

Assessment

In this assessment, you will be developing an online Pizza ordering application developed with Spring Boot; the orders are to be stored in a Redis database.

The application consist of 3 views

- View 0 - This is the 'landing' page. Customers will use this page to select their pizzas
- View 1 - After selecting the pizza, customers will enter their delivery address.
- View 2 - After placing the order, this view will confirm the order.

Figure 1 shows the flow of the 3 views

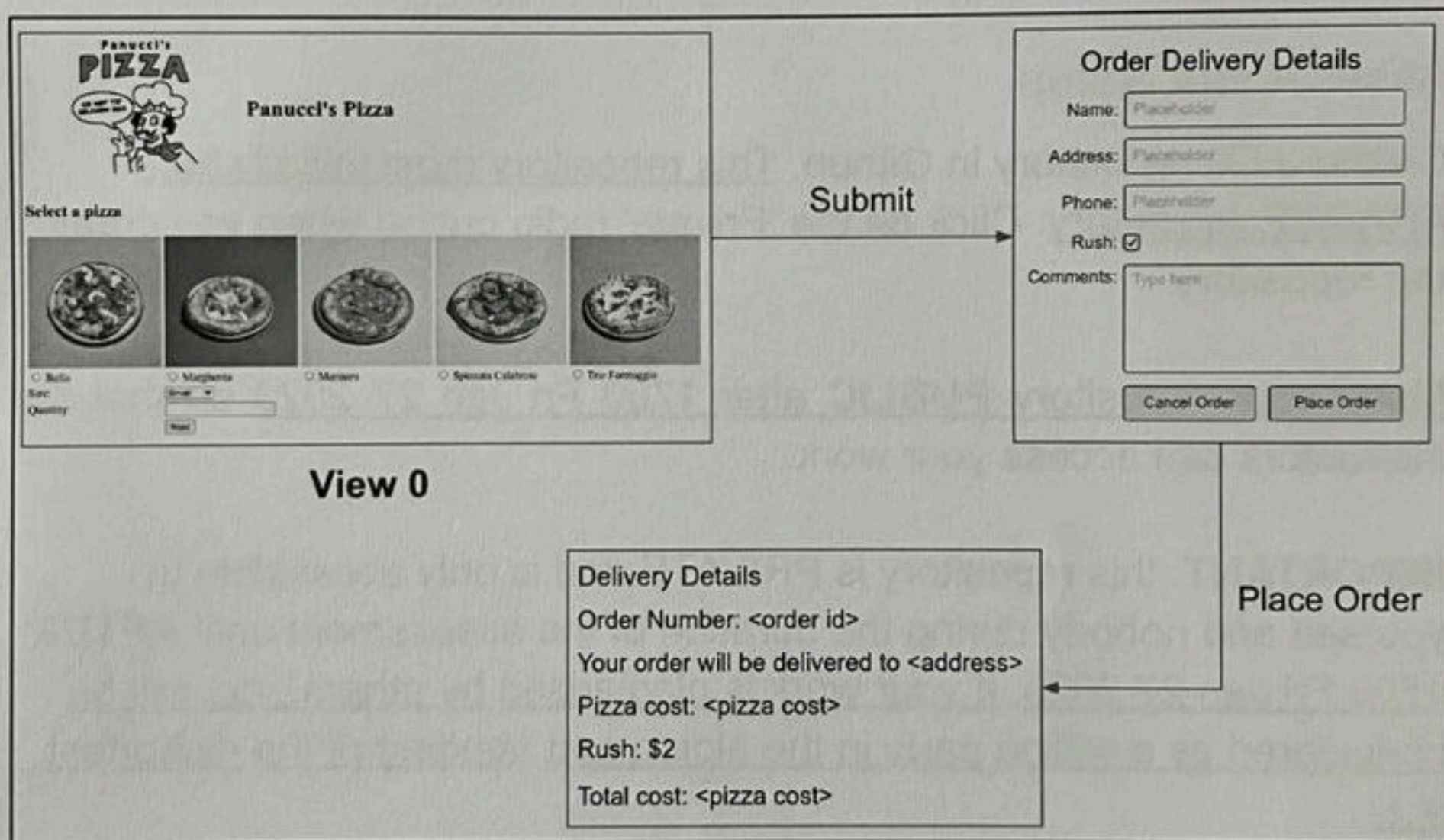


Figure 1

Detailed explanation of the 3 views will be given in subsequent tasks.

Task 1 (16 marks)

Generate a Spring Boot application; include the following dependencies in addition to the 'usual' dependencies for web applications. Your application should also include the following additional dependencies:

- Redis data and Jedis driver
- JSON-P

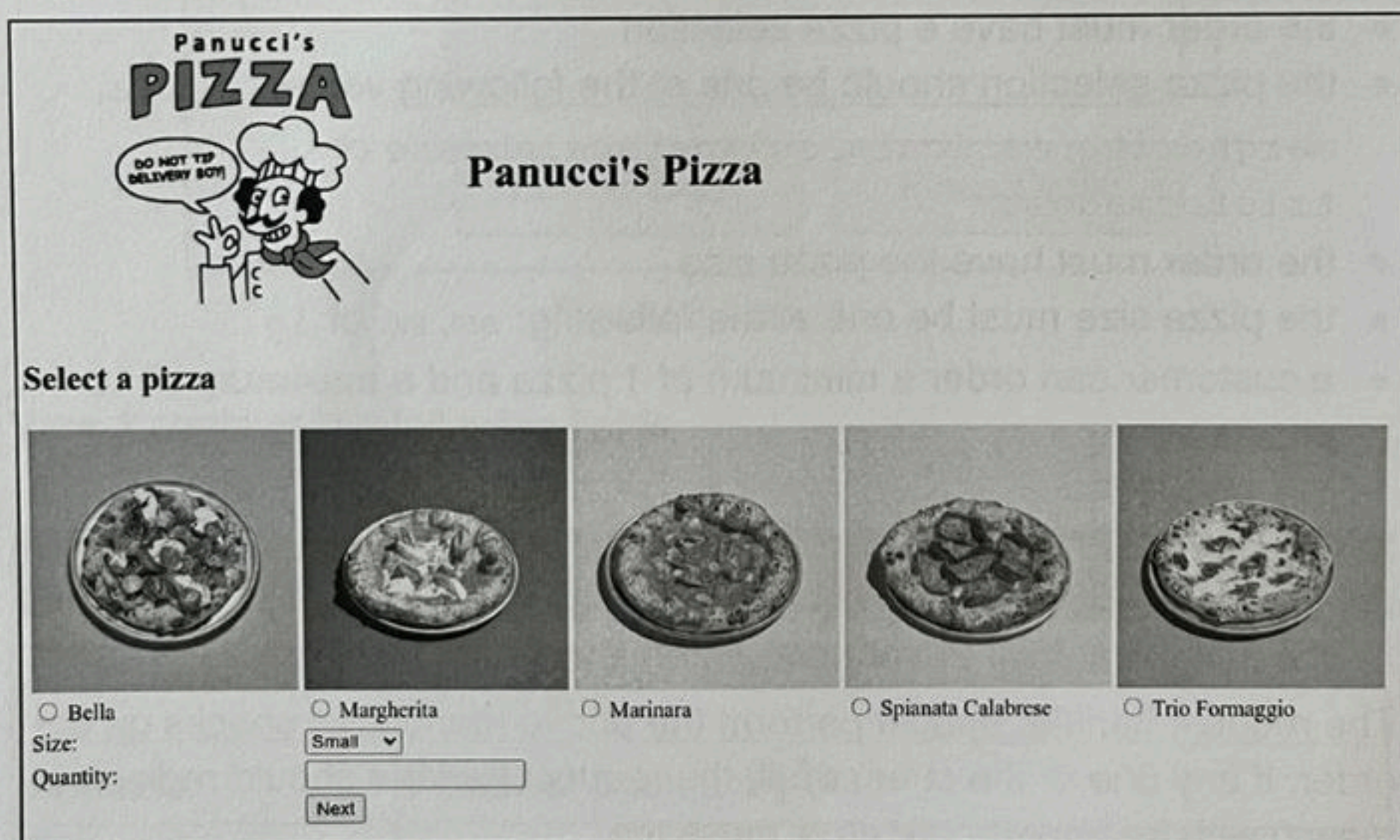
You may use any package name for your application.

You will be provided with a ZIP file; the contents of this ZIP file is the landing page (View 0). Add these files to your Spring Boot application.

Provision a project on Railway along with an instance of Redis.

Configure your Spring Boot application to use the Redis instance.

Start your Spring Boot application and open `localhost:8080`. You should see View 0 (Figure 2)



Panucci's PIZZA

DO NOT TIP DELIVERY BOY!

Panucci's Pizza

Select a pizza

☐ Bella ☐ Margherita ☐ Marinara ☐ Spianata Calabrese ☐ Trio Formaggio

Size:

Quantity:

Figure 2 View 0

Create a repository in Github for your assessment. You should now commit and push your assessment. Remember to commit and push your work often and do not wait until the end of the assessment.

Task 2 (38 marks)

View 0 is the landing page of the pizza store. Customers use this to place their order; to place a pizza order, they select

- one of the 5 type of pizzas
- pizza size
- the number of pizzas

When the Next button is pressed, the following HTTP request is made to the backend

```
POST /pizza
```

```
Content-Type: application/x-www-form-urlencoded
```

The following checks are then performed on the pizza order

- the order must have a pizza selection
- the pizza selection should be one of the following values: bella, margherita, marinara, spianatacalabrese or trioformaggio
- the order must have the pizza size
- the pizza size must be one of the following: sm, md or lg
- a customer can order a minimum of 1 pizza and a maximum of 10 pizzas

Write a controller in a class called `PizzaController`; write a request handler in `PizzaController` to process the above POST request.

The request handler should perform the above mentioned checks on the order. If any one of the checks fail, the request handler should redisplay View 0 with an appropriate error message.

Modify the landing page, `index.html` so that it can work with your `PizzaController`.

Task 3 (78 marks)

After verifying the pizza order details from View 0 are correct, display View 1 (Figure 3) to allow the customer to enter the delivery details.

Order Delivery Details

Name:

Address:

Phone:

Rush: ☒

Comments:

Figure 3 View 1

View 1 contains the following fields

Field Name	Description
Name	Customer's name, mandatory field, minimum 3 characters
Address	Address, mandatory field
Phone	Phone number, mandatory, 8 digits
Rush	Is this a rush/priority order? Defaults to no
Comments	Comments, optional field

When the Place Order button is pressed, the order details send the to the backend Spring Boot application with the following request

```
POST /pizza/order
```

```
Content-Type: application/x-www-form-urlencoded
```

Write View 1 and the request handler to process the above request. The request handler should perform the following:

a. Generate Order Id

Generate a random 8 character long order id. You can use the JDK's `UUID` class to generate the order id

b. Calculate Order Cost

Calculate the total cost of the order according to the following pricing.

The price of the pizza is as follows

- Bella, Marinara, Spianata Calabrese - \$30
- Margherita - \$22
- Trio Formaggio - \$25

Add the following multiplier for the pizza size

- Small - 1
- Medium - 1.2
- Large - 1.5

A rush order adds an additional \$2 to the total.

For example, if a customer orders 3 medium Trio Formaggio then the total cost will be $\$25 * 1.2 * 3 = \90 .

If the order is a rush order, then the total cost will be \$92.

c. Persist the Order

Configure your Spring Boot application to use the Redis on Railway. You should not hard code any sensitive information (eg Redis password) into your application source code.

Save the order as a JSON string in Redis. The pizza order should have the following JSON document structure

```
{
  "orderId": <order id, string>,
  "name": <name, string, view 1>,
  "address": <address, string, view 1>,
  "phone": <phone, string, view 1>,
  "rush": <true or false, boolean, view 1>,
  "comments": <comments, string, view 1>,
  "pizza": <pizza name, string, view 0>,
  "size": <pizza size, string, view 0>,
  "quantity": <quantity, number, view 0>,
  "total": <total cost, number>
}
```

4. Generate Order Confirmation

Generate and return View 2 shown in Figure 4.

Delivery Details

Order Number: <order id>

Your order will be delivered to <address>

Pizza cost: <pizza cost>

Rush: \$2

Total cost: <pizza cost>

Figure 4 View 2

View 2 should include the following details from the order order:

- order id
- delivery address
- pizza cost excluding the 'rush order' cost
- Rush order cost if selected. This information should not be displayed if a customer did not select 'rush order'

- total cost of the order, pizza cost plus 'rush order' option

The order id as a link (<a>); the <order id> in Figure 4 should be 'clickable'. The link resource (href) is as follows

`/order/<order id>`

where <order id> is the generated order id.

You may layout View 1 and View 2 according to your preference but all form fields and information specified must be present.

Task 4 (22 marks)

Write a REST controller to process the request when the order id (Figure 4) is clicked.

The REST controller should retrieve the order from Redis and return the order as JSON with an OK status.

If the order is not found, then return a Not Found status with the following JSON object

```
{  
  "message": "Order <order id> not found"  
}
```

Task 5 (4 marks)

Deploy your application to Railway.