Vegan Pizzeria

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.

**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> | |
|  | Add new customer signup functionality to web page. | Add back button for cook when checking pizzas to cook. |
|  | 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. |
|  | Add sign out button on customer tracking screen. | Implement a conditional so that customer receives nicer messages than just ‘Cooking’. |
|  | Add payment API | Add API to pull in animal fact to customer tracking screen. |
|  | Refactor HTMLs so CSS code is not repeated? | Link orderliness and Order mappings so that an order is written to the database containing the lines of the order. |

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.

**Initial ERD**

A screenshot of a computer

Description automatically generated with low confidence

**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.

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.

**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>