-width: 48rem;
}
.rw-table {
  @apply w-full text-sm;
}
.rw-table th,
.rw-table td {
  @apply p-3;
}
.rw-table td {
  @apply bg-white text-gray-900;
}
.rw-table tr:nth-child(odd) td,
.rw-table tr:nth-child(odd) th {
  @apply bg-gray-50;
}
.rw-table thead tr {
  @apply bg-gray-200 text-gray-600;
}
.rw-table th {
  @apply text-left font-semibold;
}
.rw-table thead th {
  @apply text-left;
}
```

```
.rw-table tbody th {
  @apply text-right;
}
@media (min-width: 768px) {
  .rw-table tbody th {
    @apply w-1/5;
  }
}
.rw-table tbody tr {
  @apply border-t border-gray-200;
}
.rw-table input {
  @apply ml-0;
}
.rw-table-actions {
  @apply flex h-4 items-center justify-end pr-1 space-x-2;
}
.rw-text-center {
  @apply text-center;
}
.rw-login-container {
  @apply mx-auto my-16 flex w-96 flex-wrap items-center justify-center bg-base-100;
}
.rw-login-container .rw-form-wrapper {
  @apply w-full text-center;
}
.rw-login-link {
  @apply mt-4 w-full text-center text-sm text-gray-600;
}
.rw-webauthn-wrapper {
  @apply mx-4 mt-6 leading-6;
}
.rw-webauthn-wrapper h2 {
  @apply mb-4 text-xl font-bold;
}
```