

Dauids book club Documentation



David's
Book Club

Table of contents

Project definition.....	2
Overview.....	2
Technical requirements.....	2
Documentation.....	2
Deadline.....	2
Frontend.....	3
Frontend overview.....	3
Global states.....	3
Cart.....	3
Frontend diagrams.....	4
Flowcharts.....	4
User stories.....	8
Wireframes.....	9
Use Case Diagram.....	20
Use case Specifications:.....	21
Frontend dependencies.....	28
Backend.....	30
Backend overview.....	30
API endpoints.....	30
Get requests.....	30
Post requests.....	32
Hashing and encryption.....	35
Backend diagrams.....	36
Class diagram.....	36
Backend dependencies.....	37
Database.....	38
Database overview.....	38
Database users and permissions.....	38
Stored procedures.....	39
Database diagrams.....	41
ERD.....	41
RDS.....	42
General diagrams.....	43
Sequence diagram:.....	43
Scrum.....	53
Sprints.....	53
Conclusion.....	55
Overall.....	55
Gantt diagram.....	55
Github.....	55

Project definition

Overview

Our team has been approached by a client, David. He has expressed his desire to establish his own ecommerce platform specifically for the purpose of selling his books.

David has been inspired by a webshop called “Bog & idé” and suggests that we visit their website to pull some ideas.

Technical requirements

David has specified that the platform should be built using a MySQL database. However, he has not made any specific requests regarding the choice of backend or frontend language/framework.

Documentation

A key requirement from David is the provision of comprehensive documentation and diagrams. This will ensure that he gains a thorough understanding of his new ecommerce system.

He specifically requested the following diagrams:

- Class diagram
- Sequence diagram
- Flow chart
- User stories
- Use case
- ERD

Deadline

The project has a set deadline, which is the 27th of June, 2024. It is crucial that all aspects of the project are completed by this date.

Frontend

Frontend overview

The frontend part of David's book club is made using the React library. In addition to using react, we are using the BEM naming convention for our class names. This provides a common standard, across the whole frontend, which is easy to understand. It also works great with the Scss language.

As part of storing our data, we use Redux, which offers global states. These states can be read by any React component, making it easy to retrieve locally stored data.

Global states

Cart

Items added to the cart are stored in our Redux global state. This means we can show the products anywhere in our web application. They will only be visible in the cart popup and checkout page.

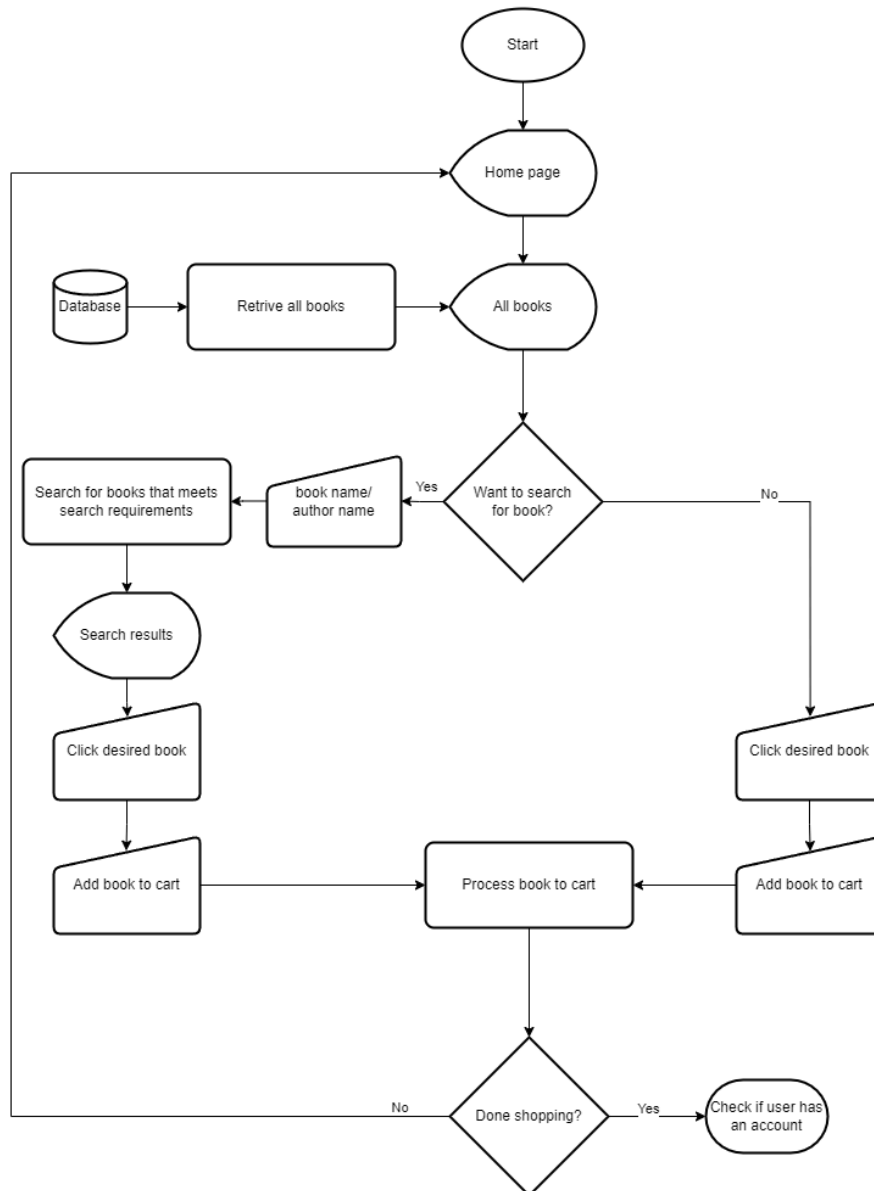
This is an example of how the data will look, within the global state.

```
1  [
2    {
3      "bookId": 1,
4      "title": "Book of David",
5      "Price": 120,
6      "image": "base64 image"
7    }
8  ]
```

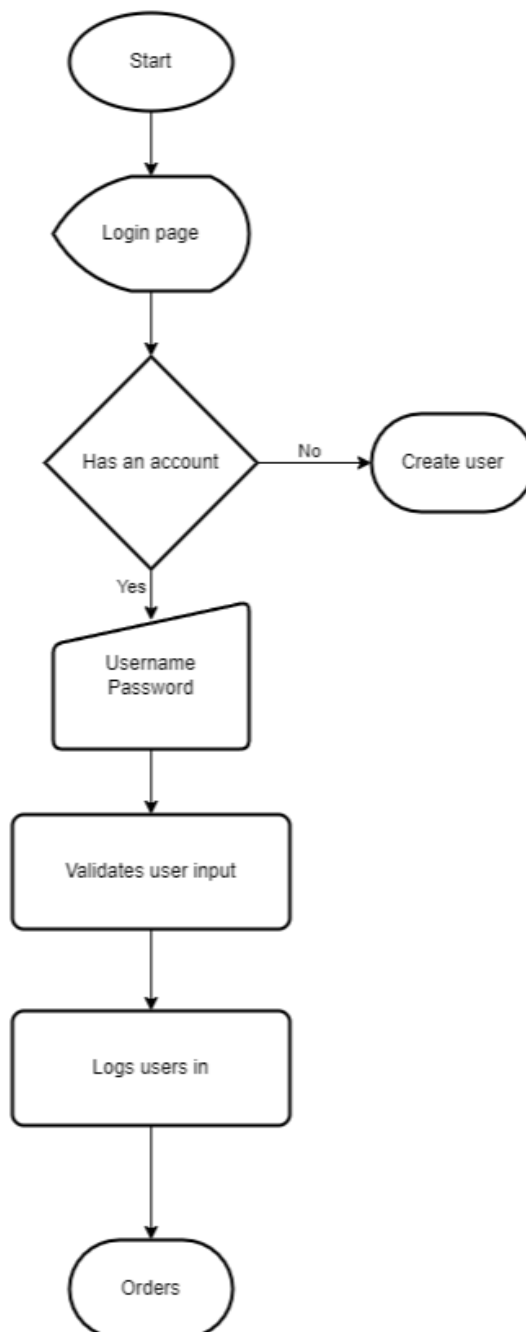
Frontend diagrams

Flowcharts

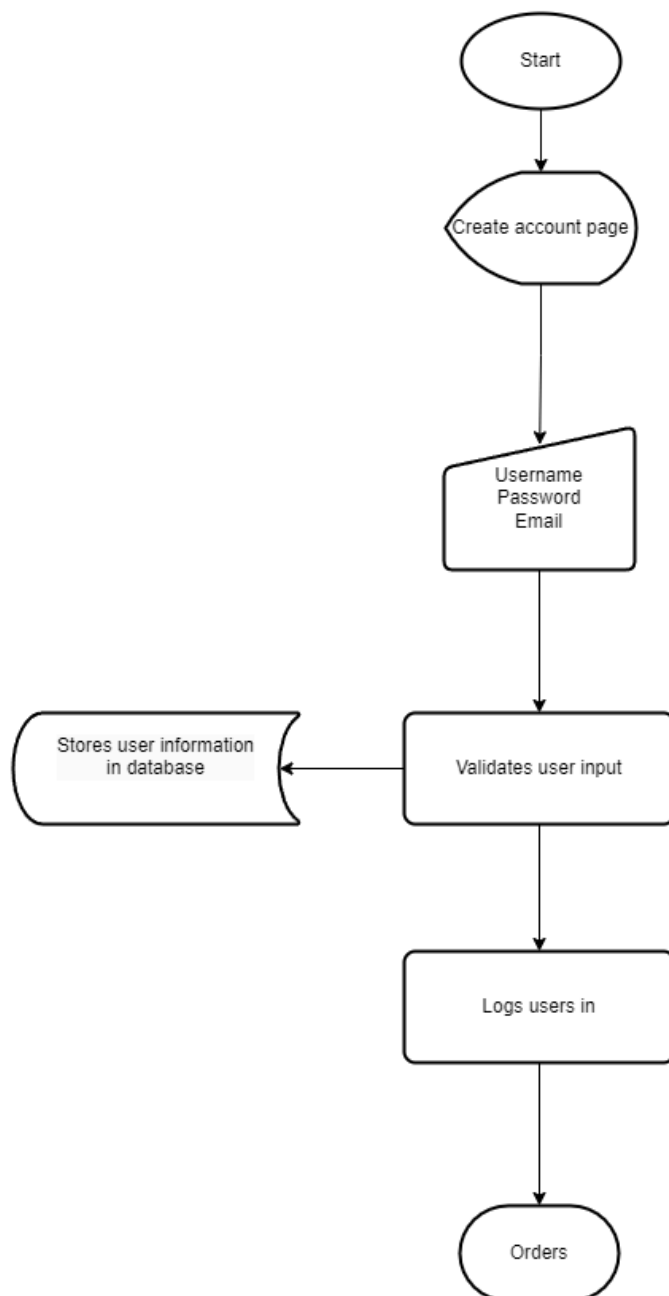
Choose products flow



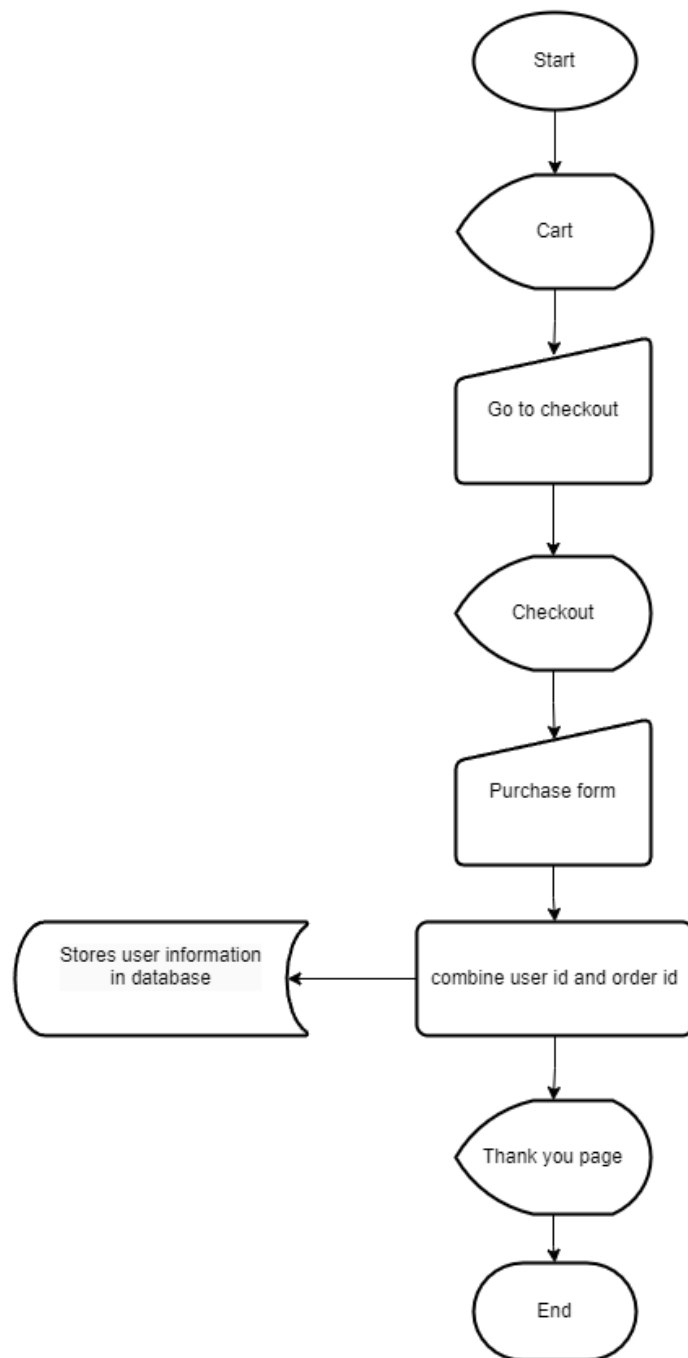
Login flow



Create account flow



Checkout flow



User stories

USER STORY ID	Som <bruger>	Vil jeg gerne <gøre noget>	... så jeg kan <opnå noget>	Accept kriterie
1	Admin	Tilføje bog	Så jeg kan tilføje bog til display	Den nye bog bliver tilføjet til webshoppens
2	Admin	Fjerne bog	Så jeg kan fjerne bog fra display	Den gamle bog bliver fjernet fra webshoppens
3	Admin	Tilføje bog til databasen	Så jeg kan tilføje bog til databasen	Bogen bliver tilføjet til databasen
4	Admin	Fjerne bog fra databasen	Så jeg kan fjerne bog fra databasen	Bogen bliver fjernet fra databasen
5	Admin	Se alle ordre	Så jeg kan få vist alle ordre	Alle ordre bliver vist
6	Admin	Afslutte en kundes ordre	Så jeg kan sætte ✓ ved en ordre	Der bliver sat et ✓ ved ordren
7	Admin	Fjerne en kundes ordre	Så jeg kan slette ordren fra databasen	Kundens ordre bliver slettet fra databasen
8	Kunde	Oprette bruger	Så jeg kan få en bruger	Kunden kan oprette bruger
9	Kunde	Logge ind	Så jeg kan tilgå hjemmesiden som bruger	Kunden kan logge ind
10	Kunde	Tilføje produkt til sin kurv	Så jeg kan købe flere produkter af gangen	Produkt bliver tilføjet til kurven
11	Kunde	Fjerne produkt fra sin kurv	Så jeg kan fjerne produkt fra dem jeg vil købe	Produkt bliver fjernet fra kurven
12	Kunde	Se alle produkter i sin kurv	Så jeg kan se alle de produkter jeg vil købe	Alle produkter fra kurven bliver displayed
13	Kunde	Se ordre	Så jeg kan huske hvad jeg har købt	Ordre bliver vist til kunden
14	Kunde	Købe produkter som er tilføjet til kurv	Så jeg kan købe de produkter jeg har valgt	Kundens valgte produkter bliver lavet om til en ordre
15	Kunde	Se alle produkter	Så jeg kan finde ud af hvilke produkter jeg vil købe	Alle produkter til salg bliver displayed til kunden

Wireframes

Wireframe - Home Page



Log in

Welcome

motto

Book name
Price

BUY

Book name
Price

BUY

Book name
Price

BUY

Book name
Price

BUY

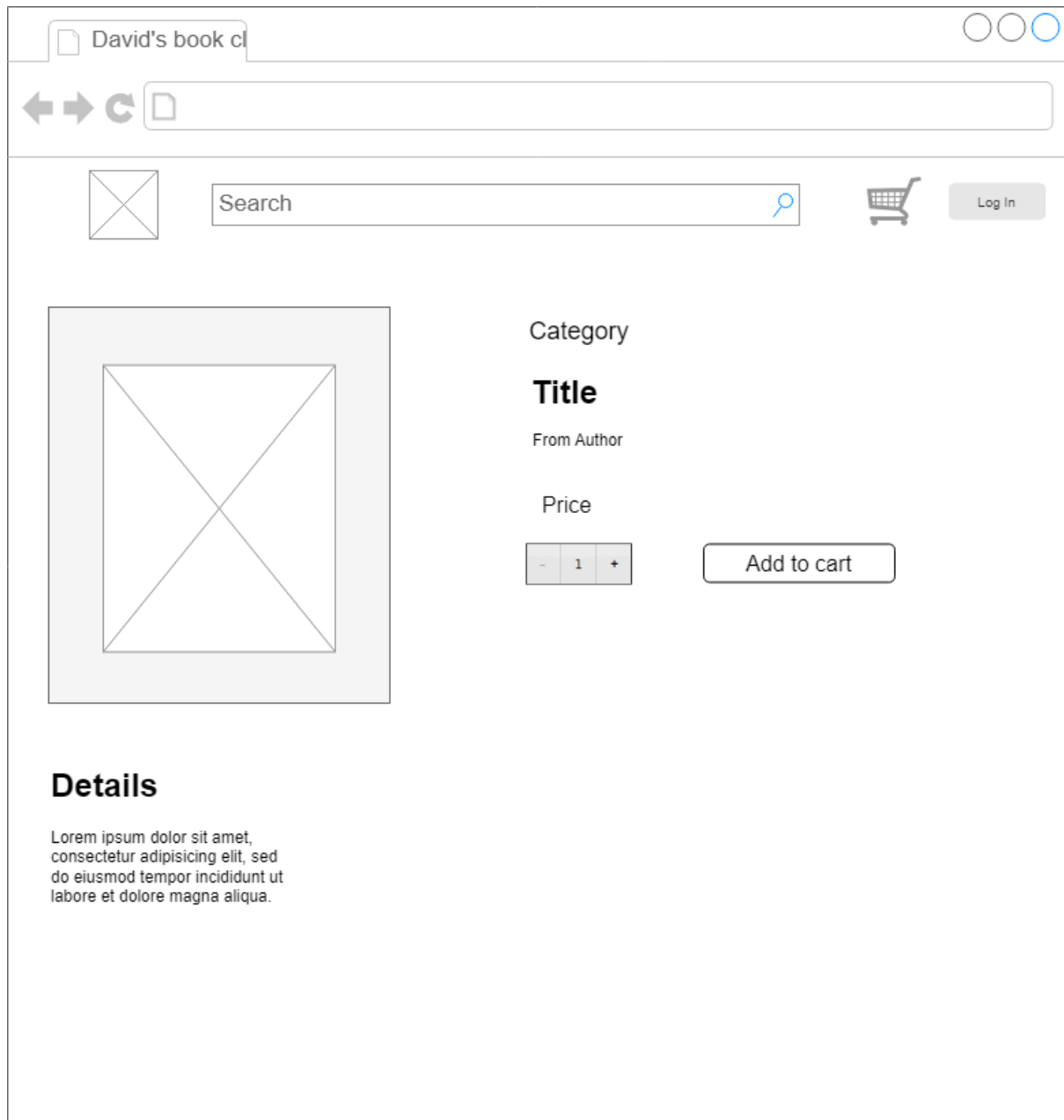
Book name
Price

BUY

Book name
Price

BUY

Wireframe - Product page



Wireframe - Log In

The image shows a wireframe of a web browser window. The browser's address bar contains the text "David's book cl" and has three window control buttons (minimize, maximize, close) on the right. Below the address bar is a navigation bar with back, forward, and refresh icons, and a search icon. The main content area is a large rectangle. In the center of this area is a "Sign In" form. The form has a title "Sign In" and a link "Go Home" in the top right corner. It contains two input fields: "Email:" with the value "example@example.com" and "Password:" with the value "*****". Below these fields is a "SIGN IN" button. At the bottom of the form is a link "or Sign Up".

David's book cl

← → ↻ 🔍

Sign In [Go Home](#)

Email:
example@example.com

Password:

SIGN IN

[or Sign Up](#)

Wireframe - Sign Up

The wireframe shows a browser window with a single tab titled "David's book cl". The address bar is empty. The main content area contains a "Sign Up" form. The form has a title "Sign Up" and a link "Go to Sign In". It includes several input fields for user information, with some fields pre-filled with example data. The fields are: "First & Lastname *" (pre-filled with "John Doe"), "Email *" (pre-filled with "example@example.com"), "Password *" (masked with "*****"), "Repeat password *" (masked with "*****"), "House number (optional)" (pre-filled with "Example: 27"), "Post number (optional)" (pre-filled with "Example: 5100"), and "Street name (optional)" (pre-filled with "Street name"). A "SIGN UP" button is located at the bottom of the form.

David's book cl

← → ↺ 📄

Sign Up [Go to Sign In](#)

First & Lastname *

John Doe

Email *

example@example.com

Password *

Repeat password *

House number (optional)

Example: 27

Post number (optional)

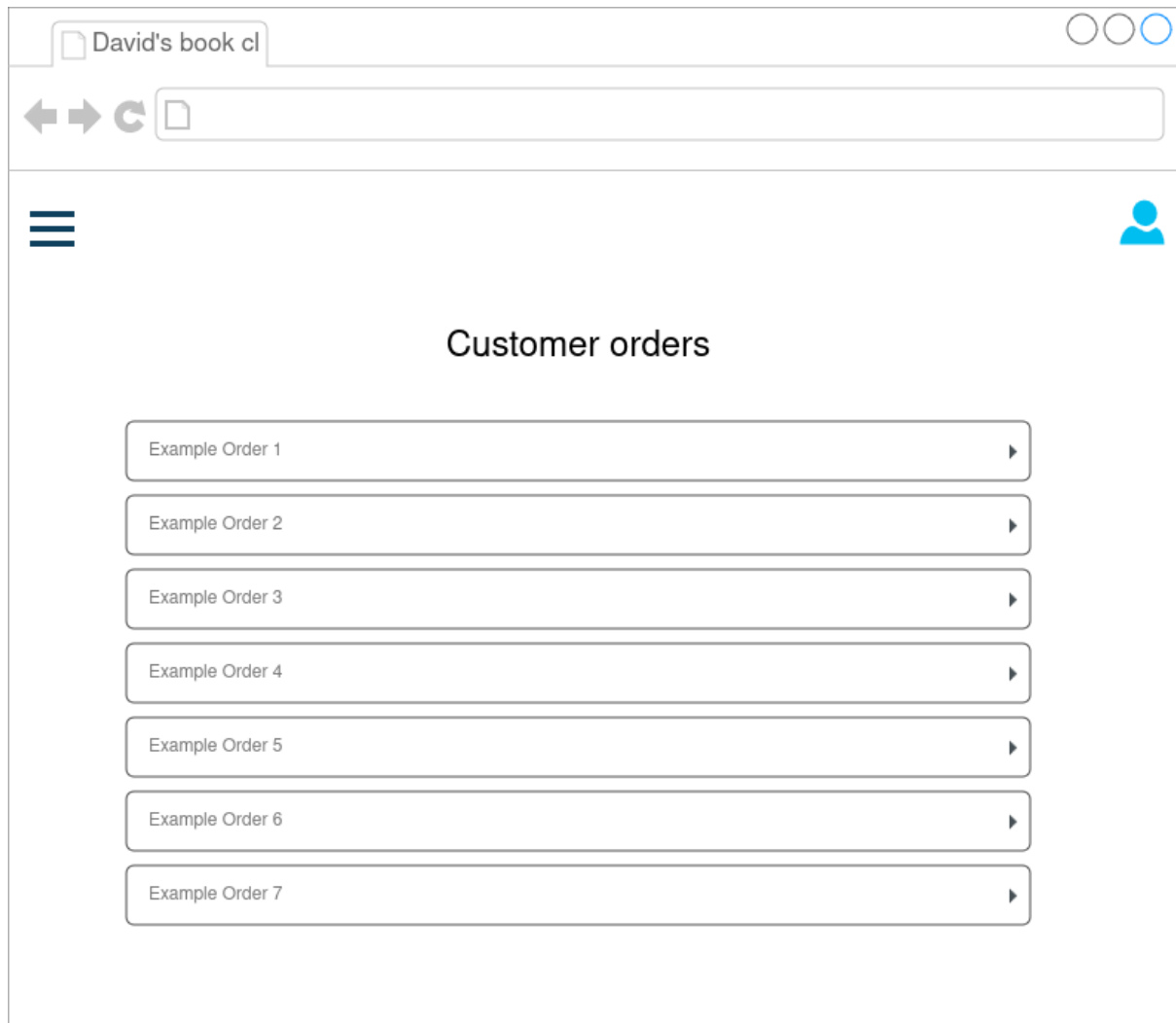
Example: 5100

Street name (optional)

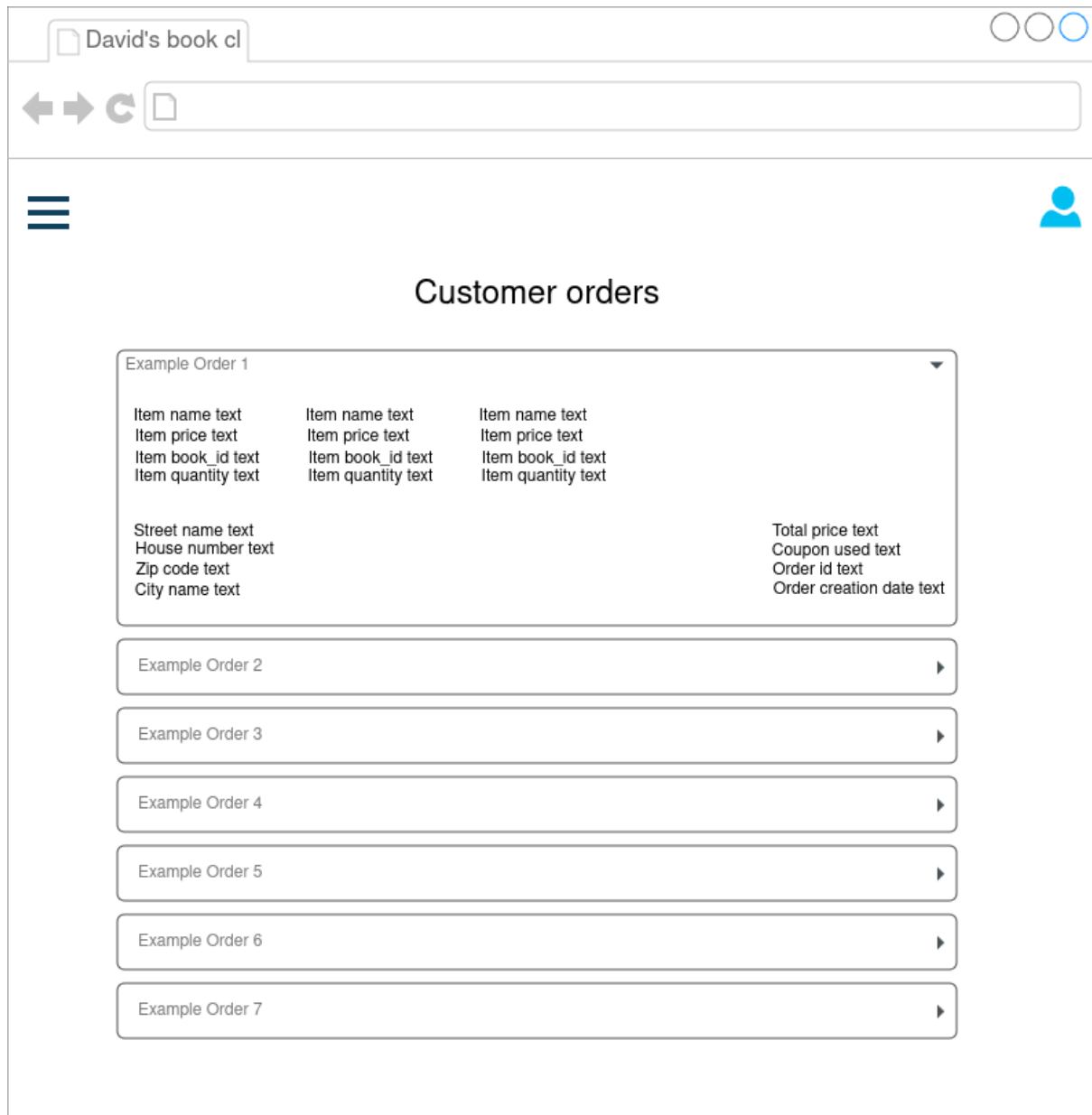
Street name

SIGN UP

Wireframe - Admin



Wireframe - Admin Open Order



Wireframe - Checkout coupon accordion open

David's book cl

← → ↺ 📄

1. Client information

Email*

First and lastname*

Address & house nr.*

Post nr.*By*

Phone nr.*

Continue

2. Delivery

3. Payment

Total items

Edit

Book title

Item amount

Price

Subtotal

Delivery

Total

Amount

Cost

Amount

Discount coupon

Example coupon:

Add Coupon

15

Wireframe - Checkout State 1

David's book cl

← → ↺ 📄

1. Client information

Email*

First and lastname*

Address & house nr.*

Post nr.*

By*

Phone nr.*


Continue

2. Delivery

3. Payment

Total items

Edit



Book title

Item amount

Price

Subtotal

Delivery

Total

Amount

Cost

Amount

Discount coupon

▼

Wireframe - Checkout State 2

David's book club

← → ↺

1. Client Information

Edit

2. Delivery

David's book club - Location

Address

Post nr. City

See info and opening hours ▾

Collect here

David's book club - Location

Address

Post nr. City

See info and opening hours ▾

Collect here

Continue

3. Payment

Total items

Edit

Book title

Item amount

Price

Subtotal

Amount

Delivery

Cost

Total

Amount

Discount coupon

▾

Wireframe - Checkout State 3

David's book cl

← → ↺

1. Client Information

Edit

2. Delivery

Edit

3. Payment

☐ Payment Type 1

Supported payment method vendors

☐ Payment Type 2

Supported payment method vendors

☐ Payment Type 3

Supported payment method vendors

☐ Agree to Terms and Conditions

Confirm & Pay

Total items

Edit

Book title

Item amount

Price

Subtotal

Amount

Delivery

Cost

Total

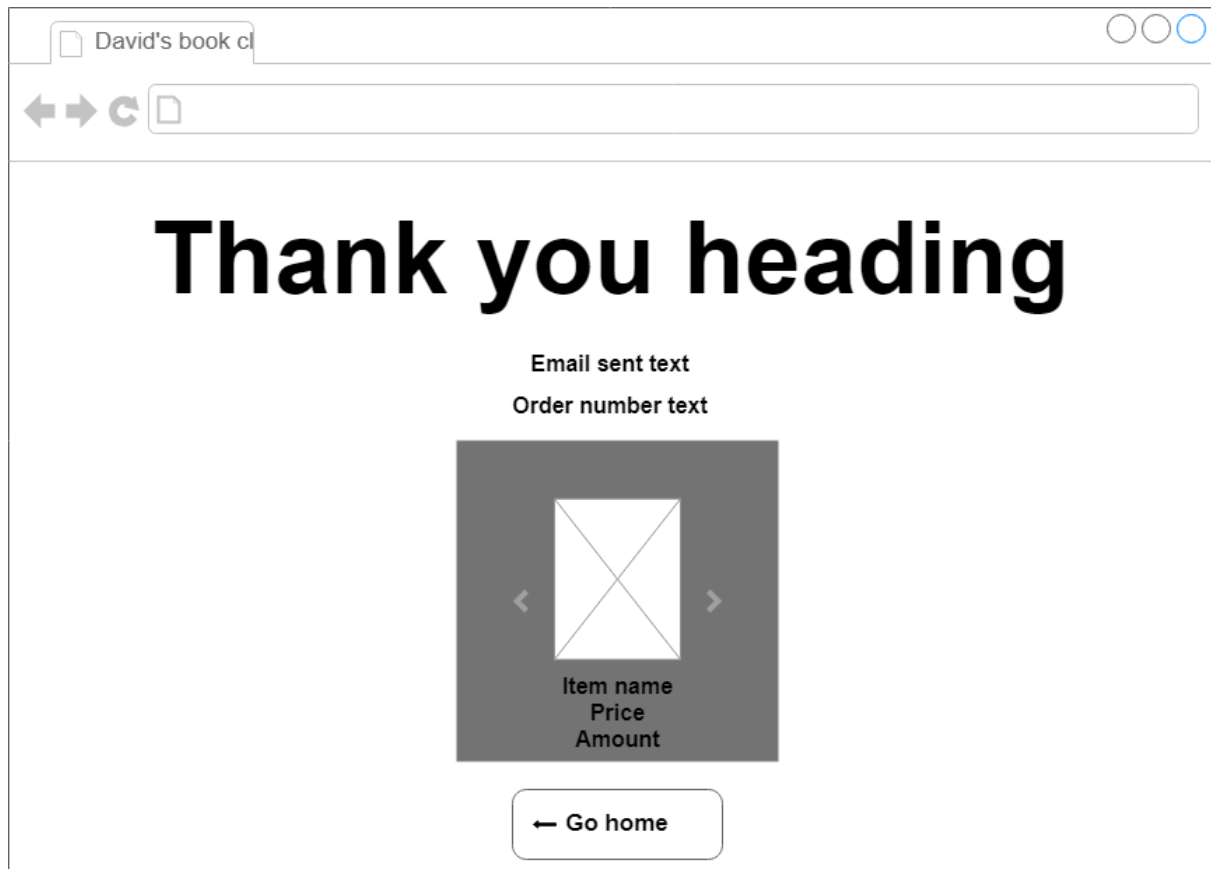
Amount

Discount coupon

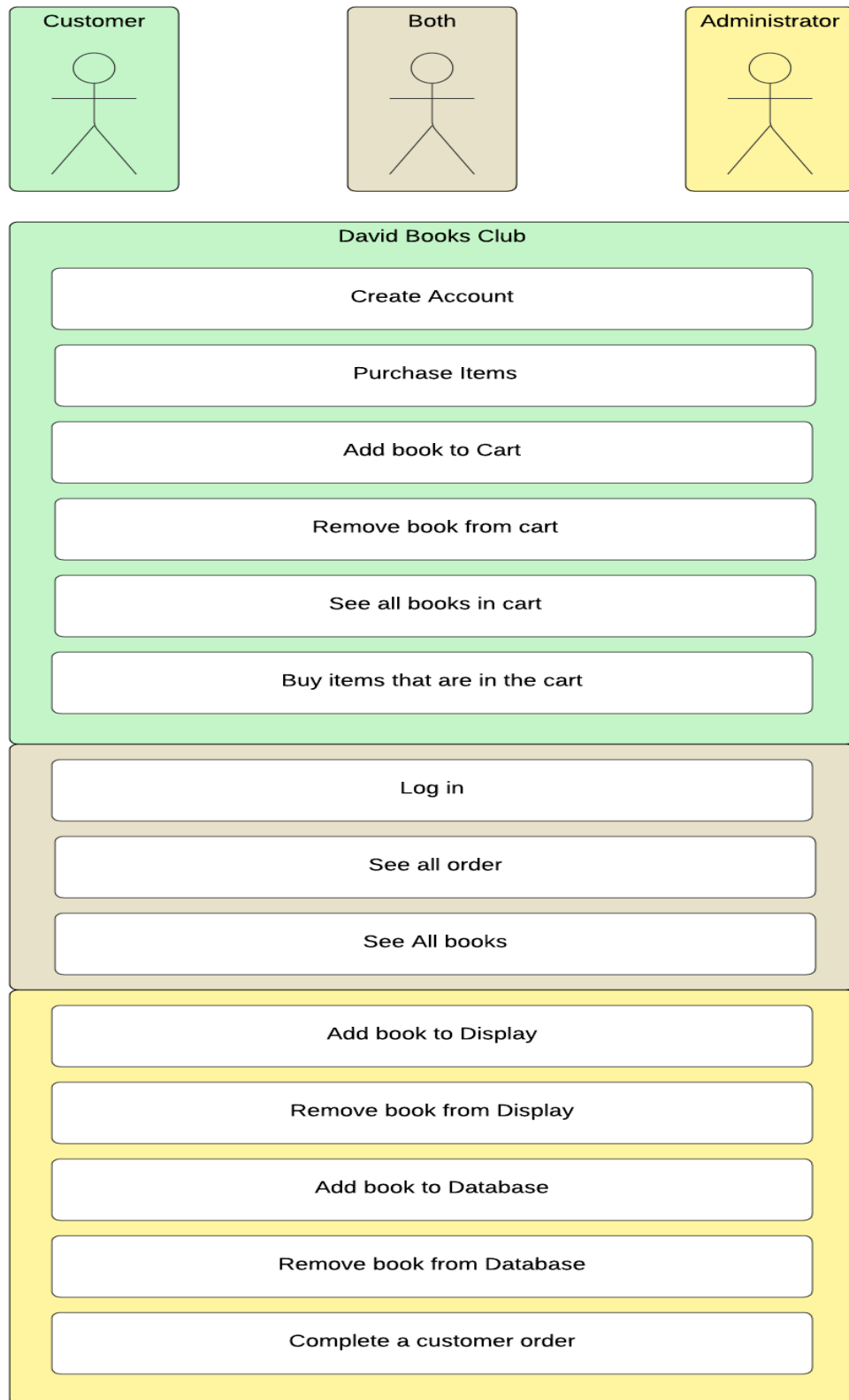
▼

18

Wireframe - Thank you



Use Case Diagram



Use case Specifications:

ID	UC-1.1-CreateAccount
Title	Create Account
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Customer
Purpose and Scope	This use case describes the process of creating a new account for a customer. Customers can create accounts, but administrators have additional privileges for managing the system.
Preconditions	The user has navigated to the "Create Account" page and clicked on the "Create Account" button.
Actions	Customer: Fill in the login information: email, password.2. Fill in personal data: first name, last name.3. (Optional) Fill in additional profile information such as address, phone number, etc.4. Click on the "Save Account" button.
Postconditions	The account is successfully created, and the user can log in and begin using the system.
Error Handling	- If mandatory fields (email and password) are empty, an error message is displayed to the user.- If the email already exists in the system, an error message is displayed to inform the user.
End State	The new user account is displayed in the user overview.

ID	UC-1.3-AddItemsToCart
Title	Add Items to shopping cart
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Customer
Purpose and Scope	This use case describes how users can add items to their account or shopping cart.
Preconditions	The user does not necessarily need to log into the system.
Actions	1. Navigate to the "Items" section.2. Select the items to add.3. Click on the "Add to Cart" button.
Postconditions	The selected items are added to the user's cart.
End State	The items are successfully added to the cart and ready for purchase.

ID	UC-1.4-RemoveItemsToCart
Title	Remove Items to shopping cart
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Customer
Purpose and Scope	This use case describes how users can Remove items from their shopping cart.
Preconditions	The user has added items to cart
Actions	1. Navigate to the "Cart" section. 2. Select the items to Remove. 3. Click on the "Remove from Cart" button.
Postconditions	The selected items are Removed from the user's cart.
End State	The items are successfully Removed from the cart.

ID	UC-1.5-SeeBooksInCart
Title	See all books in cart
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Customer
Purpose and Scope	This use case describes how users can see items in their shopping cart.
Preconditions	The user does not necessarily need to log into the system.
Actions	1. Navigate to the "Cart" section.
End State	After going into the cart user is able to see there selected books

ID	UC-1.6-BuyBooksInCart
Title	Buy all books in cart
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Customer
Purpose and Scope	This use case describes how users can buy items in their shopping cart.
Preconditions	The user does not necessarily need to log into the system.
Actions	1. Navigate to the "Cart" section. 2. Click on "continue with payment"
End State	After the payment user will receive a message regarding it's successful purchase

ID	UC-2.1-Login
Title	Log in
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Administrator & Customer
Purpose Description and Scope	This use case describes the process for an administrator and customer to log in to their account.
Preconditions	The user has navigated to the "Login" page.
Actions	1. Enter email 2. Enter password 3. Click the "Log in" button.
Postconditions	The user is logged into the system and directed to the homepage.
Error Handling	- If the email or password is incorrect, an error message is displayed
End State	The user is logged into the system.

ID	UC-2.2-CustomerSeeOrder
Title	Customer See Order
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Customer
Purpose Description and Scope	This use case describes the process for a customer to see their order/orders.
Preconditions	The customer is logged in
Actions	1. Click on profile icon, 2. Click on See orders
Postconditions	The user is logged into the system and directed to the homepage.
Error Handling	- If the email or OrderID is incorrect, an error message is displayed
End State	The user is able to see its order..

ID	UC-2.3-AdminSeeAllOrder
Title	Admin sees all customer orders
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Administrator
Purpose Description and Scope	This use case describes the process for an administrator to see all customer orders
Preconditions	The admin is logged in
Actions	1. The admin clicks on profile in navbar 2. The admin clicks on "See customer orders"
Postconditions	All customer orders are shown to the admin
End State	The Admin can see all customer orders

ID	UC-2.4-SeeAllBooks
Title	Show Books
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Administrator & Customer
Purpose Description and Scope	This use case describes the process for an administrator and customer to see all the books.
Preconditions	The user has navigated to the "Home" page.
Actions	Scroll to see
Postconditions	The user is directed to the homepage.

ID	UC-3.1-AddBooksToDisplay
Title	Add Books to Display
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Administrator
Purpose and Scope	This use case describes how an administrator can add books to the display section.
Preconditions	The administrator is logged into the system.
Actions	1. Navigate to the "AdminPanel" section. 2. Select the books to add to display. 3. Click the "Add to Display" button.
Postconditions	The selected books are added to the display section.
Error Handling	- If no books are selected, an error message is displayed.
End State	The books are successfully added to the display section.

ID	UC-3.2-AddBooksToDatabase
Title	Add Books to Database
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Administrator
Purpose and Scope	This use case describes how an administrator can add new books to the database.
Preconditions	The administrator is logged into the system.
Actions	1. Navigate to the "Add Book" section. 2. Enter book details (title, author, ISBN, etc.). 3. Click the "Save Book" button.
Postconditions	The new book is added to the database.
Error Handling	- If mandatory fields are empty, an error message is displayed. - If the book already exists, an error message is displayed.
End State	The book is successfully added to the database.

ID	UC-3.3-RemoveBooksFromDisplay
Title	Remove Books from Display
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Administrator
Purpose and Scope	This use case describes how an administrator can remove books from the display section.
Preconditions	The administrator is logged into the system.
Actions	1. Navigate to the "Books" section. 2. Select the books to remove from display. 3. Click the "Remove from Display" button.
Postconditions	The selected books are removed from the display section.
Error Handling	The selected books are removed from the display section.
End State	The books are successfully removed from the display section.

ID	UC-3.4-RemoveBooksFromDatabase
Title	Remove Books from Database
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Administrator
Purpose and Scope	This use case describes how an administrator can remove books from the database.
Preconditions	The administrator is logged into the system.
Actions	1. Navigate to the "Books" section. 2. Select the books to remove from the database. 3. Click the "Delete Book" button.
Postconditions	The selected books are removed from the database.
Error Handling	- If no books are selected, an error message is displayed. - If the book is associated with existing orders, an error message is displayed.
End State	The books are successfully removed from the database.

ID	UC-3.5-CompleteCustomerOrder
Title	Complete a Customer Order
Developer	Magnus, Marcus, Robert, Shazil, Yordan
Actors	Administrator
Purpose and Scope	This use case describes how an administrator can complete a customer order.
Preconditions	The administrator is logged into the system and there are pending orders.
Actions	1. Navigate to the "Orders" section. 2. Select the pending order to complete. 3. Verify the order details. 4. Click the "Complete Order" button.
Postconditions	The order is marked as completed and a confirmation message is displayed.
Error Handling	- If the order cannot be completed due to missing information, an error message is displayed.
End State	The order is successfully completed and updated in the system.

Frontend dependencies

Axios

We utilize Axios for our HTTP requests to our API.

It provides better error handling, and overall ease of use, compared to fetch.

<https://www.npmjs.com/package/axios>

React, React-dom, React-router-dom

To run our React web application, some React dependencies are required.

Of which is React itself, which is obviously required.

React-dom follows the installation of React.

React-router-dom lets us go to other routes, within our React SPA website. Changing the page route in other ways, such as using `` would remove our global states and is not the recommended approach for changing the website view.

- React: <https://www.npmjs.com/package/react>
- React-dom: <https://www.npmjs.com/package/react-dom>
- React-router-dom: <https://www.npmjs.com/package/react-router-dom>

React helmet

This dependency is used through a custom React hook.

The purpose is to dynamically update the website title, based on the route being displayed.

<https://www.npmjs.com/package/react-helmet>

Redux, React-redux, ReduxJS/toolkit

We use Redux to keep track of global states.

These 3 dependencies are all required to achieve this.

Redux and React-redux are required for making redux work within React.

The ReduxJS/toolkit is required to create a redux store, using best practices, without deprecated methods.

- Redux: <https://www.npmjs.com/package/redux>
- Redux-react: <https://www.npmjs.com/package/react-redux>
- ReduxJS/toolkit: <https://www.npmjs.com/package/@reduxjs/toolkit>

Classnames

This dependency is used to toggle classes on components, based on a boolean value.

That is useful to display a specific styling, depending on a value or state.

<https://www.npmjs.com/package/classnames>

Sass

This dependency is required to run Scss, which we utilize within our web application.

This allows us to use variables between different scss files, which is useful for changing a specific style for the overall project.

In addition to this, it also serves well with the BEM naming convention, which we are using.

<https://www.npmjs.com/package/sass>

Redux-persist

This dependency allows our redux global state to not reset, on page reload.

It works by using local storage.

<https://www.npmjs.com/package/redux-persist>

redux-thunk

We are gonna be utilizing this dependency for making our API calls, which relate to changing global state values.

<https://www.npmjs.com/package/redux-thunk>

react-infinite-scroll-component

We use this to have an infinite scroll on the home page, admin product management page and admin orders page. In addition to using this dependency, we make API calls using “collections”. By this we mean that the frontend keeps track of how many collections it has received and will ask the API to fetch the next collection. The size of the collection changes, based on the page.

<https://www.npmjs.com/package/react-infinite-scroll-component>

Backend

Backend overview

The backend API is made with PHP 8.3.3, which is incredibly up-to-date.
We are using composer to handle namespaces and dependencies.

The API is made with the thought of high reusability.
This means that the functions are highly dynamic, without becoming too complex and hard to understand.

API endpoints

Get requests

Products

`/api/v1/products/getProducts`

This endpoint is used to get the products for the home page.
It uses infinite scrolling. Refer to the frontend dependency documentation, `react-infinite-scrolling-component`.
As an input parameter, it takes a collection.
It returns a json object, which includes:

- Primary images
- Title
- Price
- Book id

`/api/v1/products/getProduct`

This endpoint gets used on the product page.
It takes a parameter, which is the book id.
This is used to identify which book should be retrieved.
It returns a json object, including:

- All product images
- Title
- Release date
- Stock
- Genre
- Category
- Price
- ISBN
- Language

`/api/v1/products/searchProducts`

This endpoint is used for searching products, through the search bar.

It takes an input parameter, which is what the user searched for.

This could be either the book author or the book name.

It returns a json object, with:

- Book id
- Title
- Price

It returns 5 products.

Orders

`/api/v1/orders/verifyCoupon`

This endpoint is used on the checkout page.

It has an input parameter, which is the user input in the Coupon field

It returns the coupon discount percentage, matching the user input, if the coupon is not expired.

`/api/v1/orders/getOrders`

This api endpoint is used on the admin panel.

It has a “collection” input parameter.

For a more detailed description on “collections”, refer to the frontend dependency, react-infinite-scroll-component in the documentation.

As json, it returns:

- Order id
- Full Address
- Created at
- Products added (Primary image, title, author, price)

`/api/v1/orders/searchOrders`

This endpoint allows the admin to search for a specific order, within the admin panel.

It uses infinite scroll, refer to react-infinite-scroll-component in frontend dependencies, in the documentation.

Input parameters:

- Name or order id
- Collection

In the json format, it returns:

- Order id
- Full Address
- Created at
- Products added (Primary image, title, author, price)

`/api/v1/orders/getUserOrders`

This endpoint gets all orders made by a logged in user.

For this API call, we use infinite scrolling. Refer to react-infinite-scroll-component in the frontend dependencies part of this documentation.

Input parameter is the collection, for infinite scrolling.

It then returns

- Order id
- Full Address
- Created at
- Products added (Primary image, title, author, price)

`/api/v1/orders/getCityfromZipcode`

This endpoint is used on the checkout page.

It takes a parameter which is the zip code.

It then returns the city associated with that zipcode, as json.

Users

`/api/v1/users/getUserBillingInfo`

This endpoint gets logged-in users' billing information, for the checkout page.

The billing fields will be auto-filled, by the user data that we have.

It takes a JWT parameter to verify the user and returns:

- Name
- Email
- Zip code
- City
- Phone number
- Street name
- House number

Post requests

Orders

`/api/v1/orders/createOrder`

This endpoint takes the form data from the checkout as an input parameter.

This includes:

- Full name
- Zip code
- Street name
- House number
- Phone number
- Email
- Card number
- CVV
- Card expiration date
- Shipping method

Users

`/api/v1/users/createUser`

This endpoint creates a new user, based on some input parameters. These are:

- Name
- Email
- Password
- Street name (optional)
- House number (optional)
- Zip code (optional)
- Phone number

It returns a few states, which are:

- Successful sign-up
- Error, Email exists
- Error, server error

`/api/v1/users/loginUser`

This endpoint logs in users. It requires some input parameters, which are:

- Email
- Password

If the login information is correct, then it returns a JWT to the user.

Else, the user will be informed of invalid login credentials.

`/api/v1/users/logoutUser`

This endpoint logs out the user.

As an input parameter, the JWT is required.

The backend will remove session data and the JWT itself.

Once done, it returns a signal that the task is completed and the front-end logs the user, removing session tokens.

Products

`/api/v1/products/toggleBookDisplay`

This api call toggles the display state of the book.

It is a boolean value which changes between 0 and 1.

Input parameter is the book id.

it only returns a HTTP status code.

Hashing and encryption

JWT hashing

For our JSON Web Tokens, we are using the HS256 hashing algorithm.

This is due to the following reasons:

- Ease of use. HS256 is super easy to get started with.
- It uses a symmetric key, meaning we do not need to manage public keys.
- FAST!

It comes with flaws, again pointing back to the symmetric key.

If the symmetric key is leaked, it could aid an attacker in creating valid JWTs.

However, for our school project, it is an excellent choice as our JWT hashing algorithm.

Password hashing

As for password hashing, we have decided to use the Argon2id algorithm.

It combines the best parts of Argon2d and Argon2i.

This makes it highly resistant to GPU cracking attacks and protects against side-channel attacks and time-memory trade-off attacks.

In addition to just hashing the passwords, we are using both salt and pepper.

Encryption

We are gonna be encrypting sensitive data, such as email addresses.

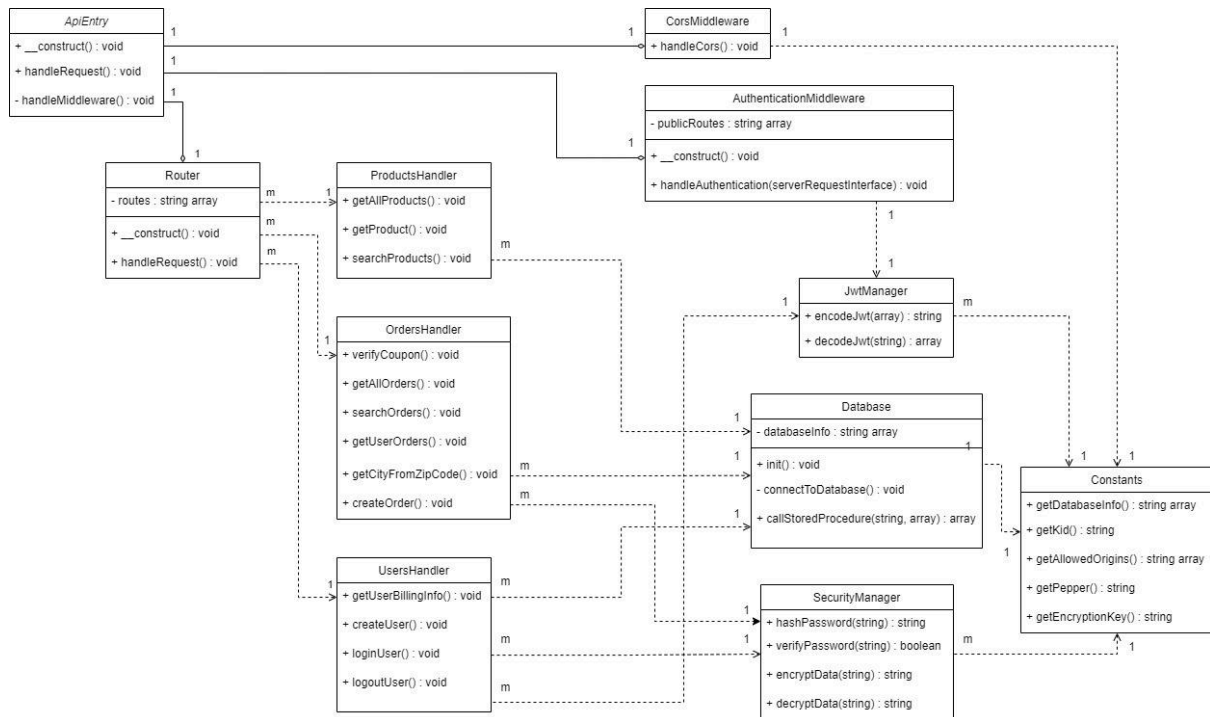
To do this, we will be using AES-128.

The reason for not using AES-192 or AES-256 is due to AES-128 being faster and requiring less computational resources. AES-128 is less secure than the higher bit versions, however the encrypted data is not top secret, so AES-128 will do the job just fine.

Speed is important, as part of the overall user experience, which is a key factor for why we use AES-128.

Backend diagrams

Class diagram



Backend dependencies

Our API runs using composer. This has allowed us to install the following dependencies.

vlucas/phpdotenv

This dependency allows us to use .env files to keep private information.

It is initialized within index.php, and used within our Constants class.

This is because our Constants class is responsible for returning the .env data, in a pretty way.

<https://packagist.org/packages/vlucas/phpdotenv>

ext-pdo_mysql

This dependency is required to create our MySQL database connection and queries.

<https://www.php.net/manual/en/ref.pdo-mysql.php>

firebase/php-jwt

php-jwt is an easy to use tool, which we utilize for encoding and decoding our Json Web Tokens.

<https://packagist.org/packages/firebase/php-jwt>

Database

Database overview

We have created a database, which runs MySQL, with the InnoDB engine.

The database is running on the default MySQL port, which is 3306.

The hostname is mysql.magnuslund.com.

The name of the database is “davidsbookclub”.

We have followed the MySQL naming convention, for our database, tables and columns.

Database users and permissions

Role	Username	Password	Permissions
Admin	MrDavid	SuperStrongPassword!	ALL
API	BookClub	StrongPassword	Execute

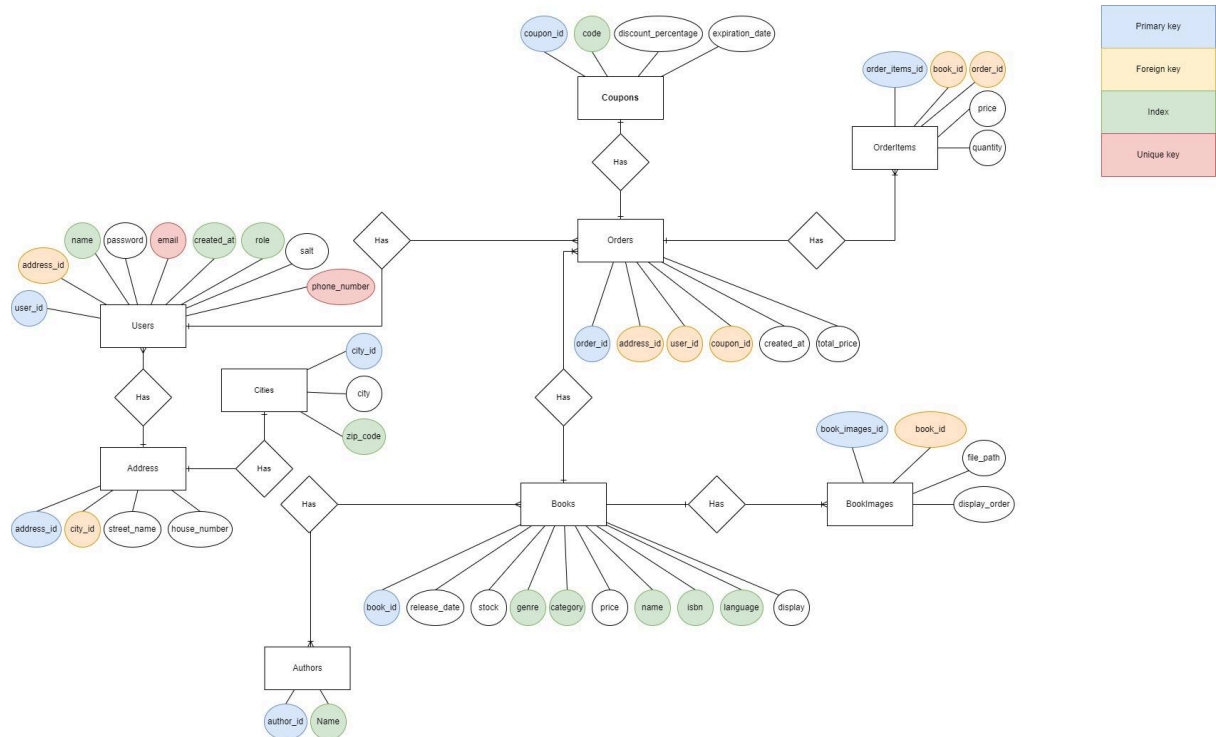
Stored procedures

Stored procedure name	Description
toggleBookDisplay	This procedure will be called to modify book display options. Procedure will be called to make the book visible or hidden. Opposite value of its current value.
createOrder	<p>This procedure will be called when the user buys something. This procedure will just add the order data into the order database and modify other information regarding the order. The input parameters are</p> <ul style="list-style-type: none"> • Name • Zip code • Street name • House number • Phone number • Email • Shipping method <p>It links up the order with a user, based on the email address. If there is no email address in the system for that user, then a new guest role user gets created. It returns a message, if it was successful or not.</p>
getUserOrders	<p>This stored procedure gets orders for the logged in user. The stored procedure is impacted by the infinite load, which means that it returns only some of the users orders at a time. The input parameters are user_id and a start value and end value for the rows to return.</p>
getOrders	<p>This procedure will be called by the admin to see some of the orders of customers. It is impacted by infinite scrolling, so it returns an amount of the total orders. The input parameter is the start and end index of rows to return.</p>
LoginUser	<p>This procedure will be called when the user tries to login. It takes an input parameter of the email and hashed password. It returns a message, if it was successful or not.</p>
signupUser	<p>This stored procedures has the the following input parameters</p> <ul style="list-style-type: none"> • Hashed password • Encrypted email • Encrypted phone number • Name • Zip code • House number • Street name <p>It returns a response message.</p>
GetProducts	<p>This procedure gets a part of the products, due to lazy loading for infinite scroll on the home page. It takes an input parameter for the start and end row to return. It returns the rows in between the range.</p>

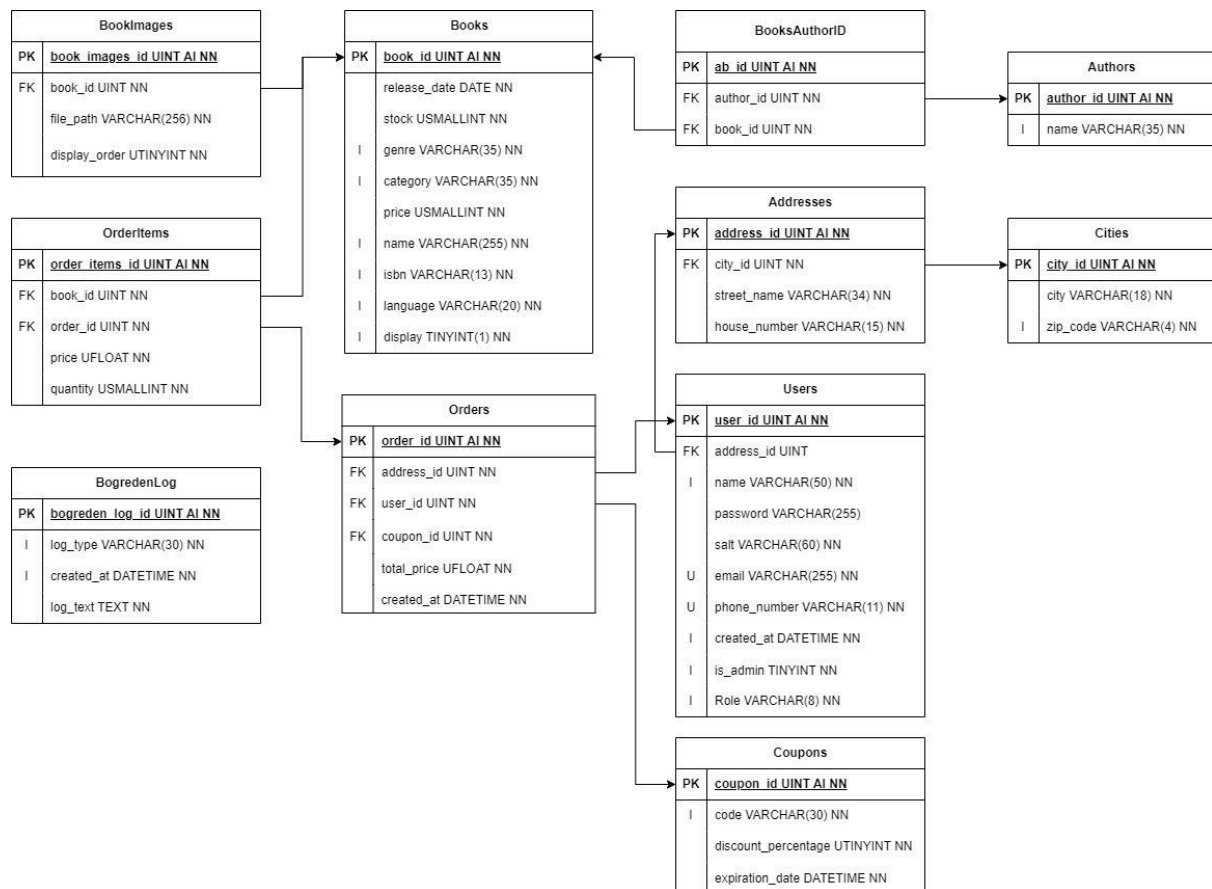
Stored procedure name	Description
getProduct	<p>This stored procedure takes an input parameter, for the product id. It then returns</p> <ul style="list-style-type: none"> • All product images • Booktitle • Release date • Stock • Genre • Category • Price • ISBN • Language
searchProduct	<p>This stored procedure takes 1 parameter. That is the search input. The procedure will match this up with the author name and book titles. It will return 5 results.</p>
verifyCoupon	<p>This stored procedure takes an input parameter, which is the inputted coupon code. If the input matches a coupon, then it returns the discount percentage.</p>
searchOrders	<p>This stored procedure gets 3 input parameters, because it uses infinite scrolling.</p> <ul style="list-style-type: none"> • Order id to search for • Start index for which rows to return • Stop index for rows to return
getCityFromZipcode	<p>This stored procedure gets an input parameter, which is the zip code. It then returns the city name which matches the zip code.</p>
createUser	<p>This stored procedure takes 7 input parameters. 3 of which are optional.</p> <ul style="list-style-type: none"> • Name • Email • Password • Phone number • Zip code (optional) • House number (optional) • Street name (optional) <p>Using this information, it can create the required rows in the database, for a new user.</p>
getUserBillingInfo	<p>This stored procedure takes an input parameter, which is the user id. It then returns the following values (some null):</p> <ul style="list-style-type: none"> • Name • Email • Phone number • Street number • House number • Zip code • City name

Database diagrams

ERD



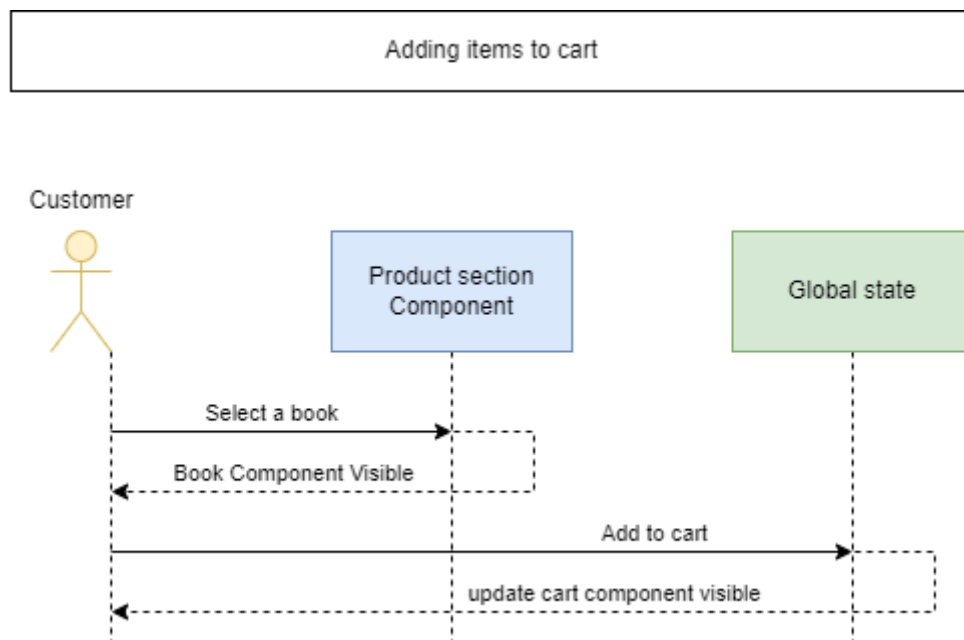
RDS



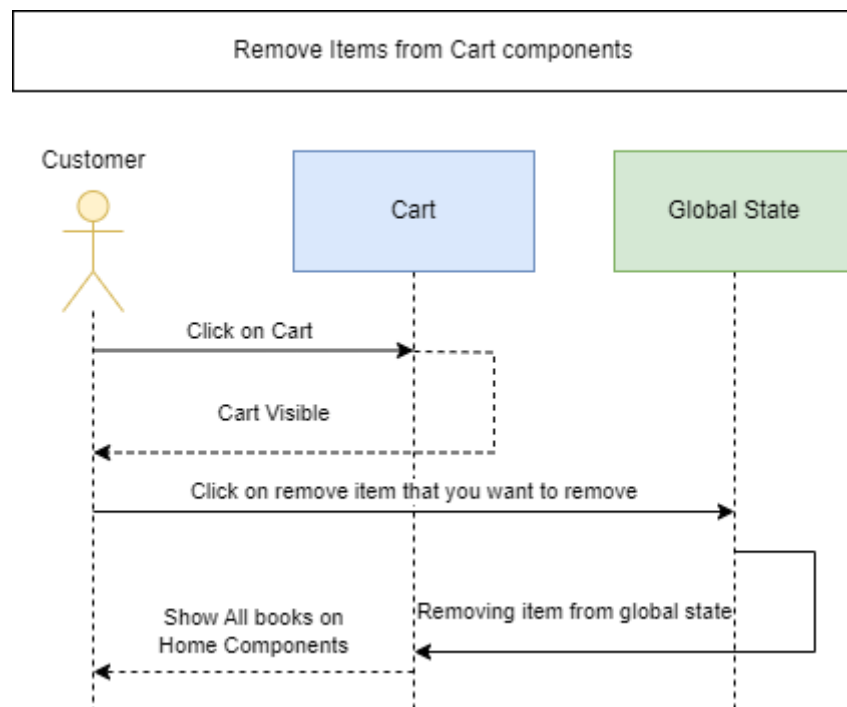
General diagrams

Sequence diagram:

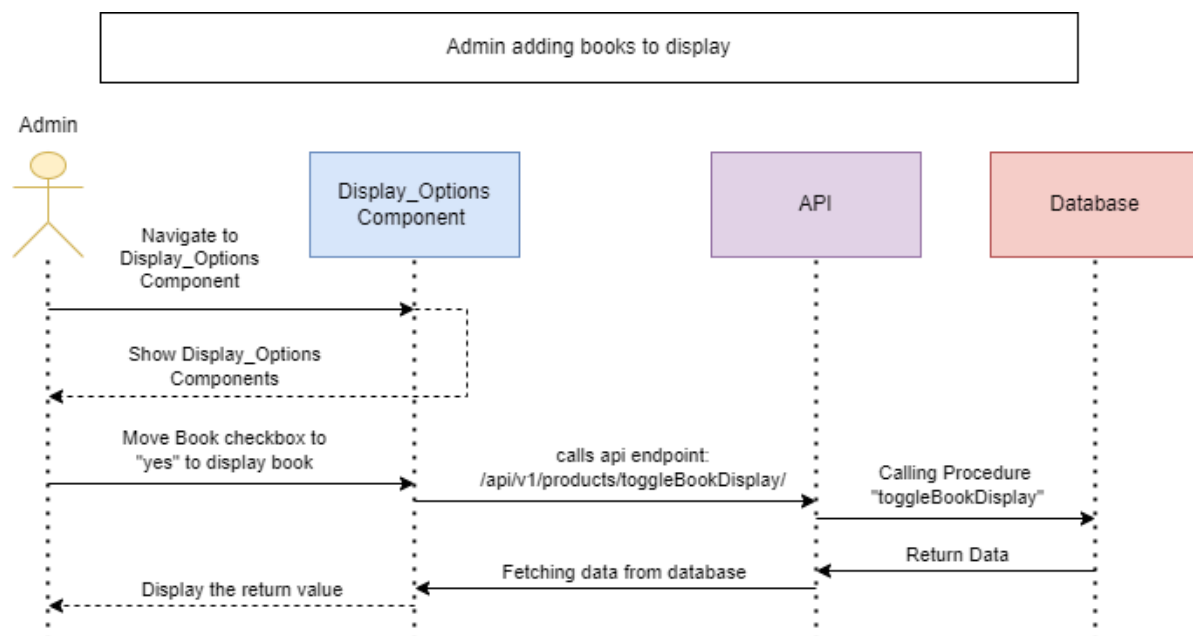
Add Items To Cart:



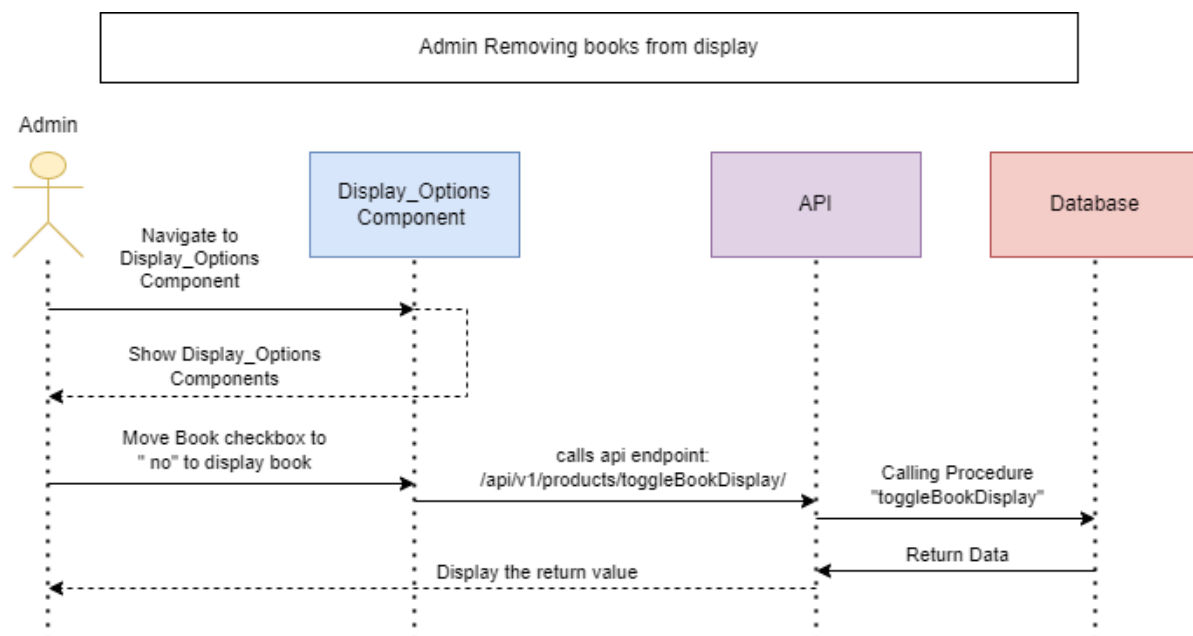
Remove Items from Cart:



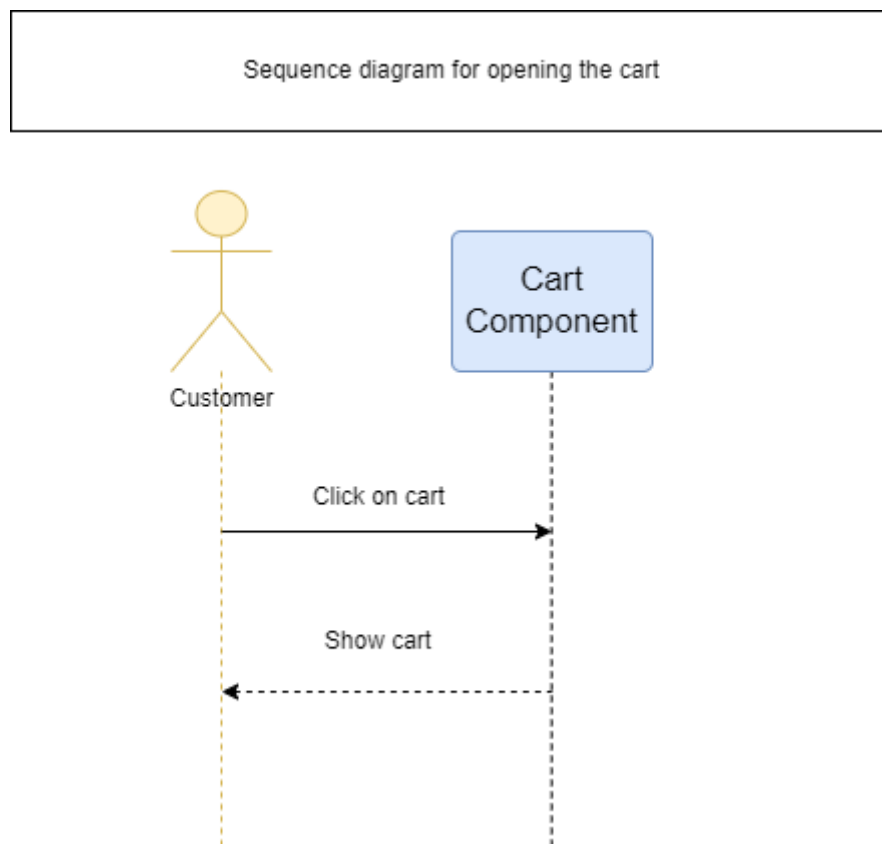
Add Items to Display:



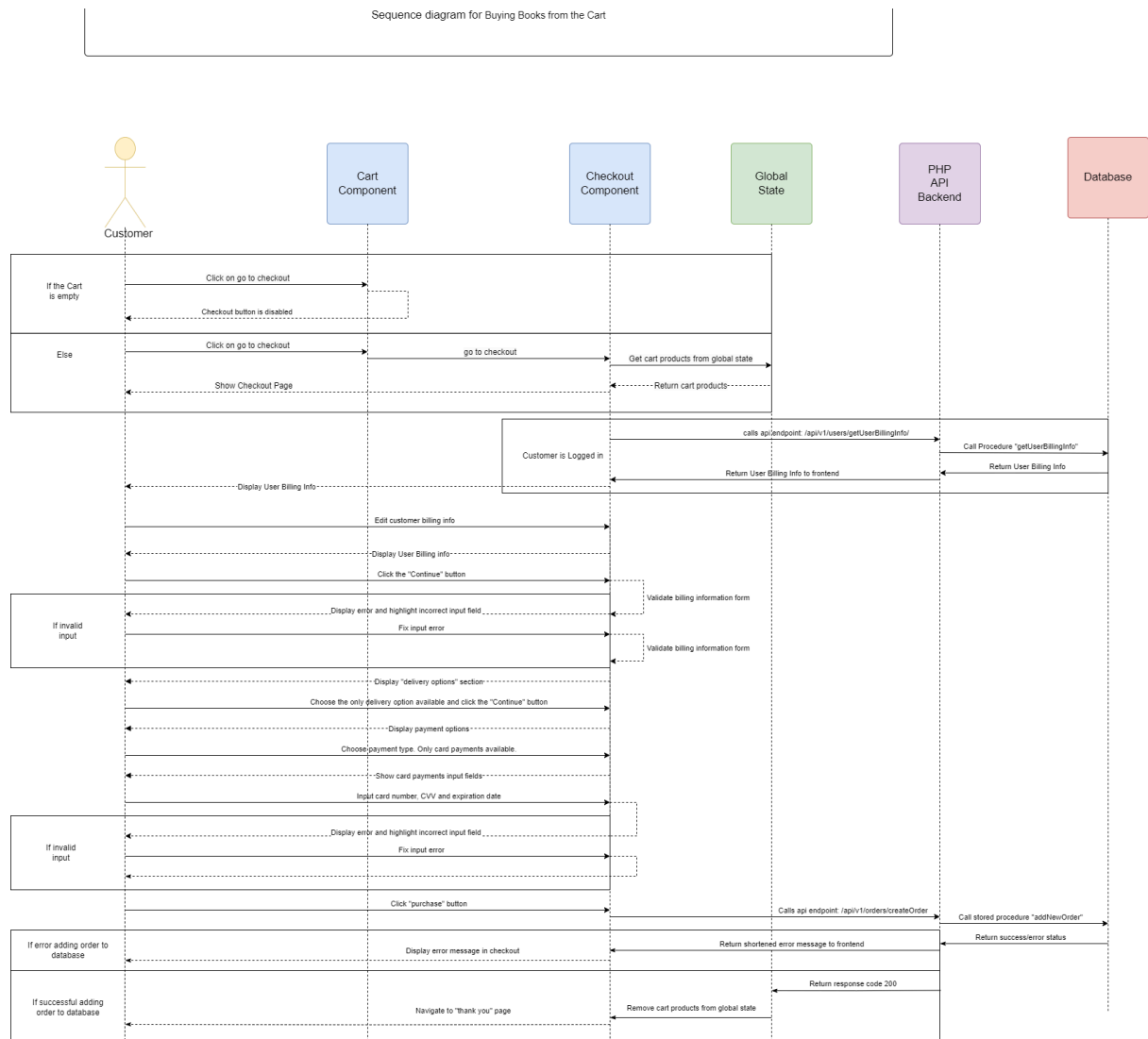
Remove items from Display



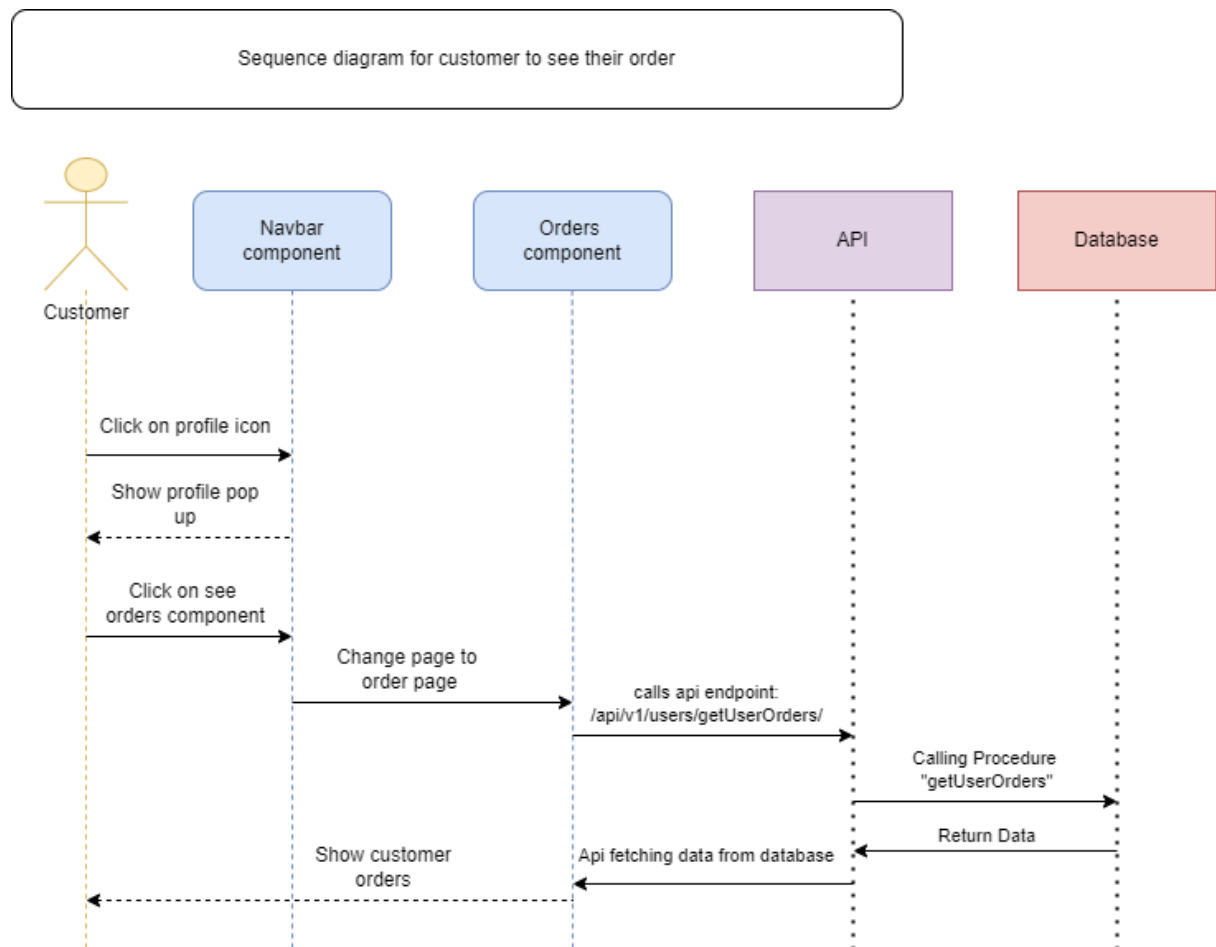
Open Cart:



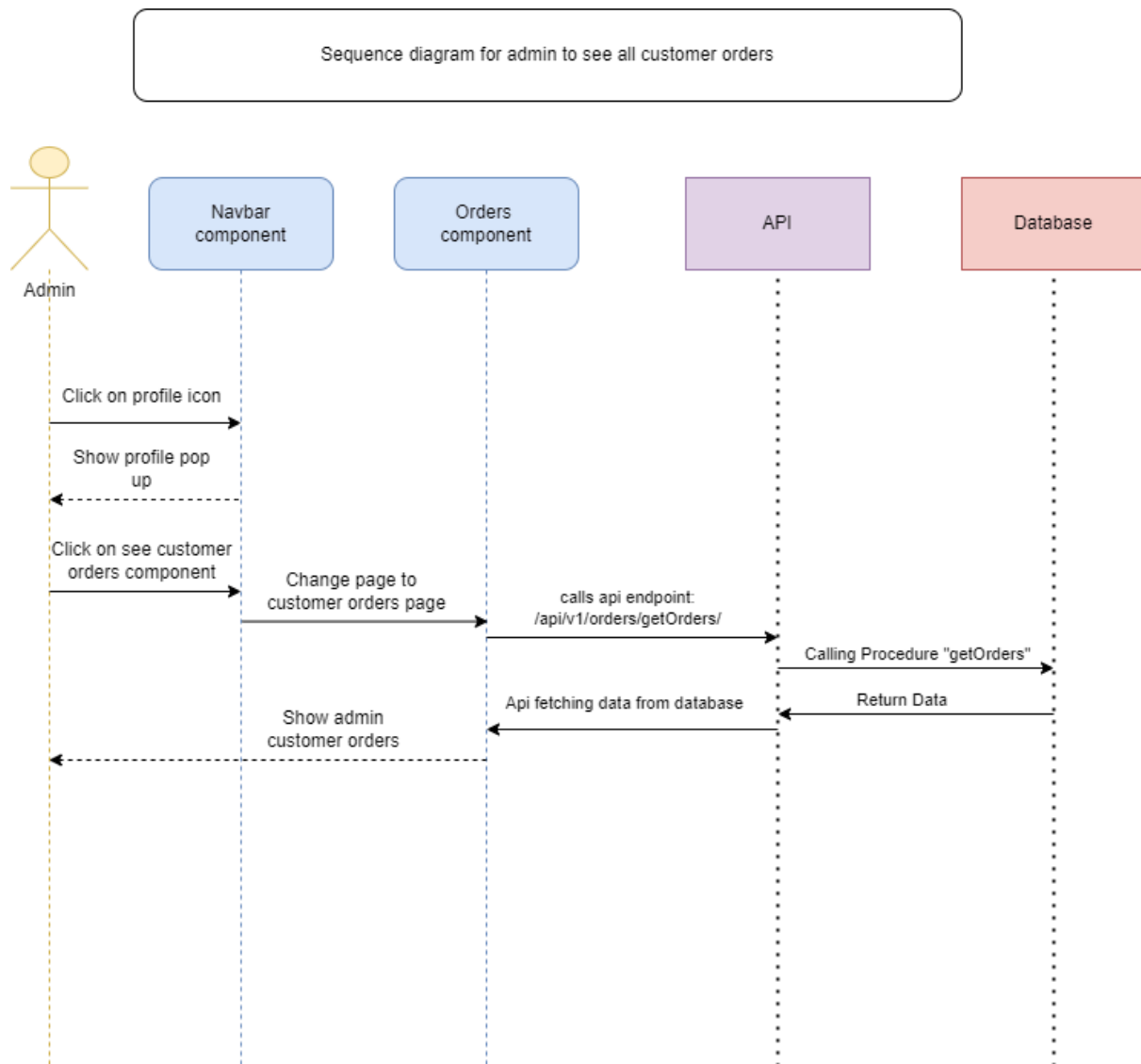
Purchase sequence:



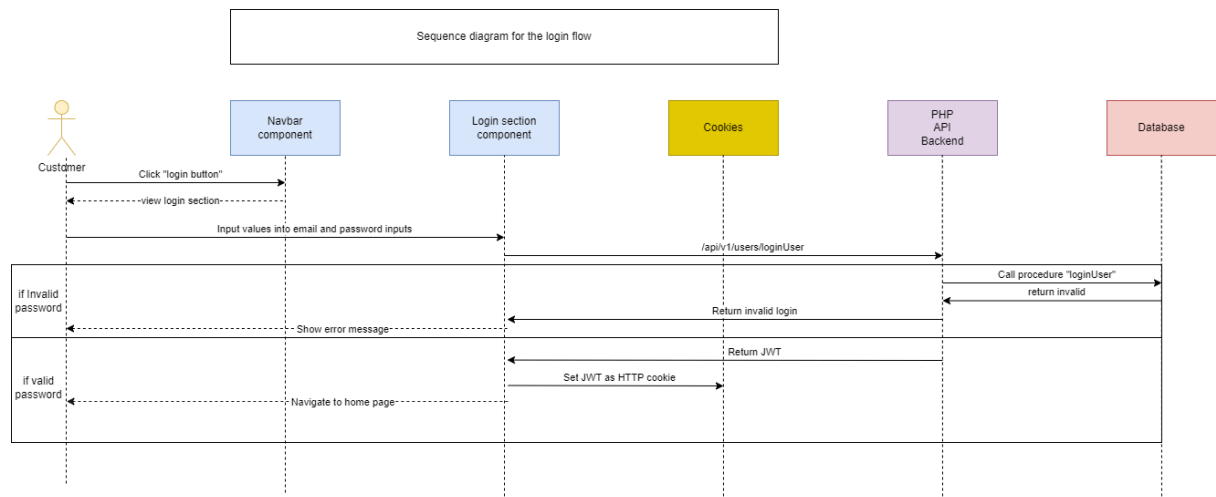
Customer seeing its order:



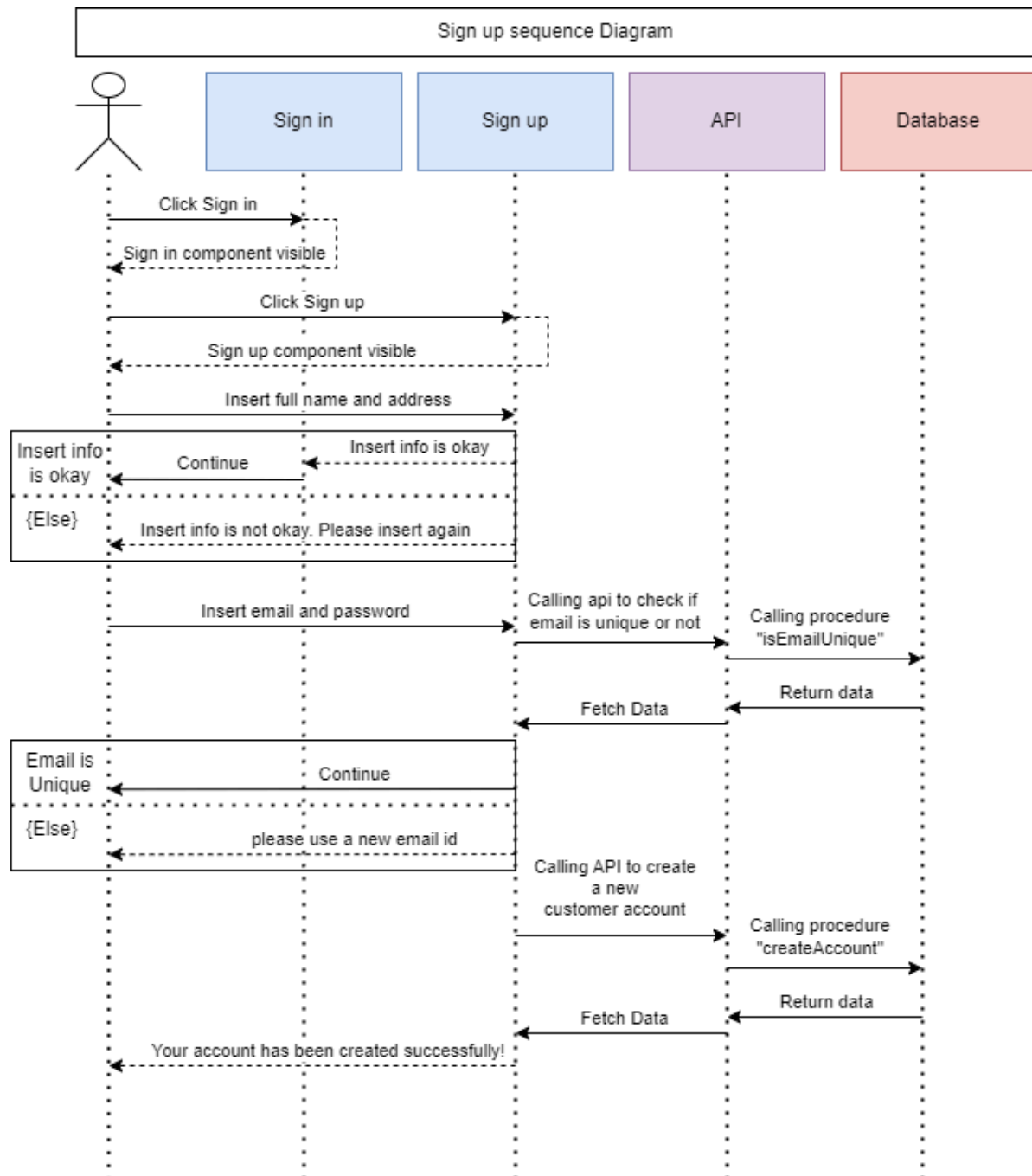
Admin seeing all customers Orders:



Login:



Sign up:



Scrum

Sprints

Sprint 1 - 27/05/2024

We decided that we had to start by making our diagrams.
We split up the work between all of us.

Name	Task
Magnus	Class diagram, Sequence diagram, documentation layout
Marcus	Class diagram, flowchart
Robert	User stories
Shazil	use cases
Yordan	Mockups, wireframe

Our database diagrams were already made, so we did not have to do those.

Sprint 2 - 28/05/2024

There is still work to do, on our diagrams.
We plan to finish all our diagrams today.
As part of our 2nd sprint meeting, we took a look at all our diagrams.
We did this to spot any potential issues, and correct them.

We also figured out all the page routes for our frontend web application.

Sprint 3 - 29/05/2024

We did not finish yesterday, despite having planned to do so.
The main reason for being behind schedule is due to the sequence diagrams.
Today we need to finish:

- Writing documentation
- Sequence diagrams
- Wireframes and mockups

Some diagrams also need a description, which makes it more obvious what happens.
Especially on mockups and wireframes.

Sprint 4 - 30/05/2024

Today we have minimal time due to PHP education.

All group members will follow along with that, except for Magnus.

Magnus will be working on

- Creating a class structure within our php API project
- Sequence diagram
- Class diagram changes

We are also gonna stay after school to work.

We found some new wireframes that need to be done by Yordan.

Everyone else will be working on finishing the sequence diagrams

Sprint 5 - 31/05/2024

Today we plan on finishing all the initial documentation.

Name	Task
Magnus	Write missing documentation. React project structure.
Marcus	Gantt diagram.
Robert	Following along with php lectures, by michael.
Shazil	Minor issues on sequence diagrams. Stored procedure descriptions, in the documentation.
Yordan	Wireframes and descriptions for them.

Conclusion

Overall

Gantt diagram

Log book for gantt diagram:

https://docs.google.com/spreadsheets/d/1H9TuabFe6nXs-_0shB2nS4GinLPiBO_rF5nfBFNeq8I/edit?usp=sharing

Github

The documentation can be found on Github, alongside the actual code.

This includes better images of our diagrams.

https://github.com/MagnusHLund/H2_Davids_book_club/tree/main/Documentation