|  |  |  |
| --- | --- | --- |
|  | **Софийски университет „Св. Кл. Охридски”**  Факултет по математика и информатика  *Магистърска програма „Разпределени системи и мобилни технологии ”* |  |

**Предмет: Разработка на клиент-сървър (fullstack) приложения с Node.js + Express.js + React.js**

***Летен семестър, 2021 год.***

**Курсов проект**

**Тема: „Онлайн E-commerce магазин“**

*Автор:*

*Станислав Илиев*

**1 INTRODUCTION**

E-commerce (electronic commerce) is the buying and selling of goods and services, or the transmitting of funds or data, over an electronic network, primarily the internet. These business transactions occur either as business-to-business, business-to-consumer, consumer-to-consumer or consumer-to-business.

E-commerce shops have become part of our daily lives. Technological advancement has made it possible for people to sit in the convenience of their homes and still shop online without going to a physical shop.

Typically, when it comes to online shops we have two general types of accounts Administrators and Customers/Users.

* The store manager and the staff members operate as the administrators. They can add, edit, update products or, delete products thus they able to change the names of products, change prices and, add or remove products.
* The customer can search for products, update the cart, remove products from the cart and check out from the shop. The customer is also able to update his information such as names, address and other data.

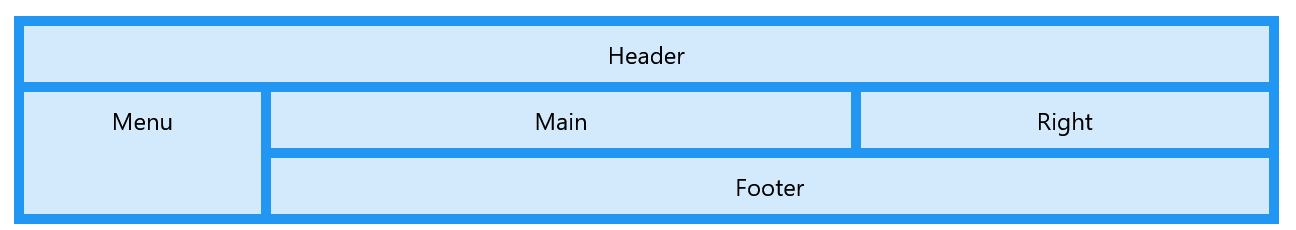
**2 TOOLS AND TECHNOLOGIES**

These are the necessary tools and materials needed to build the website both the front-end and the back-end. These include software and open source materials.

* **HTML5 and CSS3: Semantic Elements, CSS Grid, Flexbox**
* **Redux: Store, Reducers, Actions**
* **NodeJS & Express: Web API, Body Parser, File Upload, JWT**
* **MongoDB: Mongoose, Aggregation**

**HTML** means Hypertext Markup Language. This language is used in creating web pages and also supports other languages such CSS, PHP, JAVASCRIPT, etc. in creating interactive and responsive pages on the pages. HTML5 is just an updated ver-sion of the HTML. It supports new features, new attributes, new HTML elements and full CSS3 support.

**CSS** stands for Cascading Style Sheets. CSS is a standard style sheet language used for describing the presentation (i.e. the layout and formatting) of the web pages. CSS was designed to enable the separation of presentation and content. Now web designers can move the formatting information of the web pages to a separate style sheet which results in considerably simpler HTML markup, and better maintainability. CSS3 is the latest version of the CSS specification. CSS3 adds several new styling features and improvements to enhance the web presentation capabilities.

**Grid** Layout - CSS Grid Layout is a two-dimensional grid-based layout system that aims to do nothing less than completely change the way we design grid-based user interfaces. The CSS Grid Layout Module offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning. 

**Flexbox** module aims at providing a more efficient way to lay out, align and distribute space among items in a container, even when their size is unknown and/or dynamic. The main idea behind the flex layout is to give the container the ability to alter its items’ width/height to best fill the available space. A flex container expands items to fill available free space or shrinks them to prevent overflow.

**Javascript** is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. Over 97% of websites use it client-side for web page behavior, often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute the code on the user's device.

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM). React: Components, Props, Events, Hooks, Router, Axios

**Redux** is a predictable state container for JavaScript apps. It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test. React components has a built-in state object. The state object is where you store property values that belongs to the component. When the state object changes, the component re-renders.

**Store** is just an object with a few methods on it. A store holds the whole state tree of your application. The only way to change the state inside it is to dispatch an action on it. A store is not a class.

**Reducers** are there to manage state in an application. For instance, if a user writes something in an HTML input field, the application has to manage this UI state (e.g. controlled components). In essence, a reducer is a function which takes two arguments -- the current state and an action -- and returns based on both arguments a new state.

**Actions** are plain JavaScript objects that has a type field. You can think of an action as an event that describes something that happened in the application. The type field should be a string that gives this action a descriptive name. Actions are the only source of information for the store as per Redux official documentation. It carries a payload of information from your application to store.

**Node.js** is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux. Node.js is designed to build scalable network applications. It is Asynchronous and Event Driven. All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.

**Express**, is a back end web application framework for Node.js. It is designed for building web applications and APIs. Express.js can reduce the coding time by half and still help us build efficient web applications. It not only reduces the time but it also reduces the effort required to build web apps with the help of its different features.

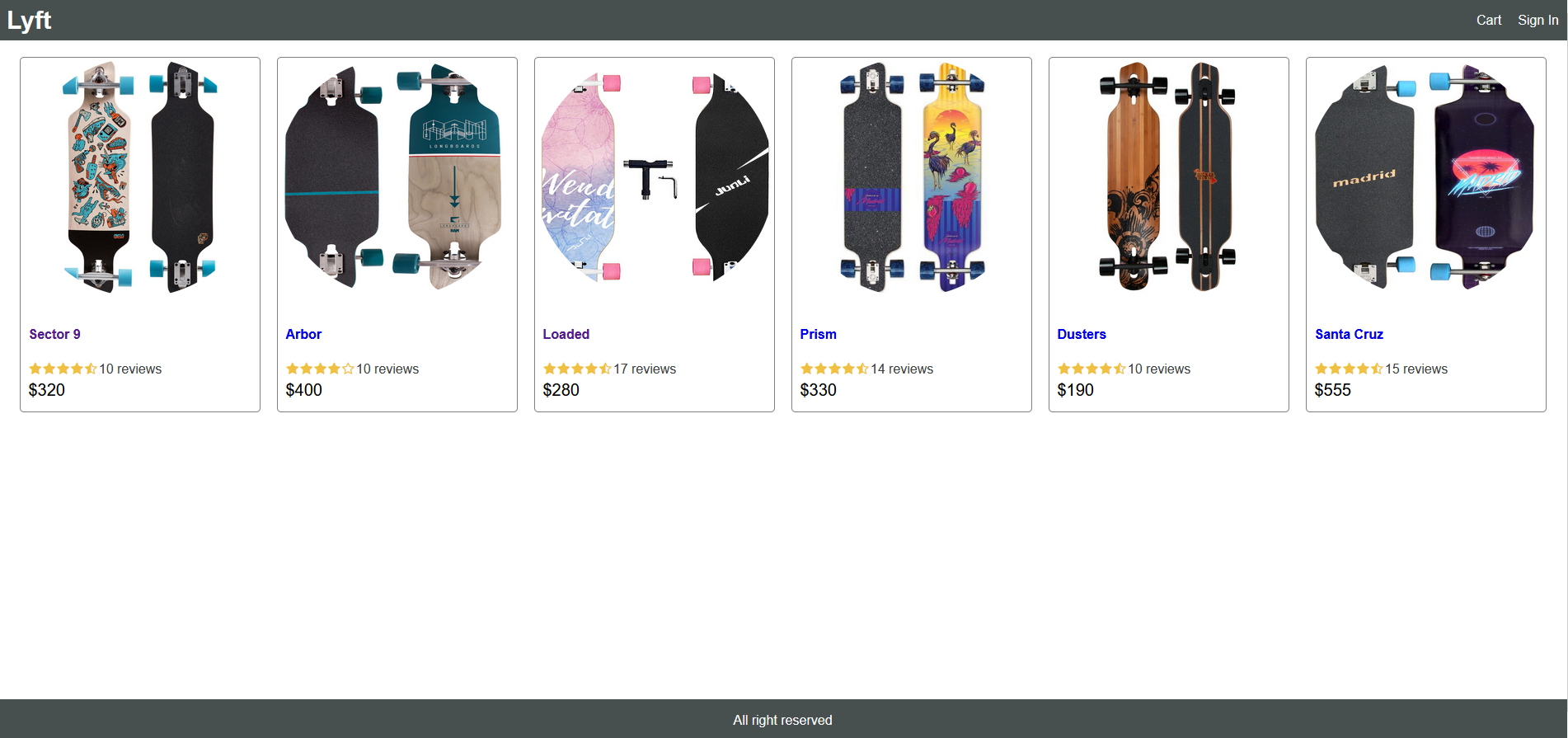
**JWT** - JSON Web Token is a means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is digitally signed using JSON Web Signature (JWS) and/or encrypted using JSON Web Encryption (JWE). The tokens are signed either using a private secret or a public/private key.

**MongoDB** stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time. The document model maps to the objects in your application code, making data easy to work with. Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyze your data. MongoDB is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use.

**Mongoose** is an Object Data Modeling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.

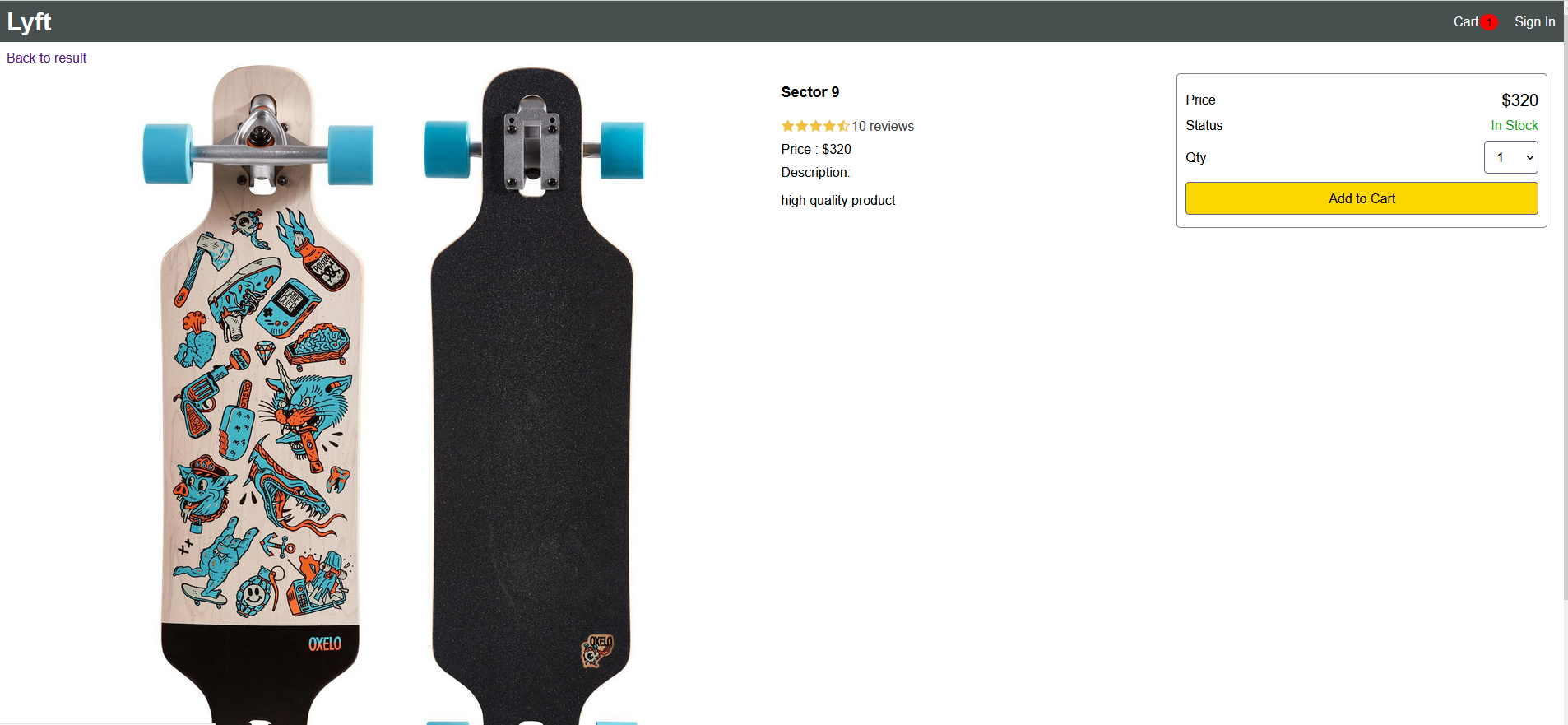
**3 VIEWS AND SCREENS**

**Home screen –** when the user first loads the website he/she will see the home screen. “Lyft” is an online shop for longboards. And the home screen is where the products are displayed.

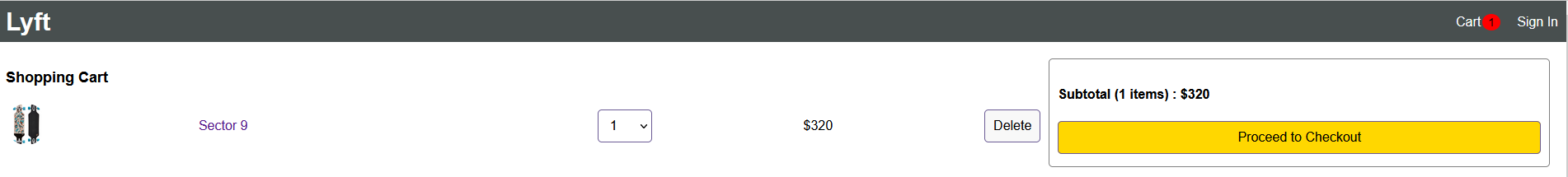


**Product Screen** - If the user wants more information regarding a specific product they can click somewhere inside the frame of the product and it will redirect them to Product Screen for a more detailed look. The user can find the following information about the product in this page:   
Full name of the model; Rating; Price; Description; Whether it is in or out of stock;

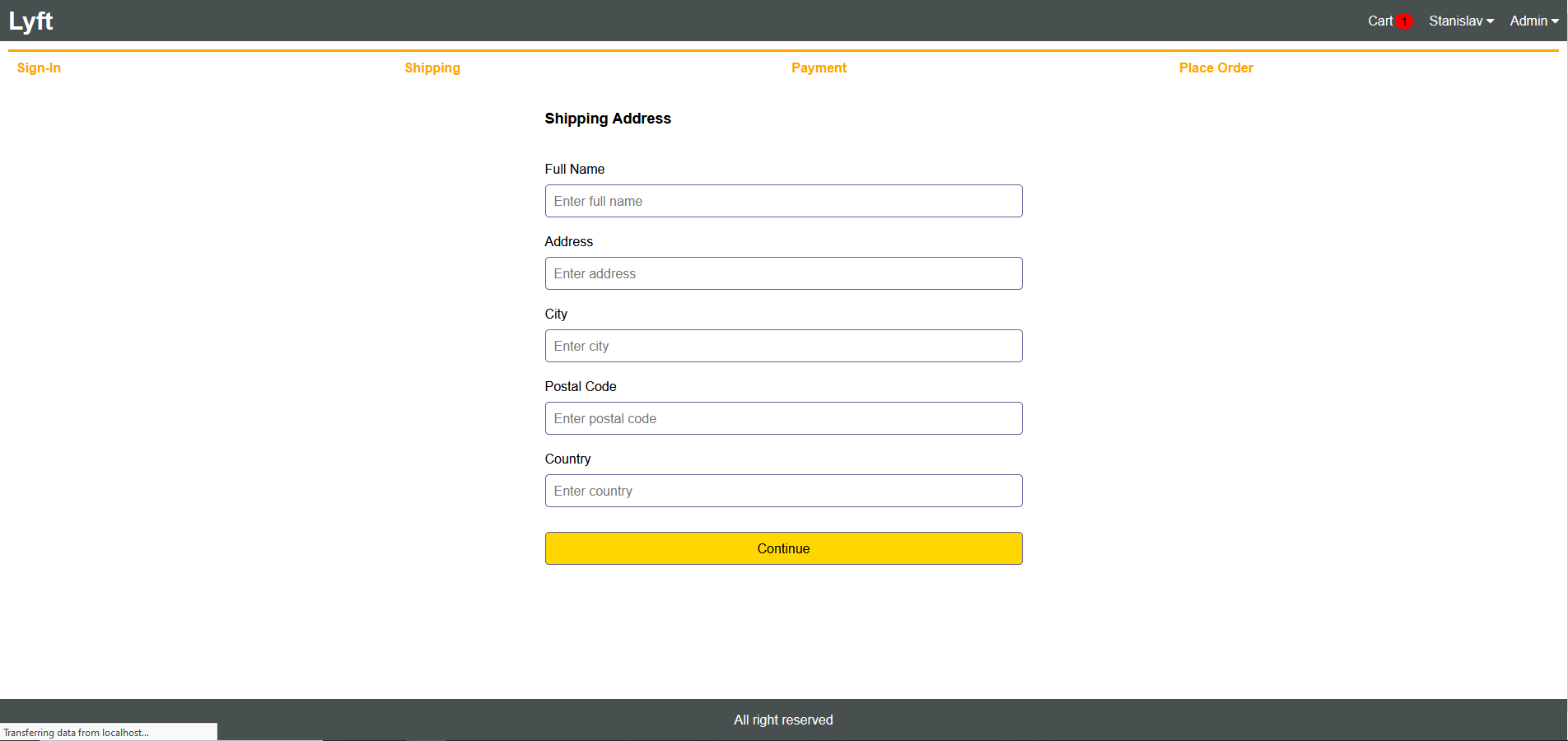
In case the customer decides to buy the product he/she can select the quantity needed and proceed by Adding the products to the cart.



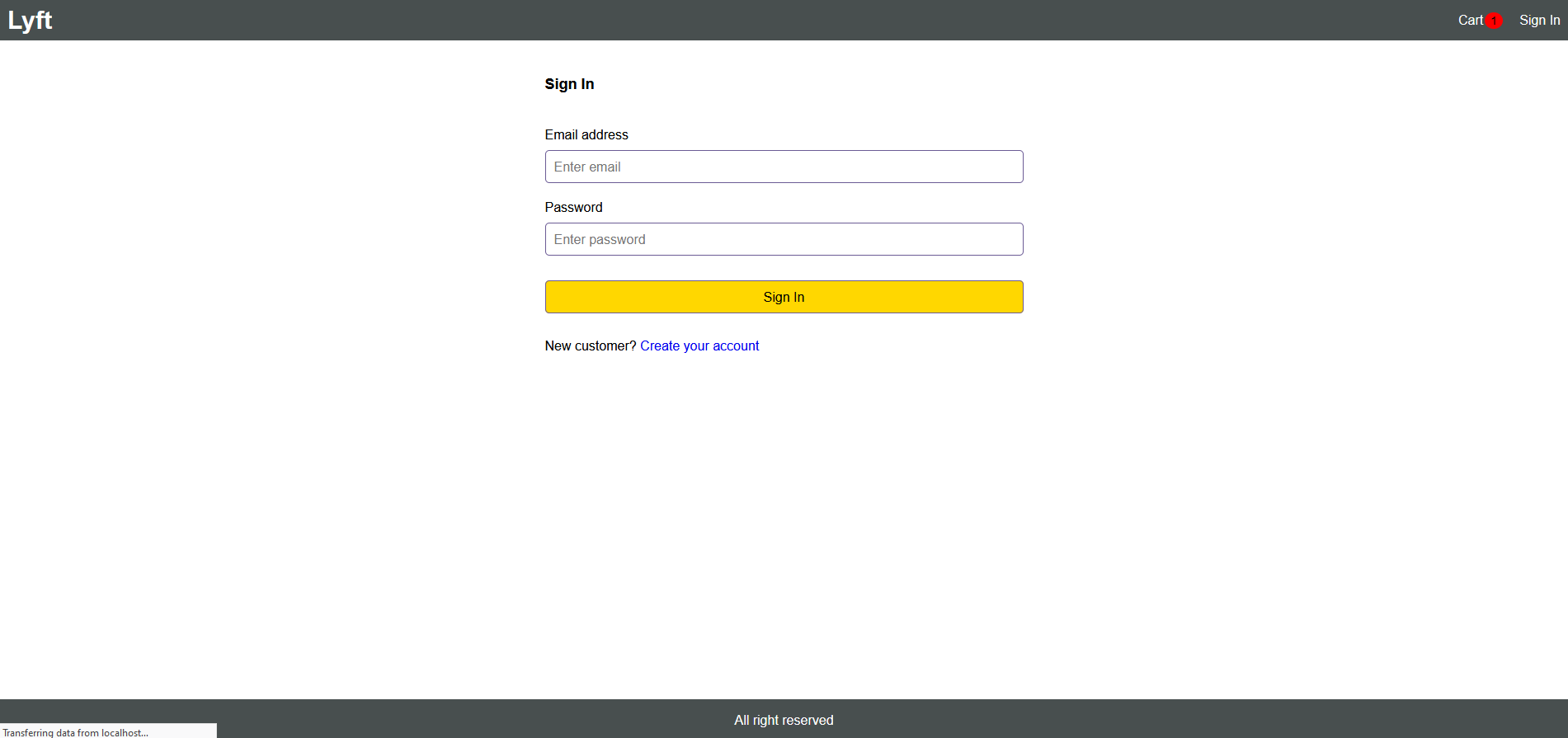
**Cart Screen –** This page contains the full list of the products that the customer wants to potentially checkout and buy. There is an option to increase or decrease the quantity again. On the right side there is a box that shows the sum of the prices and there is a button called Proceed to checkout which redirects to the Checkout section.



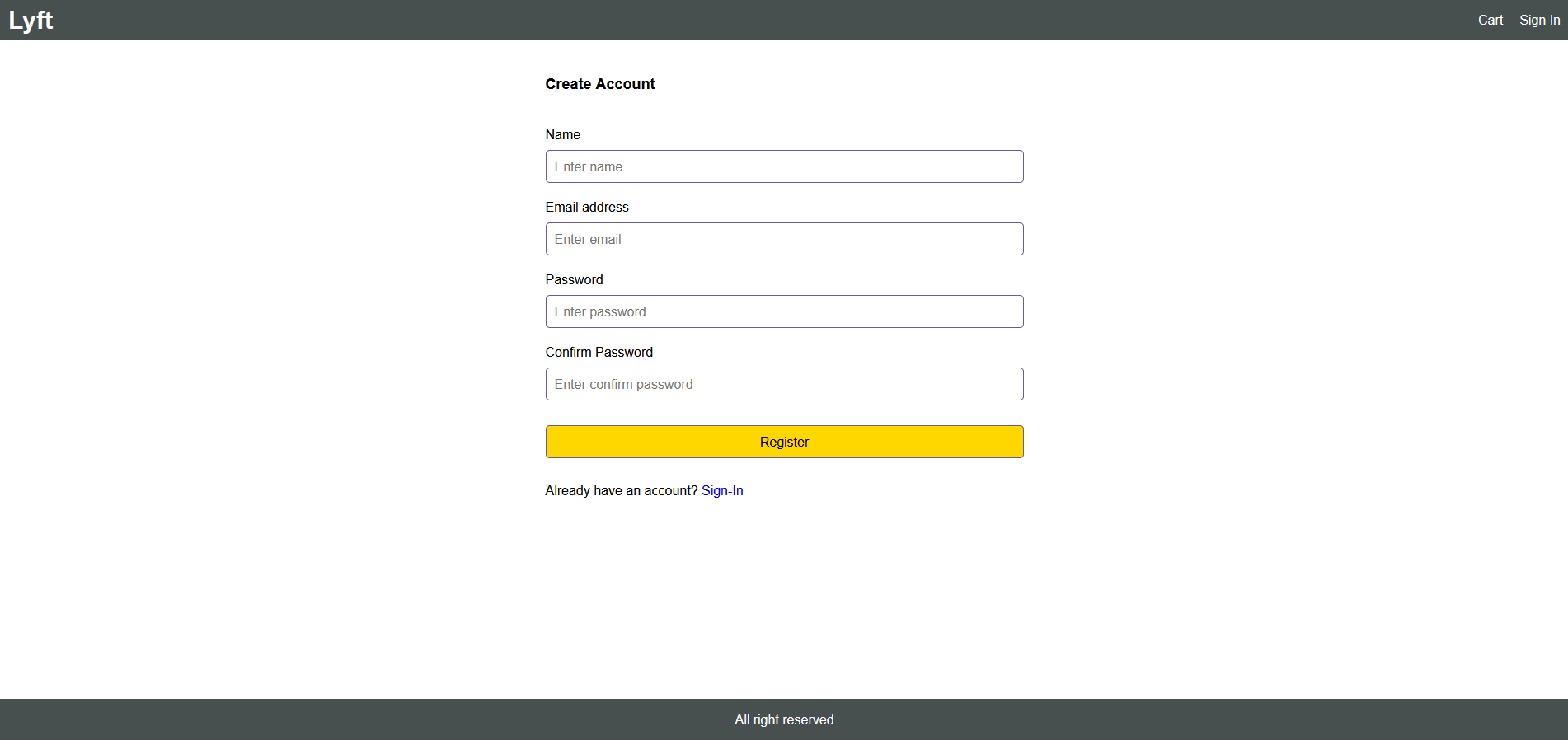
**Shipping screen –** This page is available only if the user has completed a registration process and has successfully logged into his/her account. Otherwise they are redirected to the Sign in screen. In this page the user has to fill the needed information in order to continue to the Payment method screen.

****

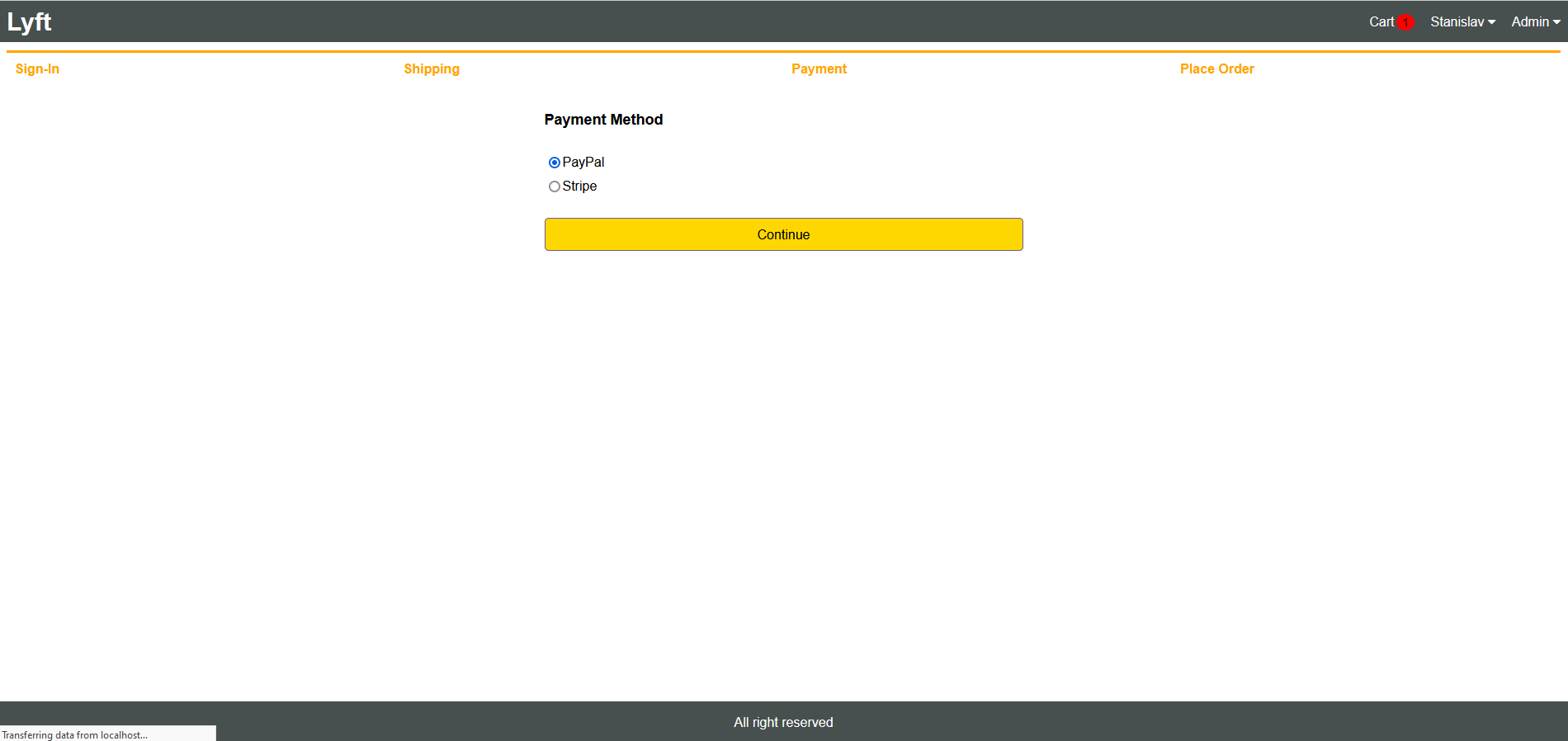
**Sign in screen** – In this page the user has to give the email that was used during the registration process and the corresponding password. If the user hasn’t completed the process he can choose to go to the Register screen and create a brand new account.



**Register screen –** A user has to complete this procedure in order to submit the order. All of the fields are required to complete the registration and the Password and the Confirm password fields must match perfectly.

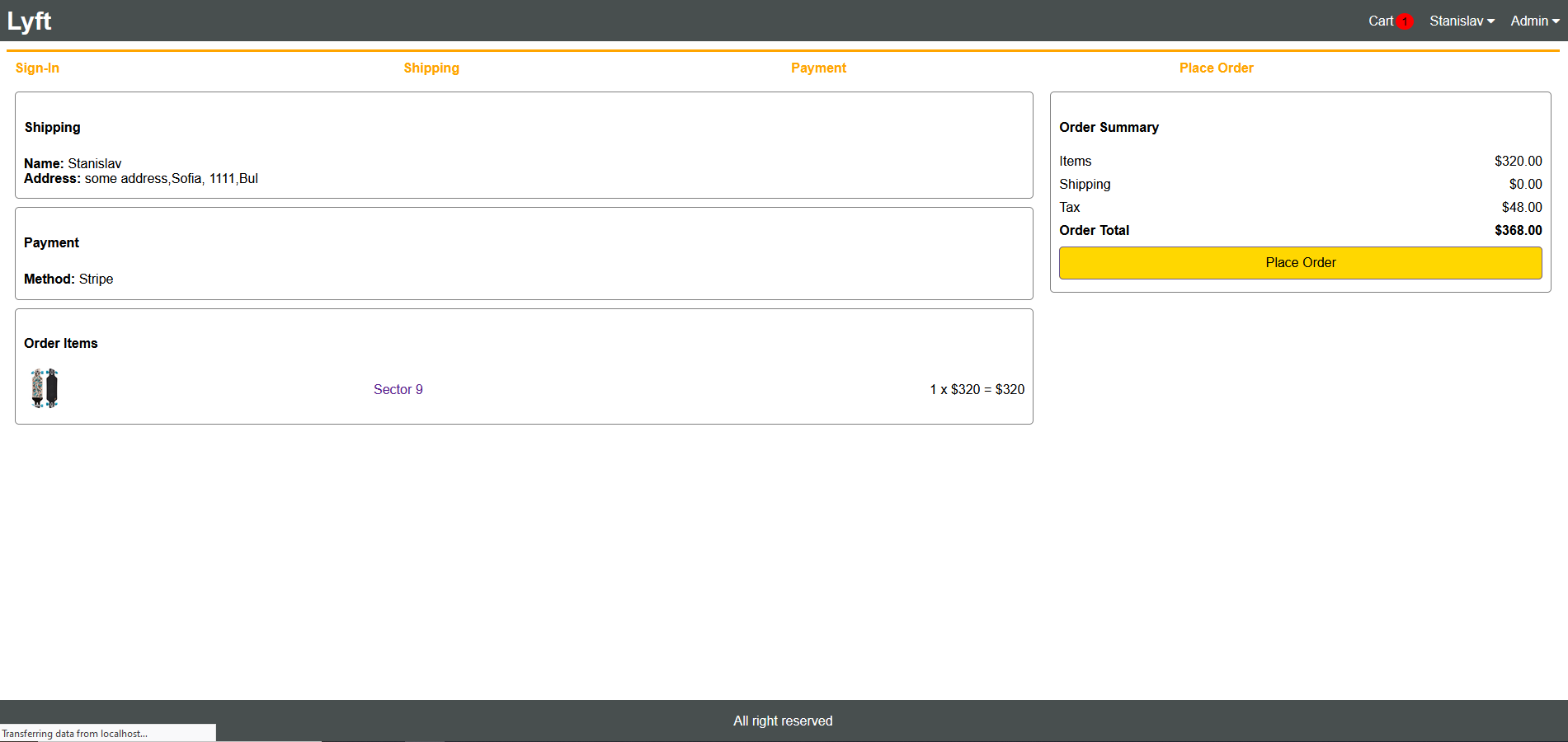


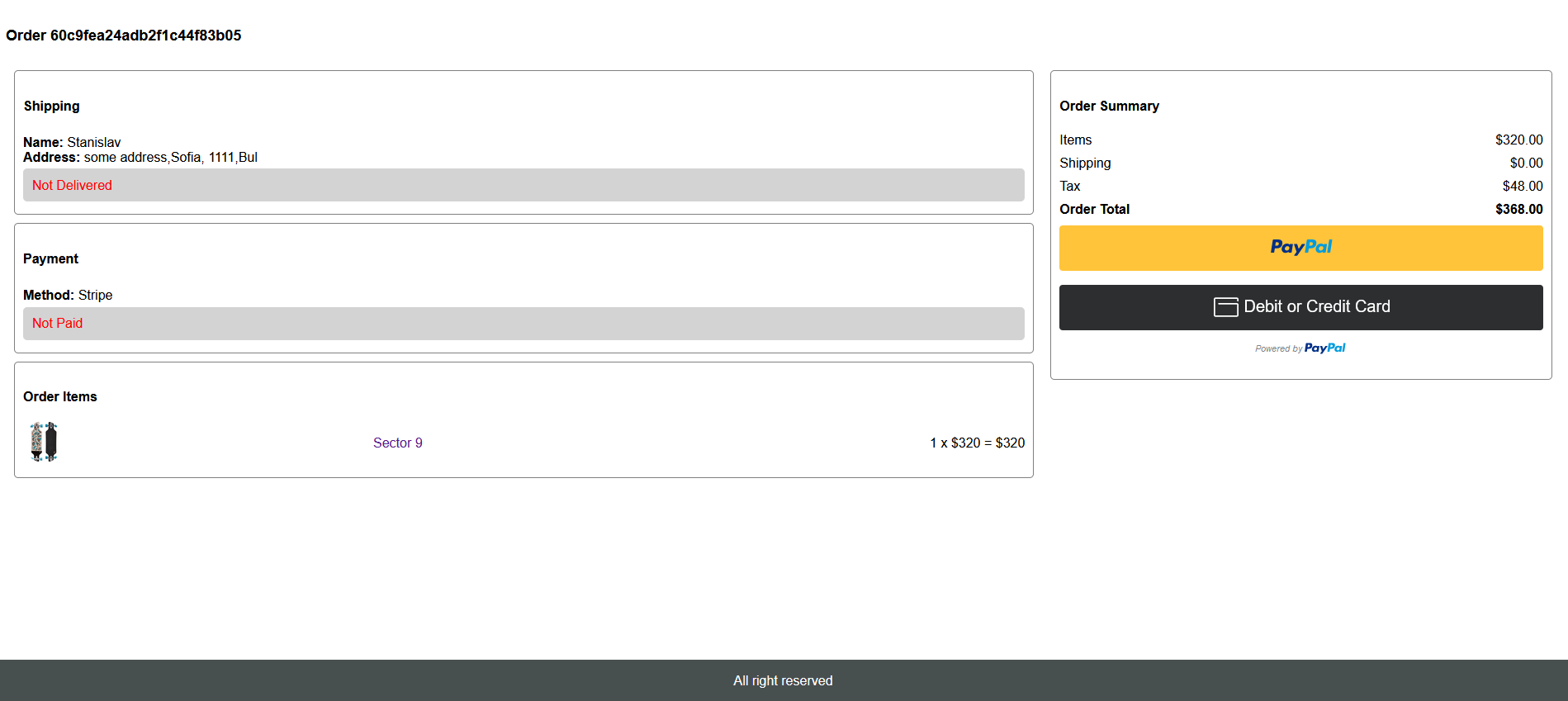
**Payment method screen** – Here the user picks the desired method to pay. There are two options Stripe and Paypal.

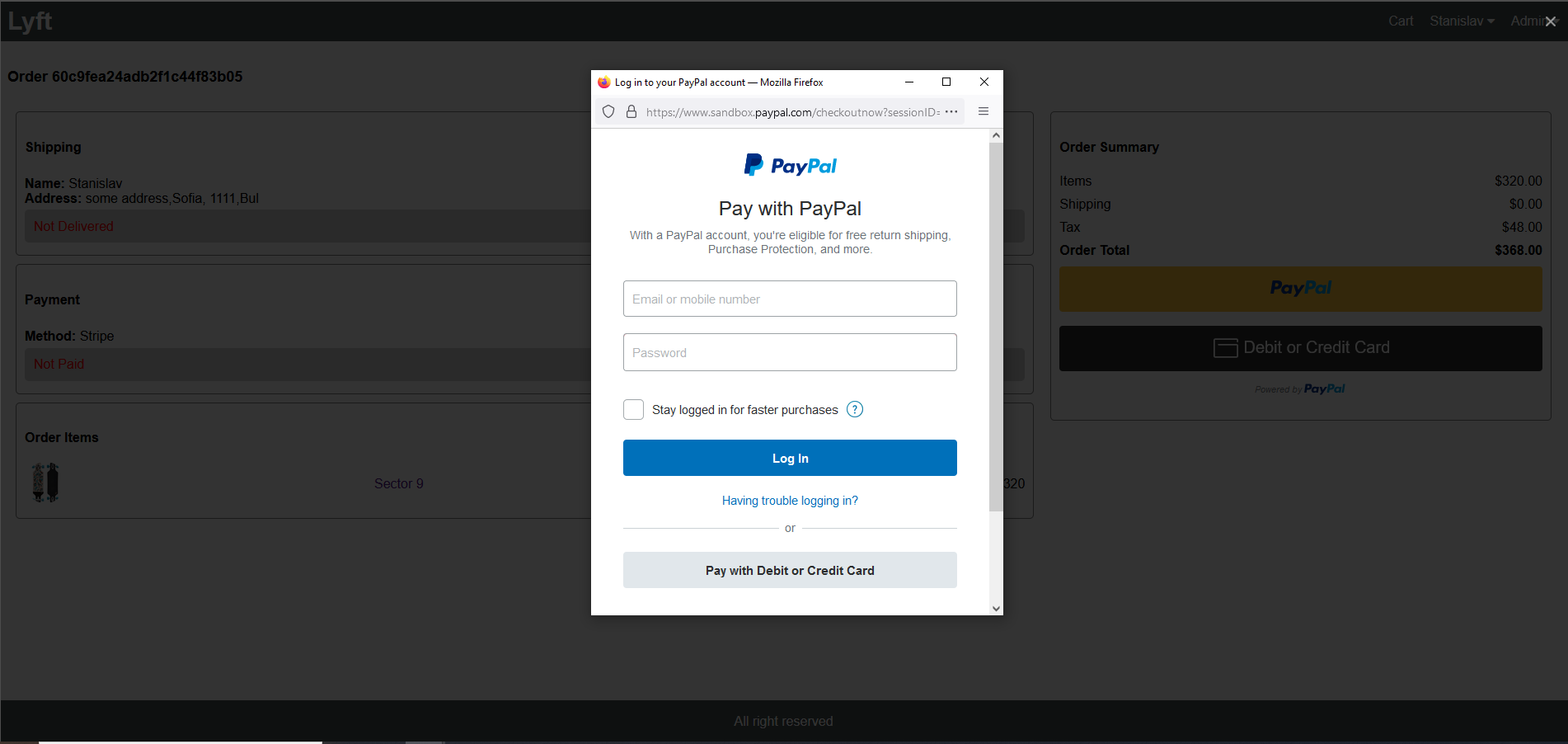


After picking one of the two you are redirected to the Place order screen.

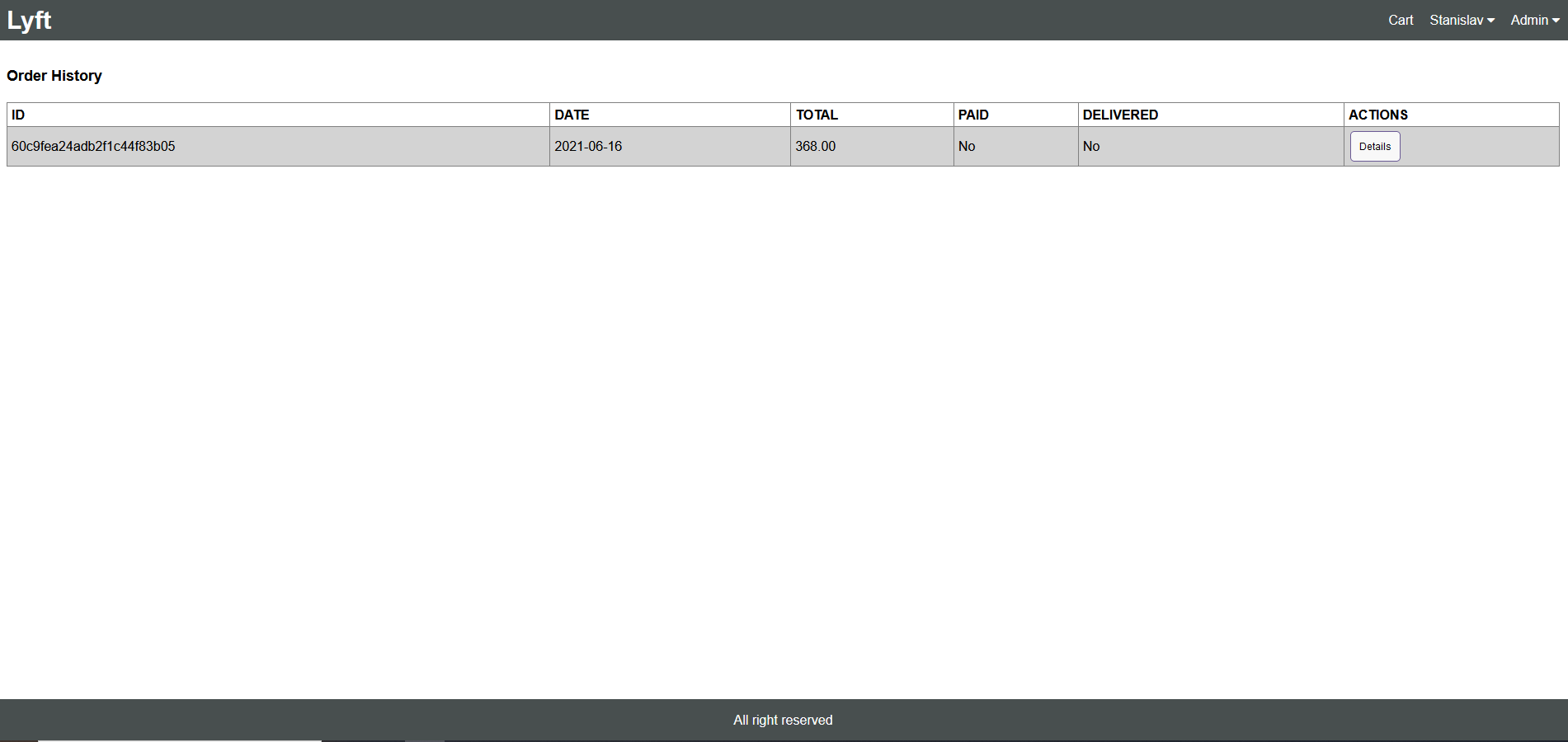
**Place order screen** – if all of the above is already set, the user is redirected to this page. Here the customer can see the shipping information, the selected payment method, the products that he is about to buy and the total order price. If he/she confirms that everything looks legitimate they can click on Place Order. That will redirect them to the specific Order screen for more detailed information where they can track the status of the order.

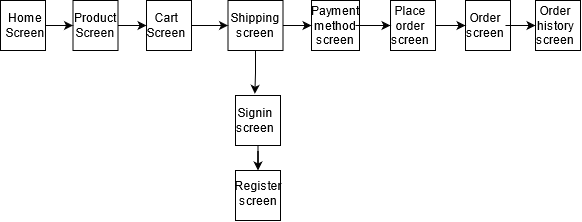


**Order screen –** Here the customer can view the status of the order. He/she could check if the order is paid, if the order is delivered and the price that it costs. If the customer has not paid yet he has to select the Paypal or Debit card button. That will trigger a pop-up window to appear prompting for the correct information. After the payment is completed the status will change. All orders that a user does are saved in the Order History screen.



**Order history screen –** This screen provides a detailed information about the history of purchases that were made by the user. Here customers can find the ID of their order; the date when it was made; the total cost; and the status.





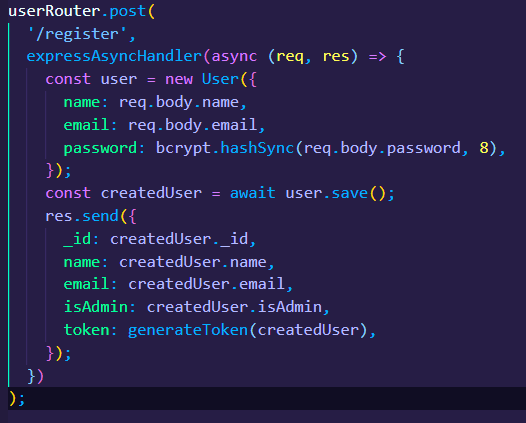
**4 ROUTERS**

A Javascript router is a key component in most frontend frameworks. It is the piece of software in charge to organize the states of the application, switching between different views. For example, the router will render the login screen initially, and when the login is successfull it will perform the transition to the user’s welcome screen. The router will be in charge of simulating transitions between documents by watching changes on the URL. When the document is reloaded or the URL is modified somehow, it will detect that change and render the view that is associated with the new URL. You define routing using methods of the Express app object that correspond to HTTP methods; for example, app.get() to handle GET requests and app.post to handle POST requests. These routing methods specify a callback function (sometimes called “handler functions”) called when the application receives a request to the specified route (endpoint) and HTTP method. In other words, the application “listens” for requests that match the specified route(s) and method(s), and when it detects a match, it calls the specified callback function.

**User Router –** In this file we have multiple router functions. We will take a look at the one that handles the registration process. We previously discussed the frontend part for the Register Screen, now let’s see the router that is configured in the back-end.



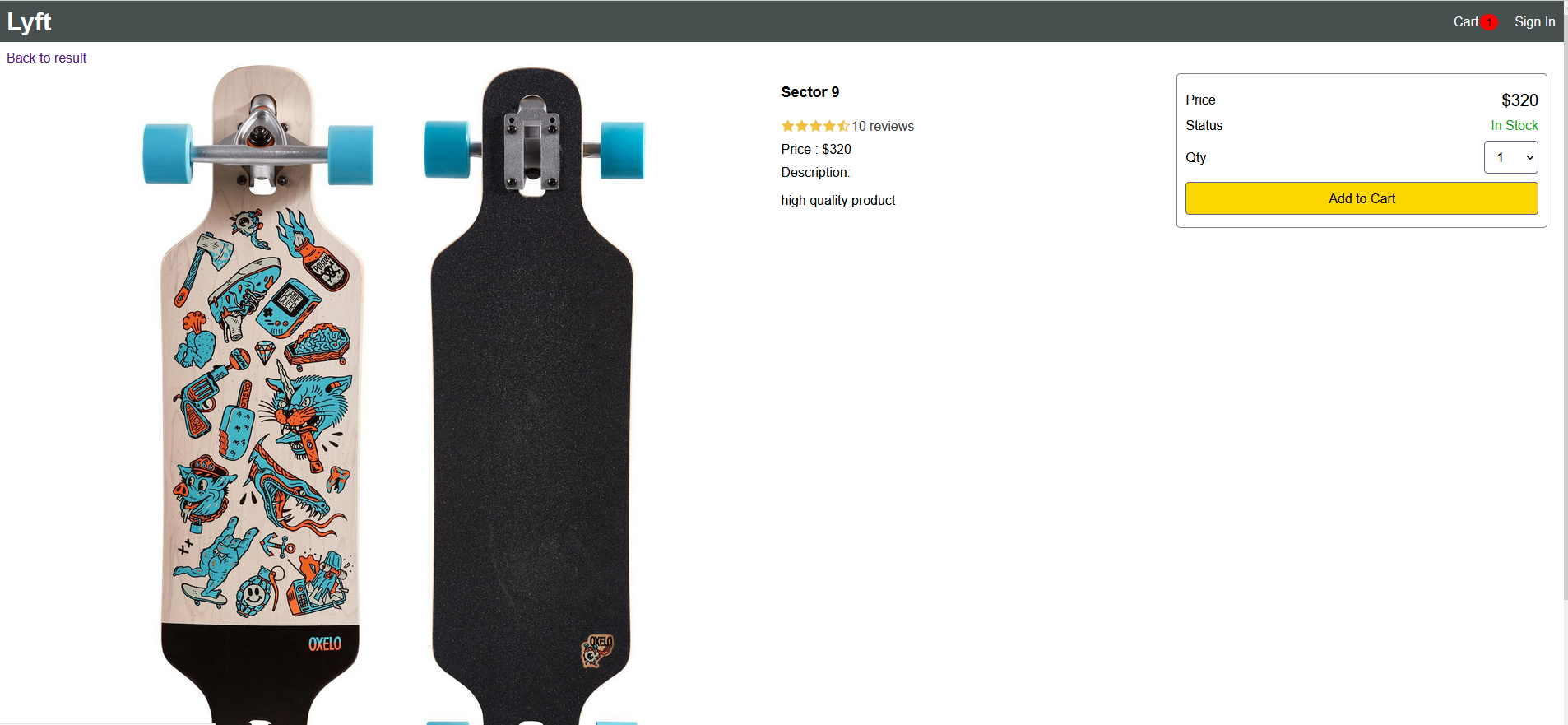
What we can see from the start is that the userRouter uses HTTP POST request. It is associated with the “/register” path in the URL.



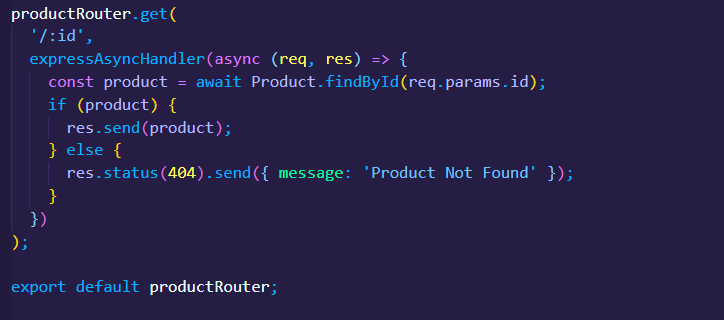
It waits for the user to complete the whole registration process to invoke the user.save method. That method sends the credentials provided to the MongoDB instance.

Let’s look at another example using the Product Router.

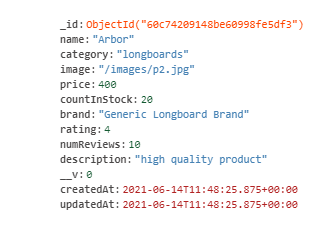
C:\Users\Adin\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ProductRouter1.png**Product Router –** Each product has an ID that is being assigned to it in order for the backend to differentiate them. When a user decides to visit a products page we see at the URL the string that represents the ID. Let’s open a random Product Screen.



Immediately we notice that the URL has changed to “/product/60c74209148be60998fe5df3”. The router responsible for loading this information in the frontend uses the .findById functionality to check if there is a product with that ID inside the database. If there is a match it loads the page. If there is no match it will return a status code 404 and the error message.



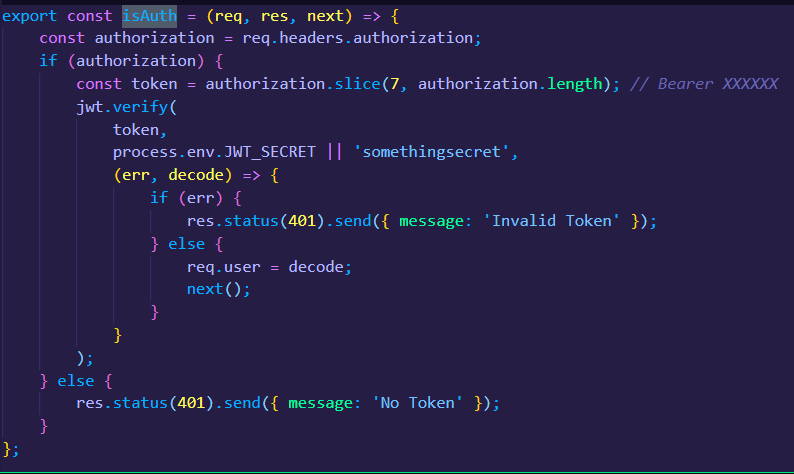
If we take a look inside the database instance, we can also see the same ID:



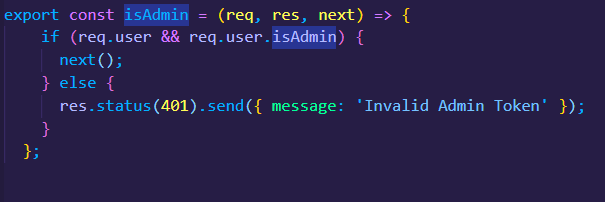
**5 UTILS**

The utils.js file is used to define utilities that are invoked frequently when you need to check if a user I signed in, or whether the profile that is being used is Admin or a regular user. This usually is the case when a user tries to do an operation that can be executed only if the user is signed in their profile. Example for such operation is the Checkout procedure. The user can’t give shipping information if he/she hasn’t logged in their profile.

One example is the “isAuth” function which verifies that the user has been authenticated.



Another example is “isAdmin” which checks if the profile that is being logged into has Admin privileges.

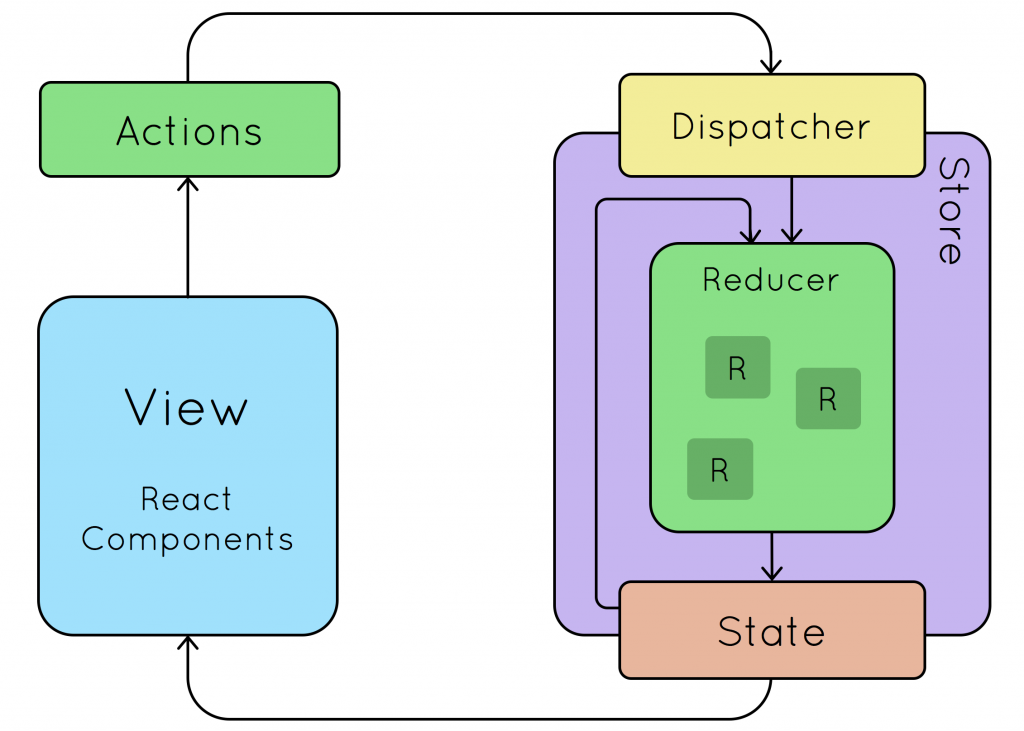


**6 REDUX / REDUCERS**

The Redux Store keeps the application state. This state can be updated by dispatching actions. An action is nothing more than a JSON object with a type and some parameters.

Example: { type: ‘SET\_DATE’, date: ‘2017-03-15T13:24:00.000Z’ })

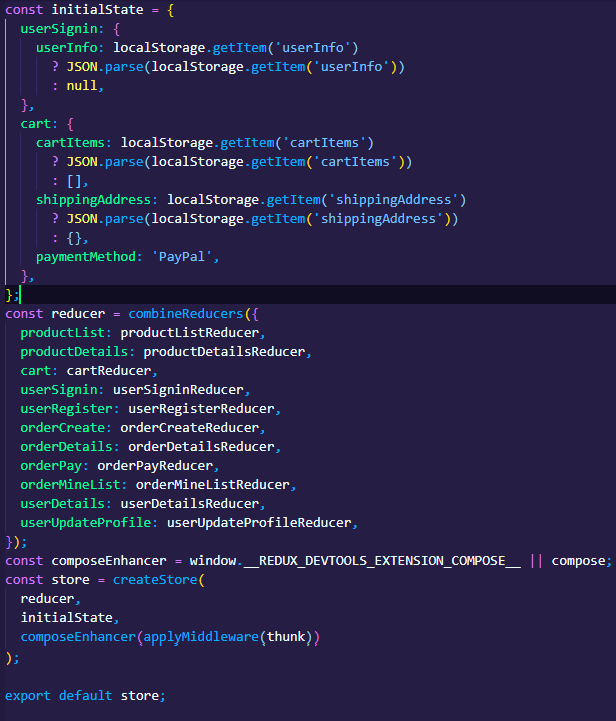
The store has a reducer, which, taking into account the action and the previous state, will produce the next state. React lets you build components that react to changes of the application state. Components affected by a state change are re-rendered with the new data. Components also dispatch actions, for example when a button is clicked.



What we are doing is to dispatch an action in the “View” part and this action sends a request to the Redux store to make a change in the state of our application. So to request a state change we need to create an Action and Dispatch that action. In the store of Redux we have **two** **things** – a **State** and a **Reducer**.

State is the current state of our application and the Reducer is a function that gets the current state; makes a change in the state and returns a new state at the end.

By having a history of states we can better manage and predict the state of our application, hence less unpredictable status. That is the benefit of having Redux.



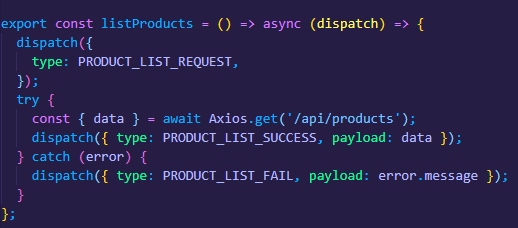
In this figure we see the Redux store.

Reducer is a function that accepts two parameters. State is the first one and Action as the second parameter and it returns the new state.

createStore is a build-in function from Redux. It accepts reducer and initial state as parameters. The third bonus parameter uses Redux-Thunk extension which makes it possible to send AJAX request in our Redux actions.

For example, we will look at the productAction file.

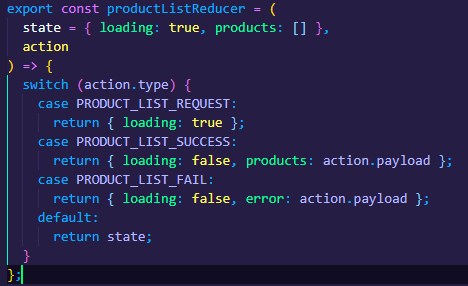
listProducts is an action, an action is a function. It returns another async function that accepts dispatch as a parameter. First we need to dispatch PRODUCT\_LIST\_REQUEST (which is a constant). The next step is to implement a try-catch.



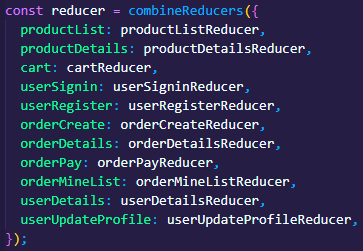
In the try we have an AJAX request to get list of products from the back-end. After that we need to dispatch another action which accepts PRODUCT\_LIST\_SUCCESS (which is another constant) and payload. Payload will contain the data from the back-end.

For the catch part we need to dispatch a fail scenario. It will accept PRODUCT\_LIST\_FAIL (constant) and payload with an error message.

To this Actions we have a Reducer to respond to these actions. Let’s look at productListReducer.

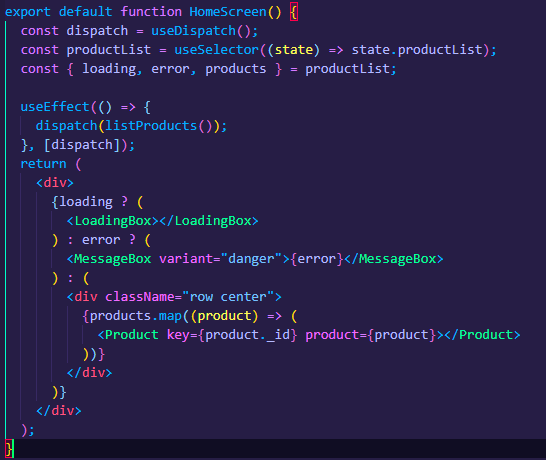


It is a function that accepts two parameters – state and action. In the body of that reducer we have a switch-case. The default case returns the previous state which means it hasn’t made any changes to the state. The Success and Fail cases use the payload that we get from back-end.



We have also added the productReducer in the store. combineReducers accepts a parameter which is an object. This object introduces the reducers to the Redux store.

In order to get the object from the Redux Store, in the Home Screen we need to use useSelector, which is a built-in function in React-Redux. useSelector accepts a function that has redux state as a parameter and from this parameter we just need to get productList. From the productList we are getting the three values – loading; error and products. What we need to do after that is dispatch an listProducts action (function).



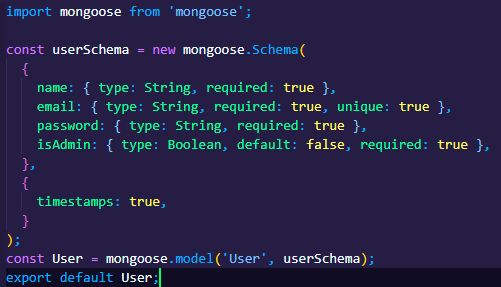
**7 MODELS**

The models basically define the Mongoose schema. The schema defines which fields and what type of data would be required. Essentially here we configure what data we would collect.

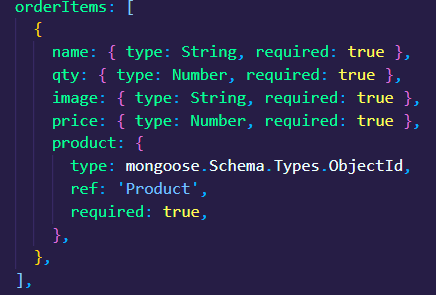
For example, by the user schema we can create a user model, and user model should be a collection in MongoDB.

userSchema is an object from mongoose.Schema. mongooseSchema accepts an object which contains all fields for the user.

In the models we can set the data types of the corresponding data fields. For example, the name of the user needs to be a string. By having required to true, Mongoose does the validation for us when it comes to creating a new user entry to the database. The email has the unique property set to true, which makes sure that no two users have the same email string.

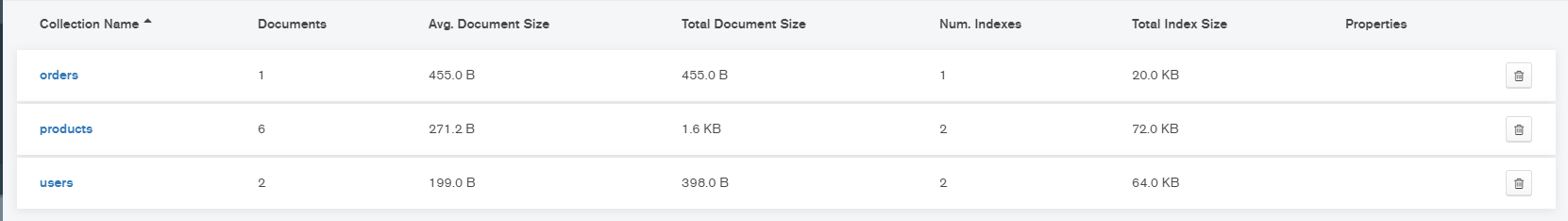


At the bottom we create the model based on the schema. mongoose.model is a function that accepts two parameters – the model name and the schema name.



Another example for why we set the data types is: In order to use the price for future calculations we would like to get it as a Number data type.

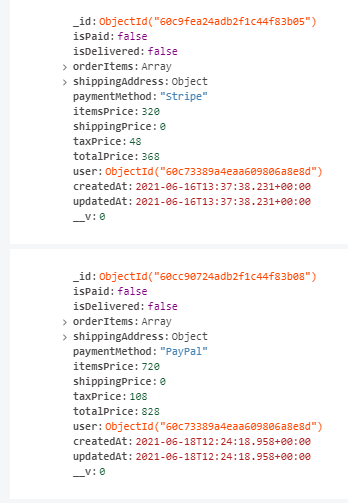
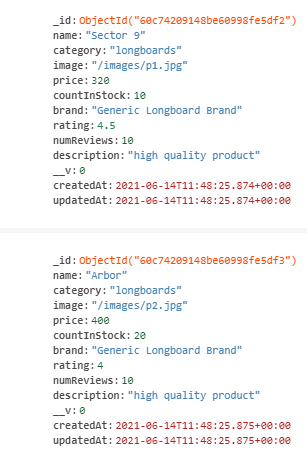
**8 MONGODB**

MongoDB database is used to save software data for this project. There are 3 main collections of data that we save – orders data; products data and user data;



In the users collection we save data that was provided by the user from the registration process. The password is encrypted and doesn’t show as plain text for security reasons. We also save the date and time when that user entry was created and updated and a specific id which is used to identify the different users.

In the products collection we store all of the information about the product that the user needs like – name of the product, the category, the price, the quantity of the product that are available for purchase, the rating, the number of reviews and the description. As well as the date and time when that user entry was created and updated and a specific id which is used to identify the different products.



In the orders collection we store all of the information about every purchase that has happened. We story information about the product ID; if the order was paid; if the product has been delivered to the customer; the payment method selected by the user; the total cost; and the ID of the user that has made the purchase.

**8 CONCLUSION**

Any consumer can log in, with his information such as his email and password. Any consumer can browse products, add, replace or delete a product from the cart. If the user has no account, the user can register. After login, the user can see the product in the cart and proceed onwards with providing shipping information and selecting payment methods. The product can be paid with PayPal and Stripe. The full information about the order is saved in the database and could be found in the order history screen under the user’s account.

**9 FUTURE IMPROVEMENT**

* Invoices need to be implemented in the shop. The customers need to receive emails and notifications for new arrivals or discounts.
* The shop needs a search engine where users can search for the product by name.
* Also there needs to be filter that allows users to sort the products by different criteria.
* Debit and credit card checkouts need to be implemented too.
* There has to be language options so that none-English speakers can shop without any difficulty.
* Admin account privileges need to be implemented, which will simulate a shop owner. For example, only admin account should have the ability to add and remove products in the shop.
* Users should be able to write comments under every product. To voice their opinion about the quality of the product and the service.