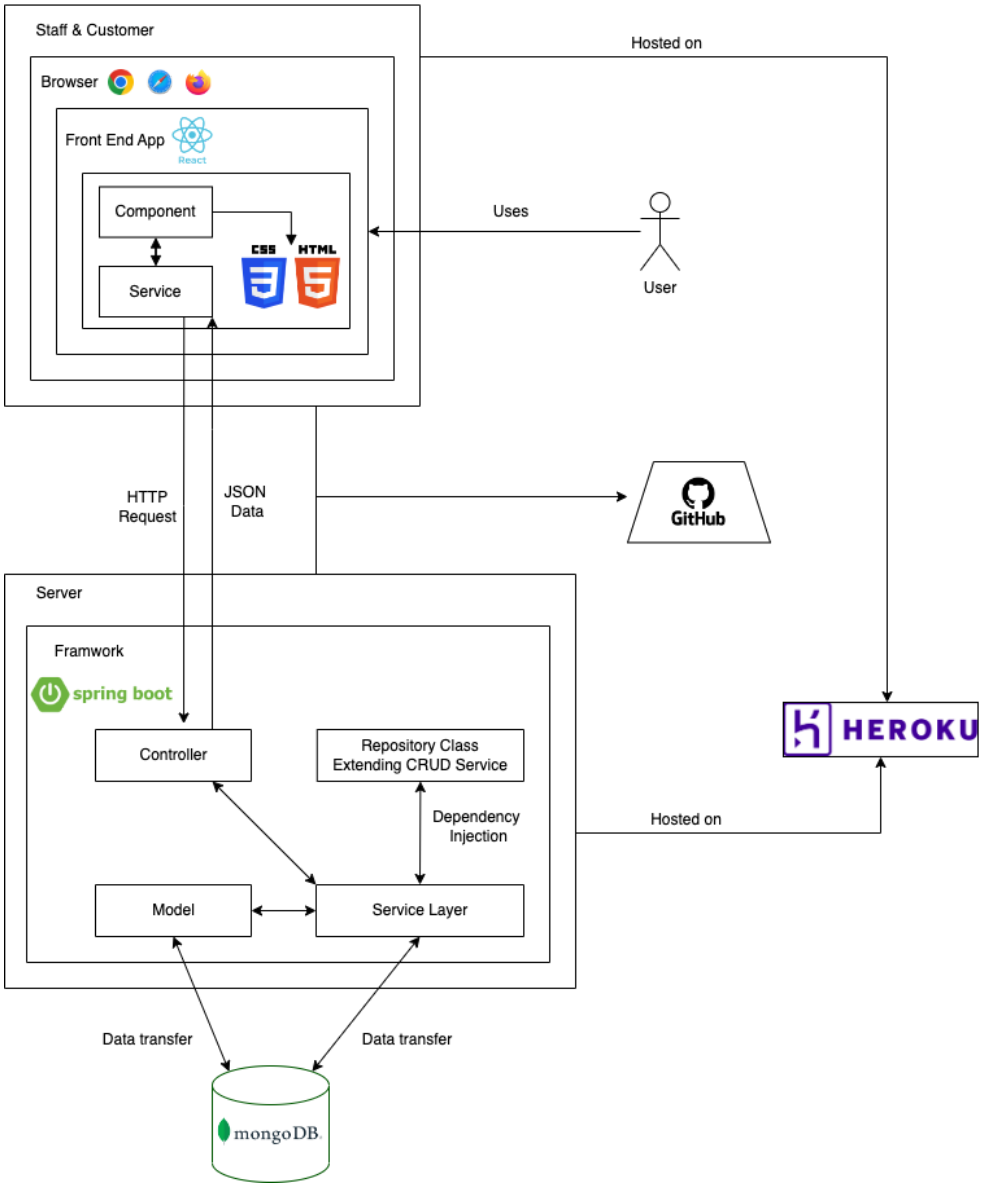
Handover

- Introduction

Our product Fidelma supports a variety of functionalities, e.g. menu management, raw materials management, image uploading, image receiving, order management, and place orders.

- System Architecture

System Architecture



- High-Level View:
  - Front end:
    - React uses the component service structure in which the service file is in charge to communicate with the controller in the back end and component in charge to render and display the HTML with CSS.
      - Service file works as an encapsulation that protect the component file which let it won't be influence by the environment of https request and responds(for example, the change the fetch api)
      - Component file in charge to receive the data that get by service file and render it with HTML and CSS, if need to change the database, then make a post or put request by service file.
  - Back end:
    - Spring Boot follows a layered architecture in which each layer communicates with the layer directly below or above it.
    - The Presentation layer
      - It handles the HTTP requests (GET/POST/PUT/etc)
      - Translate JSON parameter to object
      - Interate with Business layer
      - Controller class
    - The Business Layer
      - Handles all business logic
      - Use data in service provided by data access layers
      - Service class
    - The Persistence Layer
      - The persistence layer contains all the storage logic and translates business objects from and to database rows.
      - Models class
    - Database
      - Where CRUD operations are performed
      - Reporisty class

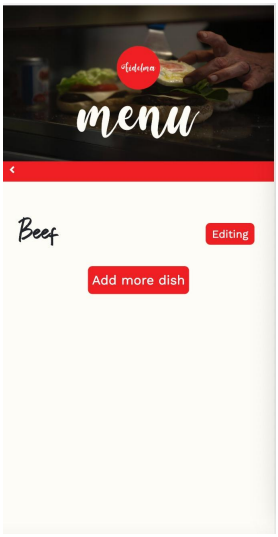
- Setups before run the product:
  - Download node.js and npm from https://nodejs.org/en/download
  - Download IntelliJ IDEA from https://www.jetbrains.com/idea/download
  - Update JDK to version 17 or later.

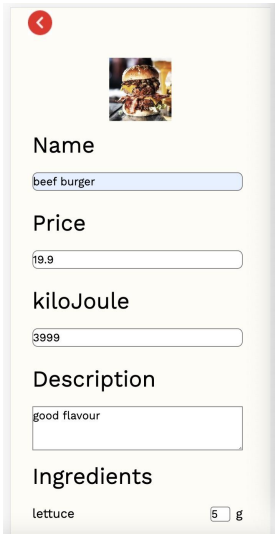
- How to run this product:
    - Run Server:
        1. Please refer to our readme.md for more detailed information
    - Run App:
        1. Go to directory ../comp30022/react-frontend
        2. Type ‘npm install’ in command line to install packages
        3. Type ‘npm start run’ in command line to run the app
        4. Browser visits localhost:3000.

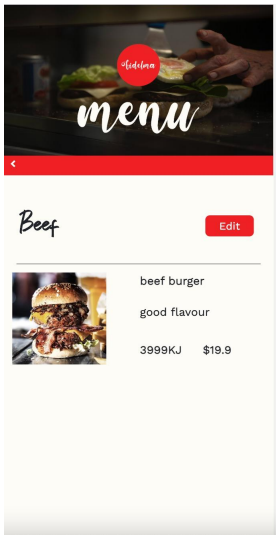
- QA workflow

Add new dish(‘edit new dish’ is similar to this function)

1. The menu page is empty first.
2. The staff click the ‘add more dish’ button
3. Enter the second page below to add a new dish
4. Save the dish and go back to the menu page
5. Now the menu page has added the new dish successfully, it has been shown in the third picture below.

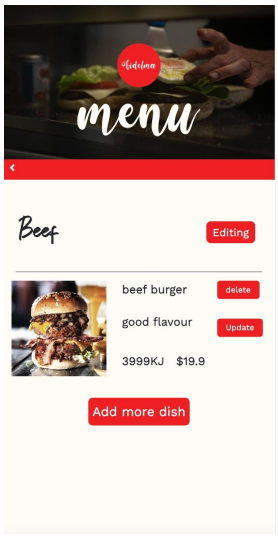


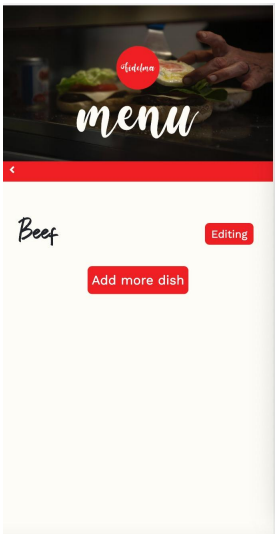




Delete Dish

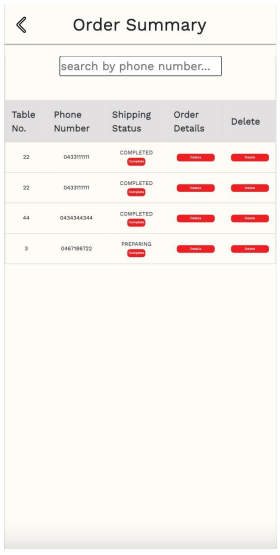
1. The staff need to click the ‘delete’ button
2. Then the dish will be deleted successfully

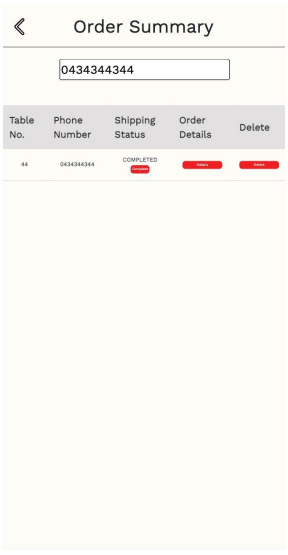




Search the order by phone number

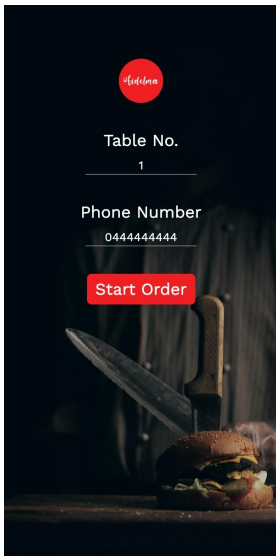
1. All orders has been listed
2. Enter Australian phone number in the input field
3. Successfully find the order

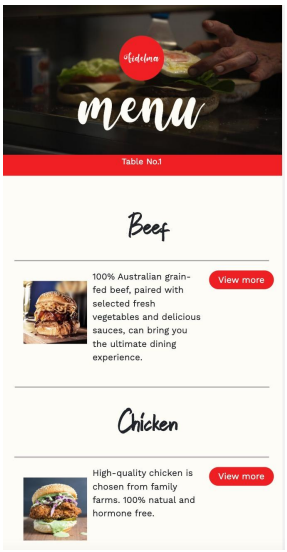


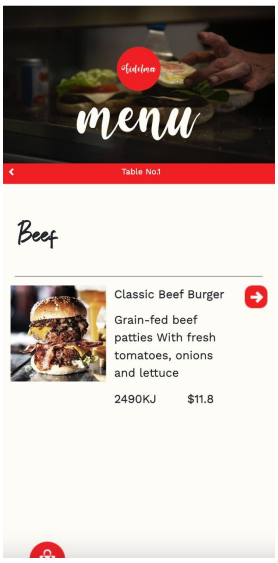


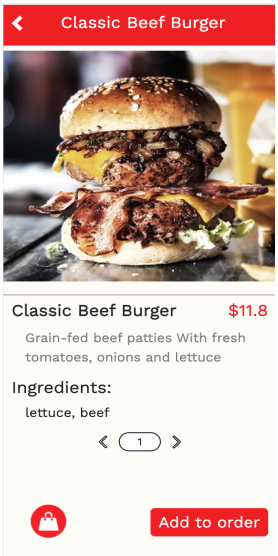
Order a meal

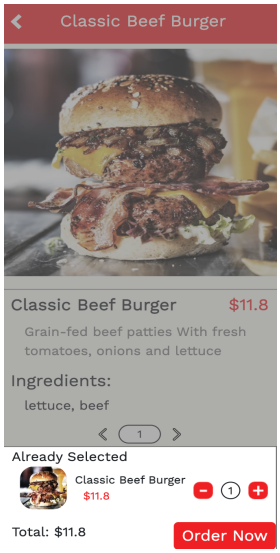
1. Enter the valid table number and phone number
2. Select dish categories and click ‘view more’ button
3. Select dishes by clicking the ‘arrow’ button
4. Select the quantity of the dish at the description page and then add it to the shopping cart
5. Check all dishes in the current shopping cart by clicking on the ‘cart’ button and clicking the ‘order now’ button would leads to the confirm page
6. Submit the order by clicking the ‘order now’ button at the confirm page

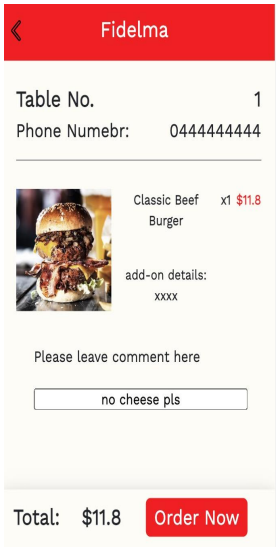


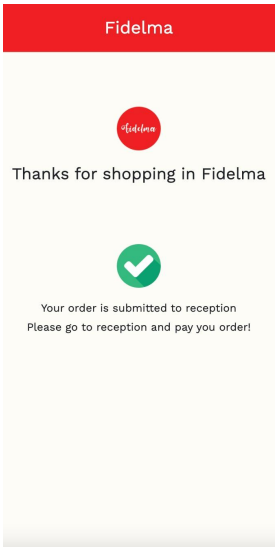












- Source Code Commits
  - Branch : DevelopmentFinalise(local use environment)
    - Commit : 4ea5217ee521f332b72bd223824b91b6be6700b3
  - Branch : main(deploy use environment)
    - Commit : 18e6782ca20e50402718614dbe18f96be252cd51

-

- Database Schemas
  - Food:

```
Map<String, Double> components;
String type;(Mandatory)
byte[] image;
String id;(Mandatory)
Boolean isSoldOut;
String description;
int kiloJoule;
boolean isCrash;
```

  - Ingredient:

```
Double quantity;(Mandatory)
String id;(Mandatory)
String name;(Mandatory)
Double price;(Mandatory)
```

  - Image:

```
String id;(Mandatory)
Binary image;(Mandatory)
```

  - Order:

```
String id;(Mandatory)
String name;(Mandatory)
Integer tableNumber; (Mandatory)
String phoneNumber; (Mandatory)
Map<String, List<Object>> cart;
OrderStatus orderStatus;
```

  - Staff:

```
String id;(Mandatory)
String Username; (Mandatory)
String firstName;
String lastName;
String email;
LocalDate dob;
String phoneNumber;
```

```
String Address;
```

- Source/Version Control Procedures
    - At least need three branches for future development, one for development, one for deployment, one for testing.
    - Suggested to add two more branches, one for front end development and one for back end development for any future development process.

Key Source Code Sections

Food automatically be sold out

```
{this.state.foods.map((dish) => (
    <div className="foodUnit" key={dish.id}>
   <hr className="separateLine"/>
    <div className="foodBox">
    </div>
    <div className = "staffMenuGridContainer">
        <div className = "staffMenuPic">{dish.image !== undefined &&<img className = "gridPic" alt = '' src={`${dish.image}`} />}</div>
        {
            dish.crash === false && <div className = "staffMenuName"><strong>{dish.name}</strong></div>
        }
        {
            dish.crash === true &&
            <div className = "staffMenuName crashDish">
                <div><strong>{dish.name}</strong></div>
                <div><strong>Ingredient Error!</strong></div>
            </div>

        }
```

Front end accept the information from backend which will contains a true or false value on if the food is sold out, if received data is sold out, will use a displayed text to replace the view more button(which leads to view more page and can add food to cart), make customer unable to add sold out food into the cart, and let customer know it is sold out.

```
public boolean checkAvailability(Food food){
    for(String ingredientName: food.getComponents().keySet()){
        Ingredient ingredient = ingredientService.findIngredientByName(ingredientName);
        if(ingredient == null){
            return false;
        }
        Double currentQuantity = ingredient.getQuantity();
        Double requiredQuantity = food.getComponents().get(ingredientName);
        if (requiredQuantity == null){
            requiredQuantity = 0.0;
        }
        if(currentQuantity - requiredQuantity < 0 ){
            return true;
        }
    }
    return false;
}
```

After receiving the food, the system iterates through every ingredient inside the database. If the current quantity is less than the required quantity from the food, it will return false, which means that the food is not available. On the other hand, the method will return true, indicating that the food is available.

Add a new raw material

```
<form data-testid = 'form' >
<div>
<h3 className="addPopupSubTtile">Name</h3>
<input className="addPopupInput" type="text" data-testid = "name" aria-label= "name" maxLength = "15"
value = {this.state.name} onChange={this.nameHandler} />

<div>
{this.state.name && !(/^[a-zA-Z]*$/).test(this.state.name) && <span className="errorAddPopup" data-testid="error-msg-name">Please enter a valid name.</spa
```

For the front end, the user needs to type input into several fields. The above picture shows the ‘name’ field’. When the user is typing, the form will receive the data from the input field. Finally, when the user gives the value for all input fields, form data will be sent to the backend by ‘post’.

```
@PostMapping("/ingredient")
@ResponseStatus(HttpStatus.CREATED)
public Ingredient addIngredient(@RequestBody Ingredient ingredient){
    ingredientService.addIngredient(ingredient);
    return ingredient;
}
```

Here is the postmapping function in the backend. Backend uses this function to receive the form data and save it into the repository.

Place orders

```
saveOrder = (e) => {
    e.preventDefault();
    let cart = {};
    for (var i = 0; i < this.state.foodsInCart.length; i++) {
        cart[this.state.foodsInCart[i].id] = [this.state.foodsInCart[i].quantity,this.state.foodsInCart[i].price,this.state.foods
    }

    const unique_id = uuid();
    let order = {
        id:unique_id,
        tableNumber : this.state.tableNum,
        phoneNumber : this.state.phone,
        name : this.state.orderComment,
        cart
    }
    console.log("order=> " + JSON.stringify(order));
    axios.post( REST_API + "/customer/orderConfirm",order).then( (res) => {
        console.log(res.data)

        if (res.data === 1){
            this.openPopUpWindow("ingredientWarning")
        } else if (res.data === 2){
            this.props.history.push("/submitPage",this.props.location.state);
        } else {
            this.openPopUpWindow("systemErrorWarning")
        }
    })

}
```

For the front end. All order data will be processed as the formdata. Then if this order is available to be placed, it will be sent to the backend by ‘post’.

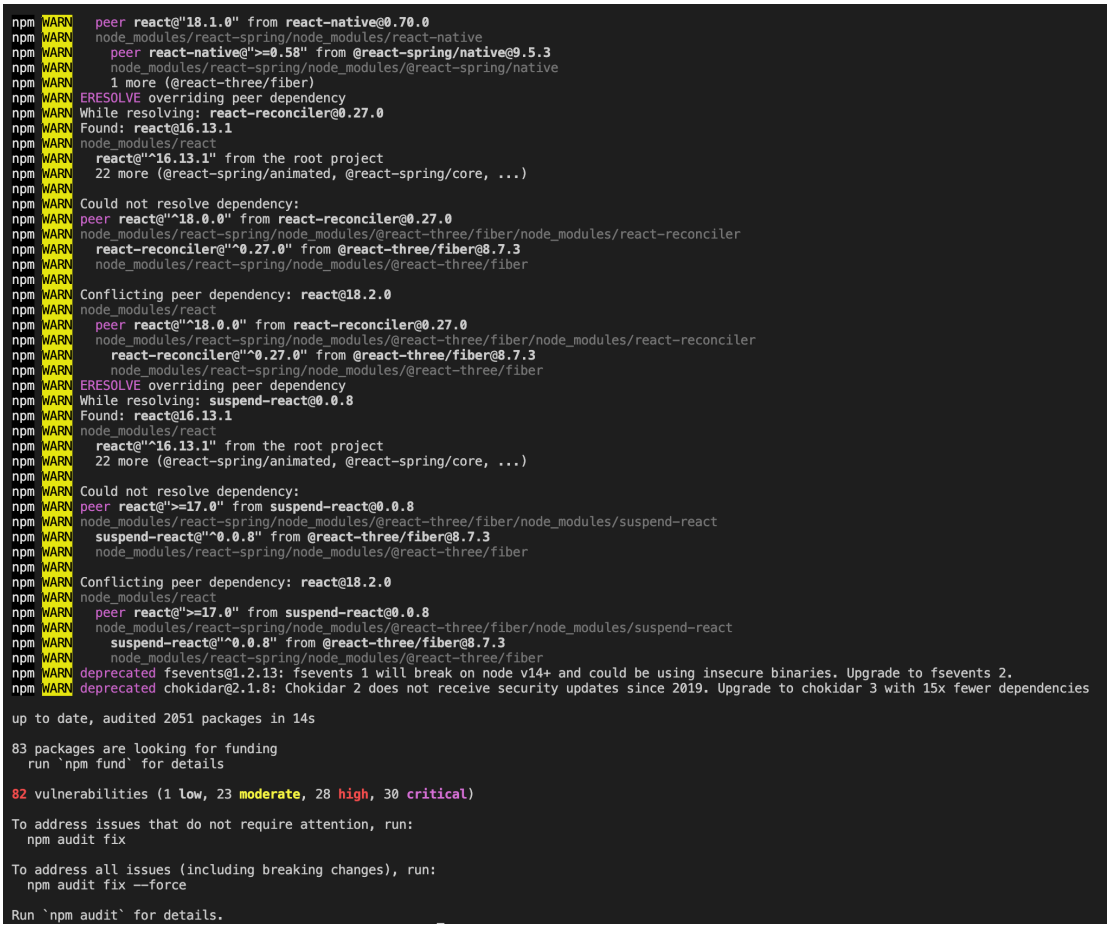
```
@PostMapping("/orderConfirm")
@ResponseStatus(HttpStatus.CREATED)
public Integer addOrder(@RequestBody Order order){
    System.out.println(order.toString());
    if (orderService.checkQuantity(order) == 0 || orderService
        return orderService.checkQuantity(order);
    }
    orderService.UpdateQuantityForIngredient(order);
    orderService.addOrder(order);
    return orderService.checkQuantity(order);
}
```

For the back end. The order information will be parsed into an order object for storing and processing. Before the orders are saved, we will need to do validation checks, checking the ingredients inventory to see if we have enough raw material to make the dishes.

- Coding Standards

Standard	Details
Variable Name	Java camel-case convention: start with a lower case letter, following word start with upper case letter(e.g. : dishType)
Naming	Use descriptive names for all variables, function names, constants, and other identifiers
Class name	Start with an upper case letter, following word also start with upper case letter(e.g.: FoodController)
Method name	Start with a lower case letter, following word start with upper case letter(e.g. : checkDishType())
In-line comment	Used to explain complicated sections of code
Spacing	Indent four space by a tab when beginning a new block.
Transfer data between pages	Must use the list type to transfer data between pages in frontend.

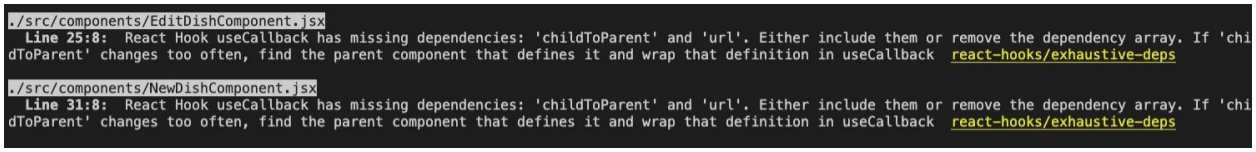
- Risk in setup and note for future development on react：
  - Our team is used a older version of react, so need download the old version of react to properly use this product
    - The way to download the old version of react is by command npm install, the older version of react will be directly installed as the information record in package-lock.json and package.json
    - When installing it through npm, warnings like the picture below will occur, it’s not influence the product to work, just ignore it.



- Since the product is made by the older version of react, so for the future development, if upgrade to newer version, need replace all the place that involve data transfer between pages which use this.state.history and this.props.location.state in code of front end to the navigate hook in newer version of react, the history hook had been deprecated in newer version of react.
- Since the product is made by the older version of react, a lots of external library that require peer dependency will not able to install, if must resolve this problem, please upgrade to a newer version of react and make maintenance to code as last dot states.

- Unsolved Bugs and Warning :
  - No common bugs.

  - For the warnings, there are two warnings generated because the product is using the older version of react.



- Deprecate file contains:
    - Security Feature: This security feature is half-way through the completion process. Due to the size and scalability of the project, user-based security is no longer required as we have only one admin user in the system. However, if in the future there are more admin accounts or more variation of the types for admin accounts, this feature can be implemented to assist the overall sustainability of the project structure as well as the security architecture. It is currently located at the **Deprecated/Security** folder in the repo.
    - Tests files(_test_ and SpringJUnitTest): All the test file for the front end and back end that generate through the development process.
        - See Testing Plan: https://zizhzhang.atlassian.net/wiki/spaces/THE/pages/7897505/Testing+plan+for+sprint+3