Vegan Pizzeria

Jane Mansell & Victoria Daramy-Williams

**Project Statement:**

This project is a pizza ordering program for a vegan pizzeria that enables customers to place pizza orders and track the status of their orders. Additionally, employees can access order details and manage order statuses. The programme also incorporates an API to showcase cute animal images, enhancing the overall user experience.

Objectives:

1. Provide a user-friendly interface for customers to browse pizza options, customise their orders, and place orders seamlessly.
2. Implement a tracking system that allows customers to monitor the status of their orders in real-time, providing updates on order preparation, baking, and delivery.
3. Develop an employee portal that enables authorised staff members to access order details and manage order statuses.
4. Integrate an API to fetch and display adorable animal images, creating an engaging and enjoyable experience for users.

The project will be developed using the following technologies: Java 17, Maven, Jar, Spring Boot (v 3.1.0), Thymeleaf (v 3.0.4), mySQL, JDBC Template, HTML, CSS, JS, Git & GitHub.

**Contents**

[Marking Grid 3](#_Toc137030958)

[Collaboration 5](#_Toc137030959)

[Initial Planning 6](#_Toc137030960)

[Work Sprints 8](#_Toc137030961)

[Sprint 1 8](#_Toc137030962)

[Sprint 2 9](#_Toc137030963)

[Sprint 3 10](#_Toc137030964)

[Project Structure Diagrams 11](#_Toc137030965)

[Initial ERD 11](#_Toc137030966)

[Final ERD 12](#_Toc137030967)

[Class Diagram 13](#_Toc137030968)

[Data Flow Diagram 13](#_Toc137030969)

[Wireframes 14](#_Toc137030970)

[Prior art 19](#_Toc137030971)

[Challenges 20](#_Toc137030972)

[Animal Fact API 20](#_Toc137030973)

[Feeding in Parameters 21](#_Toc137030974)

[Alternative Ordering Methods 22](#_Toc137030975)

[Different Formats Across Platforms 22](#_Toc137030976)

[Exception Handling & Input Validation 22](#_Toc137030977)

[Future Work (Beyond the scope of our current project) 23](#_Toc137030978)

[References 24](#_Toc137030979)

# Marking Grid

| **Criteria** | **Location** |
| --- | --- |
| **Documentation**   * The project submission includes flowcharts and/or UML diagrams to describe the code. * The project submission includes wireframes for the GUI. * The project submission includes an ERD to describe the database. | This document. |
| **The code is well-organized, using an MVC approach.**   * The code is organized in an object-oriented style. * The code uses proper naming conventions consistently for all variables. * Each layer communicates with other layers appropriately. |  |
| **The application includes functional unit testing for each part of the application.**   * The application includes testing for all CRUD operations. * Tests are implemented using JUnit 5 or another Java testing suite. |  |
| **The application demonstrates the appropriate use of error handling.**   * The application includes custom error messages and implements them where appropriate. * Error handlers are in the appropriate packages. |  |
| **The application uses Spring tools to build and organize the code.**   * The code includes Spring DI and appropriate @Annotations. |  |
| **The application includes a database.**   * The database uses appropriate tables and fields. * The database is normalized to at least 2NF. * The submitted files include a schema .sql file to create the database and a data .sql file to add data to the database. |  |
| **The application uses JDBCTemplate or JPA to perform CRUD operations on data using appropriate REST API endpoints.**   * The application includes endpoints to Create, Read all, Read by Id, Update by Id, and Delete by Id. * The application uses server data validation before writing data to the database. * The application includes other appropriate endpoints required by the project. |  |
| **The application includes a frontend GUI interface to interact with the end user.**   * The frontend is written in an appropriate language (JavaScript, jQuery, React, or Angular). * The frontend implements data and form validation. |  |
| **The application uses a version control system for code.**   * The submitted files include appropriate Git (or other VCS) controls. | Git used for version control. See process on page 5. |
| **The project was presented to the class.**   * The project presentation was 10-15 minutes long and provided an appropriate high-level overview of the project. * The project presentation was completed in a professional manner. * The presenter(s) responded appropriately to questions at the end of the presentation. | Completed on 02/06/2023. |

# Collaboration

We used Git and GitHub to collaborate as it has version control and enables you to check conflicts before adding code to the master project. The procedure followed is documented below.

|  |  |  |
| --- | --- | --- |
| **#** | Step | Command(s) |
| **1** | View existing branches (current branch indicated with \*): | git branch |
| **2** | Create new branch | git branch [branch\_name] |
| **3** | Move to new branch | git checkout [branch\_name] |
| **4** | Make changes in IDE |  |
| **5** | Check which files have unsaved changes | git status |
| **6** | Add files to landing stage | git add --all |
| **7** | Commit files | git commit -m “[descriptive message]” |
| **8** | Check changes made to master and merge | git checkout master  git pull  git checkout [branch\_name]  git merge master |
| **9\*** | If changes made to master, pull to branch | git status  git add –all  git commit -m “[descriptive message]” |
| **10** | Push changes to online repository (GitHub) | git push origin [branch\_name] |
| **11** | Check push on GitHub | Navigate to GitHub repository |
| **12** | Trigger pull request to master | Click pop-up notification ‘compare and pull request’ button |
| **13** | Induce pull request | Click ‘create pull request’ button |
| **14\*** | Confirm pull request if no conflicts. | Click ‘merge pull request’ button  **If conflicts, Zoom call to agree course of action.** |
| **15\*** | Delete branch if safe to do so | Click ‘delete branch’ button |
| **16** | Return to Git Bash and pull changes to local master | git checkout master  git pull |

# Initial Planning

Customer

|  |  |  |
| --- | --- | --- |
| Use Cases | Task Owner | Notes |
| Register as a customer or sign in | Vic | * Do we want an admin user to create employee users? |
| Place an order | Vic | * View menu (name, desc, price, pic?) * Name, mobile, address? (GPS - check distance? delivery flat rate?) * <https://developer.paypal.com/docs/api/payments/v1/> |
| Check status of order | Jane | * Automatically timed? When the initial order is placed the system should record the time. The other statuses should be set by the employees as they actually happen. |

Baker/Chef/Delivery person

|  |  |  |
| --- | --- | --- |
| Use Cases | Task Owner | Notes |
| List of orders with chosen status with function to change status. Possible statuses: ‘cooking’,  ‘ready for pickup’, ‘picked up’ or ‘delivered’ | Jane | This could be just one webpage with buttons on the top so other orders are hidden?   * Once set, have an automatic timer? (potentially different for each pizza type)   Dropdown for status? |

|  |  |  |
| --- | --- | --- |
| Databases required | Task Owner | Notes |
| Orders |  |  |
| Customers |  | GDPR - don’t ask for more info than we need. |
| Employees |  | It would be good to have a superclass for users that we extend for employees and customers so we probably want to have some common fields between customers and employees? |
| Pizzas | Vic | One of us needs to create the table but then we should probably both add to the table. |
| Animal Stories |  | One of us needs to create the table but then we should probably both add to the table. Might not need to be a table. Maybe use an API instead? <https://apislist.com/api/3/cat-facts>  <https://dukengn.github.io/Dog-facts-API/?ref=apislist.com> |

|  |  |
| --- | --- |
| Other tasks | Task Owner |
| Presentation | Vic and Jane |
| Animal Stories and puns | Vic and Jane |
| Animal pictures | Vic and Jane |
| Random generation of stories |  |
| Rescue stories |  |
| Environmental impact |  |
| MythBusters |  |

Started off with a separate baker and delivery person. Combined after Eugene’s suggestion.

# Work Sprints

|  |  |  |
| --- | --- | --- |
| Sprint 1 | | |
|  | Vic | Jane |
| **Aim for the sprint** | **Customer places an order** | **Customer and Employees can check status of order** |
| MySQL Database | Create database  Create order, pizza and customer tables. | Create employee and orderlines tables.  Put in some sample data. |
| Create repo and packages.  Add this document and SQL scripts to repo. | Complete together on Zoom call on 28/5/2023.  Zoom Meeting link:  <https://crick.zoom.us/j/2464479323?pwd=dFpmblpWQlJLa09aQTRpYU5Pcnd2QT09> | |
| RowMappers | Customers  OrderLines  Pizzas | Orders  Employees  Login |
| Entities | Customers  OrderLines  Pizzas | Orders  Employees  Login |
| DAO interfaces and implementation classes | Customers  OrderLines  Pizzas  CRUD plus  Retrieve customer by email address | Orders  Employees  Login  CRUD plus  Retrieve employee by email address |
| Controllers | Customers  OrderLines  Pizzas | Orders  Employees  Login |
| html files | orderPizza | checkOrderStatusCustomer  CheckOrderStatusEmployee |
|  |  | Add change status functionality for cook > next sprint |

|  |  |  |
| --- | --- | --- |
| Sprint 2 | | |
|  | Vic | Jane |
| **Aim for the sprint** | **New customer signup screen and improve visual appearance of customer facing pages.** | **Login for existing customer and employees** |
|  | Zoom Meeting link:  <https://crick.zoom.us/j/2464479323?pwd=dFpmblpWQlJLa09aQTRpYU5Pcnd2QT09> | |
|  | Home screen (index.html) | Login Screen |
|  | New customer signup screen | Link login screen to correct menu depending whether customer or employee. |
|  | Research, create and apply styling to the following pages:  signUp  Login  CustomerMenu  customerTrackOrder  PlaceOrder | Pass customerId parameter from login through to the mappings relating to customer. |
|  | Add signup functionality > next sprint | Add functionality for cook to change status of order |

|  |  |  |
| --- | --- | --- |
| Sprint 3 | | |
|  | Vic | Jane |
| **Aim for the sprint** | **Finish fixing any missing functionality and clean up code** | |
|  | Zoom Meeting link:  <https://crick.zoom.us/j/2464479323?pwd=dFpmblpWQlJLa09aQTRpYU5Pcnd2QT09> | |
|  | Add new customer signup functionality to web page | Add back button for cook when checking pizzas to coo |
|  | Add carousel for pizzas on order page | Remove temporary order date field and implement today’s date for checking status of orders for employee and customer |
|  | Refactor HTMLs so CSS code is not repeated | Implement a conditional so that customer receives nicer messages than just ‘Cooking’ |
|  | Add payment API > next sprint | Add API to pull in animal fact to customer tracking screen |
|  | Fix line cost max value in schema | Link orderliness and Order mappings so that an order is written to the database containing the lines of the orde |
|  | Put in exception for duplicate. | Remove customer status basket from customer tracking view |
|  | Fix testing so everything isn’t deleted when you run the test. | Back button on place order, login, sign up rather than sign out. Check any others. Only customer and employee menus say sign out. No back on signup successful. |
|  | Add sixth pizza to menu | Add sign out button on customer tracking screen |
|  | Change button hover background colour | Update doc until 4pm |
| 6/6/2023 | Update doc  10:30 – 1:00pm | Update doc  7:00 – 10:30  1:00 – 3:00pm |
|  | Meet 3pm 6/6 to wind up. | |

# 

# Project Structure Diagrams

## Initial ERD

A screenshot of a computer

Description automatically generated with low confidence

## Final ERD

A screenshot of a computer

Description automatically generated with low confidence

## A picture containing text, diagram, line, parallel Description automatically generatedClass Diagram

## Data Flow Diagram

A picture containing text, diagram, line, plan

Description automatically generated

|  |  |  |
| --- | --- | --- |
| Use Case | Description | User |
| 1 | New user registration | Customer |
| 2 | Placement of order | Customer |
| 3 | Tracking of order | Customer |
| 4 | Retrieval of order information | Employees |
| 5 | Management of order status | Employees |

## Wireframes

Landing page

A picture containing text, diagram, screenshot, line

Description automatically generated

Login page

A screen shot of a login form

Description automatically generated with medium confidence

Sign-up page

A screen shot of a login form

Description automatically generated with medium confidence

Menu page

A picture containing text, diagram, parallel, line

Description automatically generated

Homepage - customer

A picture containing text, screenshot, diagram, number

Description automatically generated

Homepage - manager

A picture containing text, diagram, line, parallel

Description automatically generated

Homepage – employees (cooks & delivery drivers)

A picture containing text, diagram, line, rectangle

Description automatically generated

Place an Order

A screenshot of a login form

Description automatically generated with low confidence

Track Orders – Delivery person

A picture containing text, diagram, screenshot, line

Description automatically generated

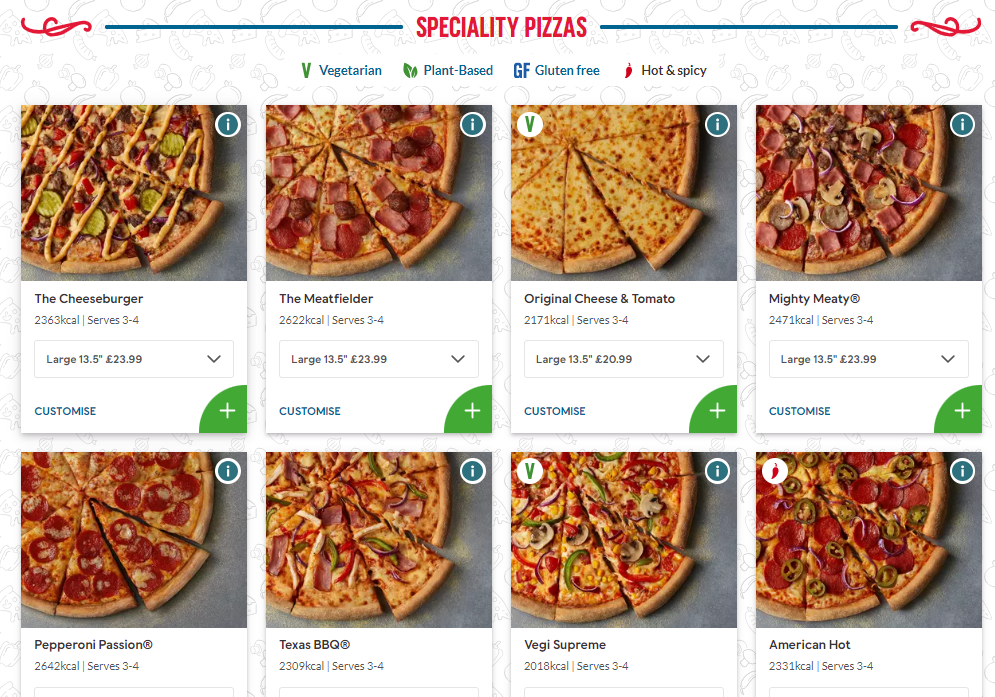
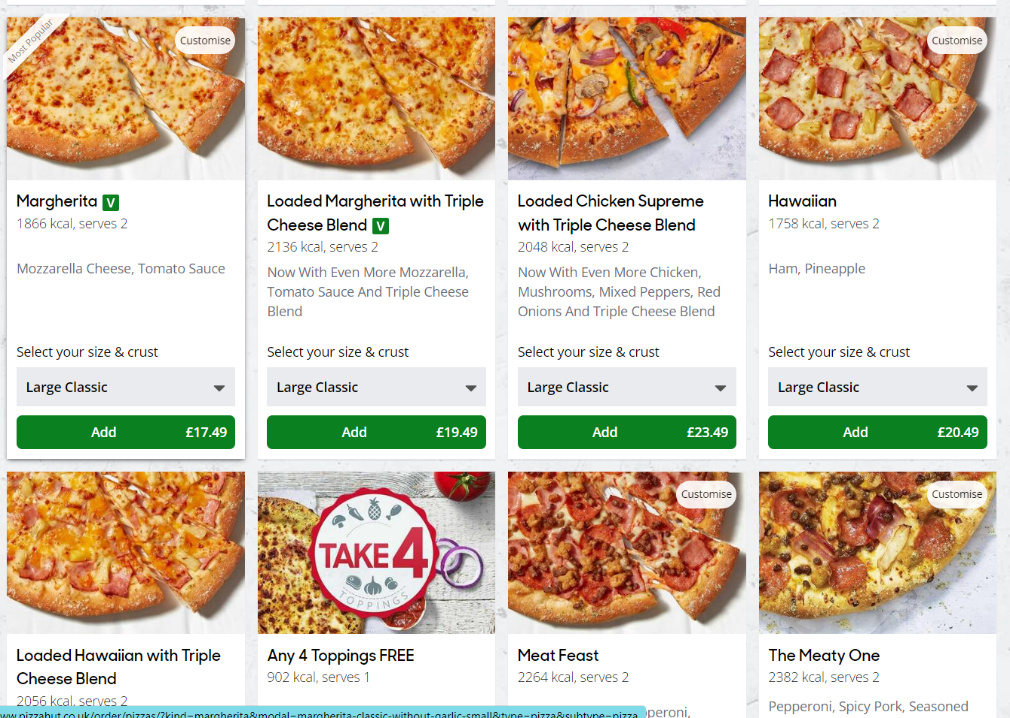
Track Orders – Cook

A picture containing text, diagram, number, line

Description automatically generated

# Prior art

Pizza ordering sites are hardly rare, so there are plenty of sites from which to take inspiration. The following screenshots were taken from the PizzaHut and Domino’s sites (links in References).



Obviously, these commercial sites have extended functionality (well beyond the scope of this project), so these were mainly taken as design inspiration. We also looked at pizza ordering applications publicly available on GitHub (links in References). These gave a good indication of what could be reasonably expected in a project of this scope.

# Challenges

## Animal Fact API

The original plan was for the customer tracking screen to update, every time the customer visited/refreshed it, with a fun animal fact.

We found several animal fact APIs (see references for links). However, on further investigation they were very disappointing. Some need to be paid for so they were excluded from the list, some were just facts, e.g., a cheetah has 4 legs. Finally, we found an API that had interesting facts. We used Postman to test out the endpoints and hence had a http method (GET) and a URI that worked. The response body was in JSON format.

*// Animal Fact from API  
 //Build get request* HttpRequest request = HttpRequest.*newBuilder*()  
 .uri(URI.*create*("https://random-d.uk/api/random"))  
 .method("GET", HttpRequest.BodyPublishers.*noBody*())  
 .build();  
*//Get response from endpoint*HttpResponse<String> response = HttpClient.*newHttpClient*().send(request, HttpResponse.BodyHandlers.*ofString*());

It took a long time to work out how to get the URL out of the JSON wrapper (see references). Even trimming and splitting the string was not successful as some of the facts have commas in them. Hence, we started a curated list of ids that we had checked did not have commas. This was quite time consuming and unfortunately bad actors had posted inappropriate facts to the site so for every interesting fact there were several unpleasant messages. Thus, we started the search for an API again. When searching for an API, several APIs containing animal photos rather than facts appeared in the search. We then decided to switch to displaying an animal photo instead. We found an API at <https://random-d.uk/api> that is free and doesn’t allow external entities to post to it that has pictures of ducks. WE found other APIs for specific animals but none with a mixture (see references). We considered using several and picking one at random but decided just using the ducks shows how it works. More animals could be added if this were a commercial product.

We were not happy with the string splitting and trimming method, so we investigated again how to get the information out of a JSON object in Java. We found the simplest way was to add a dependency for GSON to the pom.xml file.

<dependency>  
 <groupId>com.google.code.gson</groupId>  
 <artifactId>gson</artifactId>  
 <version>2.10.1</version>  
</dependency>

To use this we had to create a java class DuckPic that had fields for the fields in the JSON message and getters and setters. Then we could create an object out of the JSON string:

*//Turn response body into a json String*String responseString = response.body().toString();  
*//Use Google Gson to map the json string to a java object  
//This required an extra dependency in the pom file*Gson gson = new Gson();  
DuckPic duckpic = gson.fromJson(responseString, DuckPic.class);

The URL can be extracted using **duckpic.getUrl()**. However this did not work when added as a model attribute. The Model needed the java object to work:

*//Add duckpic object to model. Use Thymeleaf in html to get URL.* model.addAttribute("duckpic", duckpic);

We applied the logic of how the href method works in ThymeLeaf to src and found it pulled through the URL for the picture:

<div>  
 <img class = animal-img src = "#" th:src = "${duckpic.url}" alt = "Picture of animal">  
 </div>  
 <div class = "tiny-writing" th:text="${duckpic.message}">Duckpic Message></div>

The duckpic.message above is attributing the site where the images are from so that also displays on the page albeit in a smaller font.

## Feeding in Parameters

The biggest challengein the project was feeding parameters between the mappings. When a customer logs in the system has to hold on to their ID throughout their visit so that orders are placed against them, and the tracking screen shows information about their order only. This happens in different ways.

When the customer logs in and is directed to the customer menu the following code is used:

return "redirect:/customerMenu/" + customerId;

The annotation for the mapping and method signature are then:

@GetMapping("customerMenu/{customerID}")  
public String displayCustomerMenu(@PathVariable("customerID") String customerId, Model model ) {

The URL that appears in the search bar appears as “…/customerMenu/1” where 1 is the ID number of the customer that has logged in.

When the customer clicks on Place Order a new order is created and the order number created and customer ID are then passed into the placeOrder page using this line of code:

return "redirect:/placeOrder?id=" + customerId + "&o=" + order.getId();

This appears on the search bar URL as “../placeOrder?id=1&o=9” This is customer with Id of 1 and order number 9. To pull these in to the mapping the annotation and method signature are as follows:

@GetMapping("placeOrder")  
public String startOrder(@RequestParam(name = "o") String oId, Model model) {

The customer ID is not extracted in this case as the order can be retrieved using the order ID and the customer ID then retrieved from the order. However it could have been retrieved by adding @RequestParam(name = “id”) String id to the method signature.

The above two methods work for mappings of the form @GetMapping but not for @PostMapping.

As the @PostMappings will not accept parameters passed through the URL they had to be passed by using inputs of type hidden in the html pages themselves. This is the html code for the Submit Order button that passes the customer and order IDs to the @PostMapping:

<input type = "hidden" name = "id" id = "id" th:value = "${order.customerId}"/>  
<input type = "hidden" name = "oId" id = "oId" th:value = "${order.id}"/>

This requires using a HttpServletRequest in the method signature and a request.getParameter in the body of the code:

@PostMapping("/submitOrder")  
public String submitOrder(HttpServletRequest request) {  
 *// Retrieve form parameters* int customerId = Integer.*parseInt*(request.getParameter("submitCustId"));  
 int orderId = Integer.*parseInt*(request.getParameter("submitOrdId"));

## Alternative Ordering Methods

To display OrderLines in real-time for the customer as it was added to the order, it was necessary to first create and add an Order to the database, and subsequently the OrderLines, so that they could take the orderId parameter from the Order. As previously discussed, passing parameters across pages was a major bottleneck, so alternative avenues for ordering were explored.

Taking inspiration from prior art, we tried an alternative implementation which had users viewing all pizzas on the menu (similar setup to the menu page), with quantity toggles at the bottom. Users could amend the quantities for the pizzas wanted and then submit the whole order at once. This way, the Order object, and then the OrderLines would only be made and added to the database when the user submits the order, rather than as the user goes along. Having the setup this way also eliminates the need to update the orderLines in real-time, as the user can always see the components of their order.

However, passing the quantities back to the controller proved to be difficult. The pizzas were on dynamically created cards, which meant that passing the parameters back to the controller was not straightforward; each quantity parameter had to have a pizza-specific parameter name so that it could be correctly identified in the controller. After extensive research, it seems that some complicated JavaScript would be required to implement this. In the end, the original implementation was kept.

## Different Formats Across Platforms

One of the biggest challenges was the different formats used across SQL, Java and ThymeLeaf.

SQL saves its dates and times in its own DATE and TIME format. Java uses a LocalDateTime API. To convert from SQL to Java the data had to have a .toLocalDate() wrapper:

order.setOrderDate((rs.getDate("orderDate")).toLocalDate());

To convert from Java LocalDate to SQL DATE the data had to have a .toString() wrapper:

order.getOrderDate().toString()

ThymeLeaf does not automatically work with Java’s LocalDateTime either. A dependency had to be added to the pom.xml file to make this work:

<dependency>  
 <groupId>org.thymeleaf.extras</groupId>  
 <artifactId>thymeleaf-extras-java8time</artifactId>  
 <version>3.0.4.RELEASE</version>  
</dependency>

SQL stores its values as DECIMAL whereas in Java we used BigDecimal as this is an ecommerce site and hence the costs are currency. Hence values taken from SQL had to be converted to BigDecimal but values posted to the database did not require a wrapper.

## Exception Handling & Input Validation

The server-side exception handling primarily focused on handling SQL queries within the DAO implementations. These queries were enclosed within try-catch blocks to catch the DataAccessException exception type.

In order to enhance user experience and manage input exceptions, the program's website utilizes various input validation tools provided by HTML. These tools include:

1. Adding the <required> attribute within the <input> tag where necessary to enforce user input.
2. Setting a minimum value (0) for numerical inputs.
3. Specifying a maximum character length for string inputs to ensure database compatibility.

For more complex input validation, such as field matching (e.g., password and confirmPassword), JavaScript was employed. However, the implementation of JavaScript/AJAX combination for database communication, such as checking for null or duplicate emails during login and signup, was not completed by the deadline. As a result, these errors were handled on the server-side within the UserLoginController using 'if' blocks to redirect the page when such errors occurred. Although this approach may not provide an optimal user experience, future plans include updating this method to improve the handling of input validation errors.

## Security

Spring security was added to the project. This was done by adding a dependency to the project:

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
</dependency>

A Java class was created for CustomUserDetails that implements UserDetails to create an entity to hold the details of the user and the areas they have authority over. The constructor contains the entity UserLogin that is used as a DTO between the controllers and the DAO. A customerUserDetailsService Java class is used to pull the userLogin information into the UserDetails object.

BCryptPasswordEncoder was used to encode the passwords (and decode them).

Different users were given access to different parts of the website using the following code:

auth.requestMatchers("/index", "/menu", "/login", "/registerSuccess").permitAll();  
auth.requestMatchers("/customerMenu", "/customerTrackOrder", "/placeOrder").hasAnyRole("Customer");  
auth.requestMatchers("/cookTrackOrder", "/employeeMenu", "/orderDetail").hasAnyRole("Employee");  
auth.anyRequest().authenticated();

A LoginSuccessHandler class was created to divert the user to a different webpage on successful login depending whether they are a Customer or Employee.

However the security setup requires the user to login immediately on accessing localhost:8080. Hence the initial screen of choosing whether to look at the menu or login cannot be accessed until after logging in. Hence the security has been saved to a separate branch and has been removed from the submitted project.

# Future Work (Beyond the scope of our current project)

1. Screen responsiveness (e.g. for mobile screen size)
2. Customisation of pizzas (size, toppings etc.)
3. Google Maps API for location-specific functions
   * Determine delivery availability and cost
   * Delivery driver shortest route
4. Order history (previous orders)
5. Admin user functionality:
   * To create new employee users
   * Delete orders
   * Ban customers
   * Set pizza quantity limits
6. Delivery user to have screen of pizzas to pickup where they can change status to ‘picked up’ or ‘delivered’. This screen to include addresses of customers and order them with shortest route.
7. Forgot password link
8. Editing pizza quantities of already-added pizzas, rather than creating new lines
9. Improved exception handling:
   * Login: update to have “email/password incorrect” rather than refreshing page
   * Sign up: update to have “email already registered” rather than refreshing page
   * Customised exceptions for application
10. Add image references on website or create own images
11. H2 database
12. Security – research how to circumvent requirement to login immediately upon accessing the site

# References

**Collaborating on GitHub:**

<https://www.youtube.com/watch?v=_wQdY_5Tb5Q>

**Using ThymeLeaf, SQL and Java date formats:**

<https://www.baeldung.com/dates-in-thymeleaf>

<https://medium.com/@pdouvitsas/global-localdate-format-in-spring-boot-and-thymeleaf-29ff83b8f4c8>

**Passing parameters to other mappings:**

<https://dzone.com/articles/spring-boot-passing-parameters>

**CSS:**

<https://css-tricks.com/emoji-as-a-favicon/>

<https://www.w3schools.com/w3css/w3css_web_tmp_pizza.asp>

https://dev.to/dcodeyt/creating-beautiful-html-tables-with-css-428l

**Animal APIs:**

<https://www.jsonapi.co/public-api/Cat%20Facts>

<https://alexwohlbruck.github.io/cat-facts/docs/endpoints/facts.html>

<https://rapidapi.com/brianiswu/api/cat-facts>

<https://publicapis.dev/category/animals>

<https://cataas.com/#/>

<https://dog.ceo/dog-api/>

<https://github.com/quokka-api>

<https://randombig.cat/roar.json>

<https://random-d.uk/api>

<https://randomfox.ca/floof/>

<https://www.w3docs.com/snippets/java/get-a-json-object-from-a-http-response.html>

<https://howtodoinjava.com/gson/gson/>

**Bootstrap cards and carousels:**

<https://getbootstrap.com/docs/4.0/components/card/>

<https://getbootstrap.com/docs/4.0/components/carousel/>

**JavaScript / AJAX exception handling:**

* Password match: <https://codepen.io/diegoleme/pen/qBpyvr>
* Email exists: <https://stackoverflow.com/questions/39644062/check-if-email-exist-using-ajax>

**Retrieving data from dynamic inputs:**

<https://stackoverflow.com/questions/42359668/how-to-dynamically-change-html-input-name>

<https://stackoverflow.com/questions/56112162/how-to-dynamically-set-html-input-names-with-thymeleaf>

<https://stackoverflow.com/questions/70918048/how-can-i-get-data-from-dynamically-created-input-field-using-vanilla-javascript>

**Security:**

<https://howtodoinjava.com/spring-security/login-form-example/>

<https://howtodoinjava.com/spring-security/jdbc-database-form-login/>

<https://www.youtube.com/watch?v=tDZPdovCH4I>

<https://bcrypt-generator.com/>

<https://docs.spring.io/spring-security/site/docs/5.7.0-M2/api/org/springframework/security/config/annotation/web/configuration/WebSecurityConfigurerAdapter.html>

<https://docs.spring.io/spring-security/reference/servlet/authentication/passwords/form.html>

<https://www.baeldung.com/spring-redirect-after-login>

**Prior art:**

* Pizza Hut ordering page: <https://www.pizzahut.co.uk/order/pizzas/>
* Domino’s ordering page: <https://www.dominos.co.uk/store/28043/london-fulham/menu>
* GitHub repos: <https://github.com/topics/pizza-order?l=java>
* Other projects: <https://www.freeprojectz.com/project-source-code-database-download/online-pizza-ordering-system-project>

**Image references:**

* Background image: <https://londoncitypizza.ca/why-should-you-search-vegan-pizza-near-me.html>
* Pizza menu images:
  + Plain: <https://img.delicious.com.au/qRrzAHSr/del/2019/03/marinara-pizza-102752-2.jpg>
  + BBQ Jackfruit: <https://pizzatoday.com/wp-content/uploads/2018/02/jackfruitbbqpizza.jpg>
  + Mushroom: <http://bitly.ws/GzWk> (short URL generated using <http://bitly.ws/>)
  + Roasted Veg: <https://diabetes-resources-production.s3.eu-west-1.amazonaws.com/resources-s3/migration/recipes/Roasted-Vegetable-Pizza.jpg>
  + Olive & Sundried Tomato: <https://realfood.tesco.com/media/images/RFO-Vegan-Pizza-1400X919-05ee14fe-5367-4e77-965a-b5b28dc32709-0-1400x919.jpg>
  + Vegan Pepperoni: <https://veganuary.com/wp-content/uploads/2022/06/vegan-pepperoni-pizza.jpg>