E-Commerce Website

International Bot

Group Members:

Choon Wei Tong	17205786
Shijie Liu	18439314
Chee Guan Tee (Jason)	18202044
Simonas Ramonas	18763829

Requirements

User Requirements

Req #	Title	Comment	Status
G0	View the products currently on offer	Users can click the button labelled ‘Offers’ located in the drop down under ‘Products’ in the header bar, once clicked, the user will be directed to an Offers page where all discounted products are displayed. Users may also click on the ‘View More’ button in the home page under the Offers section.	COMPLETED
G1	View details about a particular product	Users can click on any of the product cards to be redirected to the specific product’s page which contains full details of the selected product and have options to add to cart.	COMPLETED
G2	Add a product to their shopping cart	Users can click on the ‘Add To Cart’ button to add any product to their cart. This option is present on the Product card display and within the product’s details.	COMPLETED
G3	Remove a product from their shopping cart	Users have to visit their shopping cart by clicking on the cart icon on the nav bar which is located on the top right side of the page. They will be redirected to a ‘Cart’ page which will contain all items that are added to their basket, along with the remove button that will remove a product from their cart.	COMPLETED
G4	Search for a product	The nav bar contains a search bar that users can input a product’s name or keyword to search for related items. Relevant products will show up as the user types. If the Enter key is pressed, they will be redirected to a search page which contains all the results of the relevant products.	COMPLETED
G5	Create a customer account	Users can sign up by clicking on the ‘Sign up’ button located on the nav bar. They will be redirected to a ‘Register’ page using Spring Security which contains a form that will POST details to the database, storing the customer account’s details. Users can only use email addresses that have never been used before to create an account. This form validates all the input provided by the user. If there is any invalid input, the form will not be posted and the user will be prompted to make any necessary changes. Password is also hashed securely in the database.	COMPLETED

G6	Log in to their account	Users can click on the 'Login' button on the navigation bar to log in to an existing account. They will be redirected to a login page that contains a form that will authenticate user account details before allowing access into the account.	COMPLETED

Customer Requirements

Req #	Title	Comment	Status
C0	Go to the checkout to purchase orders in their shopping cart	Customers will be able to checkout from their cart. Users have to be registered to an account in order to checkout. If they are a guest user instead, they can save their items in the cart temporarily. When they register for an account, all items that were in the cart during their guest session will be carried over to their account's cart instead.	COMPLETED
C1	Use a (fake) payment portal to pay for their goods.	Customers can 'purchase' items using a payment portal when they click checkout in their basket. We used Stripe to implement this. More details about this under the Additional Requirements section of the report.	COMPLETED
C2	View their order history	Once a customer purchases any product, that purchase will be made an order and they will be able to review their order history from the Orders page which is accessible through the 'Orders' button in the nav bar.	COMPLETED

Admin Requirements

Req #	Title	Comment	Status
O0	Login via the same interface as (G7)	Admin can log in using the same login page as any other users through the log in button in the nav bar.	COMPLETED
O1	Add products to the shop	Once logged in as Admin, the nav bar will contain an additional button called 'Add Product'. This button will redirect to a page that contains a form with the necessary input fields to create a new product card for the shop. The admin has to fill in the form appropriately. The form can validate for invalid inputs. If the admin provides any invalid input, the form will not POST and an error will show to let the admin know what needs to be changed.	COMPLETED
O2	Hide existing products (that are no longer available)	Admin can hide existing products through the edit product page. This page can be accessed through the 'Edit' button in the product cards. This option can be found at the bottom of the form in the edit product page. The product will be hidden from guests and registered users; however, it still can be seen by the admin in case the admin would like to unhide the product.	COMPLETED
O3	View all orders	Admin can view all orders by clicking on the 'Orders' button located in the nav bar. This redirects to the Orders page which contains all the orders made by customers. There will be four tables, 'New orders', 'Shipped Orders', 'Cancelled	COMPLETED

		Orders' and 'Delivered Orders'. These tables will contain relevant information based on the statuses of all the orders.	
O4	Change the state of orders	In the Orders page, a button called 'View' can be clicked which redirects Admin to a receipt of the order made. In the order receipt page, the admin can change the status of the order through a drop-down list.	COMPLETED
O5	Edit their product details	Admin can edit product details through the button 'Edit' located in the product cards. This button redirects to the Edit page which contains a form. This form is very similar to how the add product page is laid out and will override the product's data in the database.	COMPLETED

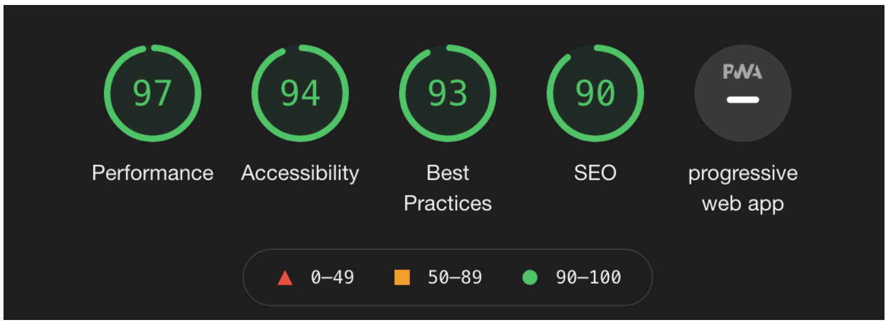
Technologies Used

Lombok
Lombok is used in multiple services and controllers to simplify code since we can save the @Autowired notation so it makes our code more concise and readable. Besides, @Getter and @Setter notations are also used in some entities to generate getters and setters automatically. Overall, Lombok improved our code quality and also saved us quite a lot of boilerplate code.

Bulma
This is the pure CSS library we used instead of Bootstrap since it natively supports responsive design using Flexbox and it also has a more modern and stylish look. Besides, the library itself is lightweight and it helps reduce our CSS code tremendously. Our team definitely found that by using a CSS library, it helps to keep our design consistent and saves us a lot of design work.

Javascript ES6
A large portion of our front-end JavaScript code is written in ES6, since it's easier to write asynchronous code using the fetch API instead of AJAX. We used async and await since it makes API calls easy to read and manage. We also used const and let instead of var because it helps to scope variables within anonymous functions and it does not pollute the global namespace. We also used payment.js to handle client-side Stripe payment code using ES6 features.

jQuery
We only used jQuery in a handful of pages (contact and GDPR page) since it introduces extra overhead, so we try to use vanilla JS for most of the pages. Keeping minimal front-end dependencies improves page loading times and makes our site feel snappy.



Spring Security

Since basic authentication is one of the requirements for this project, we decided to use Spring Security to secure our routes and hash passwords correctly. We added a security configuration file to limit /user/* routes to customers and /admin/* routes to shop owners and we also added our own custom registration page and login page. Furthermore, passwords are hashed using the Bcrypt utility which is a very impressive hashing utility. Besides, in some controllers @AuthenticationPrincipal is used to determine if the user's login status and also their role. We also use sec:authorize in thymeleaf templates to display content depending on the user's role.
Lastly, we defined a custom authentication success handler to handle post login actions such as saving the guest cart if the customer is a first-time registered user. Overall, it helps us to secure the site nicely and it's also easy to manage once we are familiar with the API itself.

Hibernate Validator

We used this for form validation at the entity level (registration form, adding/editing products, etc.), it helps easily validate a field by just using annotations. We also defined default messages for validation errors so that they can be inserted into BindingResult during validation. Another cool part is custom annotations, @UniqueEmail is a custom annotation we defined to check for duplicate emails during registration.

Stripe

Stripe is used as the payment processor for our website to handle all transactions since its API is very easy to use and we don't need to store any credit card details on our site. Besides, Stripe provides a test API key for us to test our transactions so we can view all transactions in the Stripe dashboard. During our testing, one issue we noticed is that duplicate transactions might occur if a customer hits pay multiple times or refreshes the page when the transaction is in progress. To fix this issue, idempotent keys are used to ensure that no duplicate transactions will happen.

Cloudinary

We used Cloudinary as our image CDN so that we don't need to store images in the DB and we can easily do image transformations just by using the URL link. Cloudinary also caches images during accesses so it helps reduce bandwidth consumption on our site which leads to faster load times. We also used the Cloudinary Java SDK to handle image uploads when a user updates or adds a new product image. Optimisation settings are also used on the Cloudinary website so that new images get compressed and optimised.

Gitlab CI/CD

We utilised Gitlab CI/CD by creating a .gitlab-ci.yml in our repository. For every push we make it will automatically build the project in a Docker container and deploy it to our Heroku container. Initially, we planned to build two different builds, one for staging and one for production but it took too much time since Github runners are shared and not always available. So, we ended up just settling with the production build. We also defined our private Heroku key in the environment variables section of the Gitlab Repository. Overall, the process works very well and we get to see our changes in production after around 5 minutes.

Heroku

Heroku is used as our deployment service since it offers a generous free tier which will work for small scale projects like ours. Deployment to Heroku is essentially painless, as we just use the Gitlab pipeline to auto deploy to Heroku after every push. The maven profile “prod” is used as our production profile as it uses application-prod.properties which contains configurations for Postgres. In our production build, all data are persisted in a Postgres database and will remain the same even if the server restarts, unlike the in memory H2 database. One challenge we encountered with the free plan is that it sleeps if the application is idle for more than 30 minutes, hence we created a Cron job which periodically pings the application so that it stays alive all the time.

Postgres

We used PostgreSQL as our production database since it’s natively supported by Heroku and we get 10k rows for free, which is more than enough for our use case. We have to do some modifications to our migration script so that it supports the Postgresql dialect. But other than adding some configurations in application-prod.properties, it’s pretty painless to change our database provider from H2 to Postgres since Hibernate helps us handle a lot of the heavy lifting.

Flyway DB Migration

To load initial data such as user accounts and default list of products we used a Flyway Migration script to insert all of the data. We originally used data.sql for this but it’s not ideal because it inserts all the data again on each server reboot (only good for h2). The V2_fill-tables.sql script only runs once when the DB is empty so it’s ideal for our use case.

Docker & Docker Compose

We also included Dockerfile and docker-compose.yml, so that the site can be deployed on any other environment easily. Deployment instructions are included in the readme.

Additional Requirements

Test Accounts

- email: user@gmail.com password: user@gmail.com for customer account
- email: admin@gmail.com password: admin@gmail.com for owner/admin account

Categories

- Added some additional categories such as electronics, food and featured
- Achieved by adding an extra table for categories
- Users can also browse all products which belong to the same category by visiting the corresponding page

Pagination

- We decided to implement pagination for search, categories and the products so that each page doesn’t get too crowded
- Implement using the Page entity in Spring and we are able to limit the number of products which appears on each page

Delete products

- With the inclusion of an “Add Products” feature, we decided that it only makes sense to also have a “**Remove Products**” feature for the admin. This feature is visible when an admin has logged in to their account, and a Remove button is present in every product card just below the edit button.
- Remove button works for both existing and newly added products. Onclick it calls for the removeProduct function which generates a response in relation to the product ID, and then removes the product from the database.

Contact page

- When a customer is signed in, they have access to the contact page. Here they can submit a question to the admin. In a subject dropdown they can choose from a generic option or regarding an order they have placed. Clicking send message assigns the message to the user and saves it to the database. The message has an indicator ‘NEW’.
- When an admin is logged in, they can see all users’ messages. They can reply to messages of type ‘NEW’. The customer’s message is modified to include the admins response. The message is assigned type ‘ANSWERED’. Admin can only reply to new messages, as the reply field for answered messages is hidden after a response is submitted.

GDPR compliance

- Through the contact page a user can access the GDPR compliance page. Here they can download all personal data stored in the database. By clicking on the ‘Get My Data’ button a PDF containing all of the user’s data is generated and downloaded.
- Implement using jQuery and the jsPDF library.

Stripe Payment

- In the cart page, there’s a checkout button to redirect to the checkout page to complete their payment using the Stripe API. We are using Stripe to handle processing and it will handle the transactions for us so we don’t need to store any sensitive credit card information.
- The backend server also checks if the current shopping cart includes invalid items (hidden or out of stock) and removes them from the cart, preventing the order to go through.
- Besides, “idempotent keys” are also used to prevent duplicate charges for the same order.
- We also used @Transaction annotations to tie related operations together so that if one part of the transaction fails, the whole transaction will not go through.

Authentication

- We are using Spring Security to handle authentication and passwords are securely hashed in the database. There are two roles in our system which are Admin and User and all newly registered users will have the role User.
- As mentioned in the previous section, /user/** and /admin/** are protected and only allowed for privileged users
- Passwords are hashed securely using Bcrypt

Image upload service

- Cloudinary is used as a cloud image storage (CDN) for all of our product images so that we don’t need to store images locally in the database.

- Uploading is done through the API and we retrieve the uploaded URLI and save it in our database.

Instant Search

- Instant search instantly populates the search results as the user types their search query
- This is implemented by periodically calling the instant-search endpoint to continuously fetch search results and insert them into the DOM using JavaScript.
- We also introduced an interval before the next search result gets populated to not overload our backend with too many requests.

Deployment

- We deployed our application to Heroku and it's publicly accessible by anyone.
- This makes it easy to show off our project on Github/Gitlab.

Reflections

Chee Guan Tee

In this project, I learnt how to design a Spring Boot application from scratch as I started to work on the project about a week before others so I did quite a lot of research and planned out the whole project using user stories (issues) in Gitlab. I completed a few of the core features first so that we are on track with the project and we slowly introduced additional features when all of the core features are done. Besides, I also assigned user tasks and guided my fellow teammates so that they can work on their respective parts while overseeing the whole project. Some interesting parts I implemented & learnt in this project are Spring Security, the Stripe API and also Cloudinary. I feel that good communication is extremely important in a group project and I am relieved that for our case it worked out really well and everyone contributed to the project significantly and put in tremendous effort into it.

I also used the Gitlab CI/CD feature to deploy the application to Heroku so that it can be easily viewed and tested. I think if I started over, I would probably integrate npm into our project. Other than that, I feel that the whole project is well thought out and well structured (Services, Controllers, etc).

Choon Wei Tong

After the completion of this project, I have learnt a lot about web development. I learnt how to effectively use Controllers, Services and Repositories. With the use of Thymeleaf and JavaScript, I have learnt a lot about tying the frontend and backend of the website together. Mapping the entities correctly for the database is a very important task. The challenges that I have encountered were mostly to do with the backend, mostly focusing on handling requests appropriately. On top of that, delegating tasks amongst group members was very important for us to be able to successfully complete all of the requirements and also adding extra implementations to our website. I also learnt a lot regarding the usage of fragments, which proves to be a very convenient feature when building a website of a larger scale. If I started over, I would research more about Thymeleaf and Spring features which would greatly improve productivity.

Shijie Liu

Throughout this project I learnt how to post objects containing data from the server to the corresponding database using Thymeleaf, implementing controller methods and services, detailed HTML embedding and correlating JavaScript to handle certain POST methods. This project gave me an insight into the teamwork structure needed for building a proper website, from initial design, backend planning/structure/services and frontend UI display.

The challenges I encountered were in relation to balancing all existing assignments and other work during this difficult time, where our group suffers 3 different time zones. Project wise it was challenging to figure out the correct method to post the ID of a product and allow that information to be passed and executed by the query selector. Things I would do differently would be learning more about Spring, trying out examples and Spring boot.

Simonas Ramonas
During the development of this project, I learned how to seamlessly integrate backend and frontend technologies. I learned how to get and post data from the web page to the database, how to display the data from backend on the webpage using Thymeleaf and creating entities, and repositories. I had the chance to deal with controllers more in depth and create more complicated controller methods. I gained experience in writing more intricate functions in JavaScript. I also had a go using a CSS framework called Bulma and writing elaborate HTML. On top of that, I furthered my understanding of the importance of teamwork and communication for a project of this scale.

The challenges I encountered during this project include figuring out which approach is best for the functionality I want, not enough experience with backend web development as well as lack of motivation and mounting deadlines.