**Vegan Pizzeria**

**A statement about what the project is here**

Contents

[Marking Grid 2](#_Toc136936412)

[Script for Demo 2](#_Toc136936413)

[Collaboration 2](#_Toc136936414)

[Initial Planning 4](#_Toc136936415)

[Work Sprints 6](#_Toc136936416)

[Sprint 1 6](#_Toc136936417)

[Sprint 2 6](#_Toc136936418)

[Sprint 3 7](#_Toc136936419)

[Project Structure Diagrams 8](#_Toc136936420)

[Initial ERD 9](#_Toc136936421)

[Final ERD 10](#_Toc136936422)

[Class Diagram 11](#_Toc136936423)

[Data Flow Diagram 11](#_Toc136936424)

[Wireframes 12](#_Toc136936425)

[Challenges 16](#_Toc136936426)

[Animal Fact API 16](#_Toc136936427)

[Feeding in Parameters 18](#_Toc136936428)

[Different Formats Across Platforms 18](#_Toc136936429)

[Future Work (Beyond the scope of our current project) 19](#_Toc136936430)

[References 20](#_Toc136936431)

## Marking Grid

# Project Rubric

| **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. |  |
| **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 ??. |
| **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 Friday 02/06/2023. |

## Script for Demo

1. Run schema.sql and data.sql in my workbench and change password in application.properties to your password. (The SQL files are under src/resources in project.)
2. Run project in IDE.
3. Have 2 windows open. Left for customer, right for employee.
4. Login as customer: judd@gmail.com password: P0okl
5. Order 2 plain pizzas and 3 BBQ. Submit order.
6. System will take you to order tracking where you will see order number, status: ordered and a cute duck pic.
7. Switch to employee and login: leo@pizzata.com password: P0okl
8. Click Track Order (this screen is here ready for other options to be added, eg delivery)
9. Change status of cooking pizzas to pick up.
10. Look at detail of order you placed, press back button, and change to cooking
11. Switch to customer
12. Refresh to check status. Notice duck picture changes each time you refresh.
13. You can return to employee and change status to pickup and then refresh changes in customer if you wish to.

Sign out and return to sign in screen. Choose lin

## 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. This is the procedure we followed:

**GitHub Process when Sharing a Project**

1. You can see existing branches using the following (the branch you are on has a \*):

git branch

1. Create new branch:

git branch branch\_name

1. Move to new branch:

git checkout branch\_name

1. Make changes in IDE.
2. Check which files have unsaved changes:

git status

1. Add files to landing stage:

git add –all

1. Commit files to be saved:

git commit – “commit message…”

1. Check no changes have been made to project by other people since checking out own branch. Pull these changes and merge into new branch:

git checkout master

git pull

git checkout branch\_name

git merge master

1. If pulling from master there were changes:

git status

git add –all

git commit -m “Commit message…”

1. Push changes to GitHub:

git push origin branch\_name

1. Go into GitHub where you will see a message about the branch.
2. Click on green button: ‘compare and pull request”.
3. Click on button: Create a pull request.
4. If no conflicts click on button: merge pull request.

If there is a conflict Zoom call to agree course of action.

1. Should be a confirmation message saying safe to delete branch. Click on button to delete branch.
2. Return to GIT BASH and pull changes through 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 | Notes |
| 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.

Max 6 web pages including landing page.

## 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 |

### 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 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> | | |  | |
| X | Add new customer signup functionality to web page. | Add back button for cook when checking pizzas to cook. | X | |
| X | 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. | X | |
| 2. | Refactor HTMLs so CSS code is not repeated? | Implement a conditional so that customer receives nicer messages than just ‘Cooking’. | X | |
| 1. | Add payment API | Add API to pull in animal fact to customer tracking screen. | X | |
| X | Fix line cost | Link orderliness and Order mappings so that an order is written to the database containing the lines of the order. | X | |
| 4 | Put in exception for duplicate. | Remove customer status basket from customer tracking view | 1 | |
| 5 | Get login by email and if there is none it will return null so the new login can be added. If it is not null message to new user that email address already exists. | 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. | 3 | |
| a | The menu screen would look nicer with a sixth pizza. | Add sign out button on customer tracking screen. | 2 | |
| 2 | Hover buttons on place order screen different. They go white on a white background at the moment. | Update doc until 4pm. |  | |
|  | Fix testing so everything isn’t deleted when you run the test. |  |  | |
| Tomorrow | Update doc  10:30 – 1:00pm | Update doc  7:00 – 10:30  1:00 – 3:00pm |  | |
|  | Meet 3pm 6/6 to wind up. | | |  | |

Other tasks we might want to do if time but to be added to future work if not:

Exception Handling

Security

H2 database

Fix testing as currently deletes order database

Check no duplicate emails.

Forgot password link.

If customer adds pizza of same type line quantity is edited rather than new line.

## 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

### Class Diagram

**A picture containing text, diagram, line, parallel

Description automatically generated**

### Data Flow Diagram

**A picture containing text, diagram, line, plan

Description automatically generated**

**Could do with description of use cases here**

### 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

Homepage - employee - chef

Homepage - employee - delivery driver

Place an order - what do we want our page to look like? Drop-down boxes? Or let them choose from the menu page? e.g. like Domino’s website. specify quantity, add to basket (adds to lineOrder). But then would need an extra page for the basket to display information, which could add extra work.

A screenshot of a pizza

Description automatically generated with medium confidence

A screenshot of a pizza

Description automatically generated with medium confidence

Alternative (Pizza Hut) - instead of specifying quantity, just add as many times as wanted

Background image from : <https://londoncitypizza.ca/why-should-you-search-vegan-pizza-near-me.html>



Pizza descriptions

1. Plain: A classic vegan delight with a crispy crust, flavourful tomato sauce, fresh herbs, and a touch of sea salt.
2. BBQ Jackfruit: Tangy BBQ jackfruit on a crispy crust, topped with colourful bell peppers, red onions, and vegan cheese.
3. Mushroom: Robust and earthy, this pizza features a blend of cremini, shiitake, and oyster mushrooms with vegan cheese and fresh thyme.
4. Roasted Veg: Bursting with flavours of roasted zucchini, bell peppers, eggplant, and cherry tomatoes, finished with herb-infused olive oil.
5. Olive and Sundried Tomato: A Mediterranean-inspired delight with Kalamata olives, sundried tomatoes, vegan feta cheese, and a drizzle of olive oil.

## 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.

I 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, eg , a cheetah has 4 legs. Finally I found an API that had interesting facts. I 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 me 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 I started a curated list of ids that I 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. Hence I started the search for an API again. When searching for an API, several APIs containing animal photos rather than facts appeared in the search. Hence I decided to switch to displaying a cute photo instead. I 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. I found other APIs for specific animals but none with a mixture (see references). I 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.

I was not happing with the string splitting and trimming method, so I investigated again how to get the information out of a JSON object in Java. I 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 I had to create a java class DuckPic that had fields for the fields in the JSON message and getters and setters. Then I 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);

I 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

### Different Formats Across Platforms

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

1. Make customer web pages flex to mobile screen size as this would allow customers to have a better user experience if using a phone instead of a desktop to order. It would be necessary for delivery drivers as they won’t have their desktops on their backs.

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

1. Different pizza sizes (large, small)
2. Create/customise own pizza
3. Use location to determine delivery availability and cost
4. Allow user to order without signing in (i.e. check out as a guest)
5. Other items (not just pizza)
6. Order history (previous orders)
7. Admin user to create new employee users.
8. 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.
9. Automatic timer to change status of pizzas from cooking to cooked.

## References

**Resources for collaborating on GitHub:**

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

**Resources for 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>

**Resources for passing parameters to other mappings:**

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

**Resources for CSS:**

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

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

**Resources for Animal API:**

<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/>