**Live Demo**

**Intro [Liwen]**

Good afternoon all, today Team Fantastic Four is here to present to you Two Tyred, our one-stop solution for cycling enthusiasts. Two Tyred is designed to help you with all your cycling route planning needs. Our application aims to allow you to create your own custom cycling route and track all your rides. You can also interact with routes created by others and cycle their routes. However, if you are feeling adventurous today and would simply like to hop on a bike for a workout without worrying too much about where you are headed, there’s also an automatic route planner for you. Let’s jump into how this application works.

**Authentication [Liwen]**

Users of the Two Tyred application will need to create an account before using our services. Once the account has been created, the user will be automatically logged in.   
  
Once they are logged in, users will be brought to the main dashboard. From here, they will be able to see all the routes that others have created. They are able to filter these routes based on recency or the number of likes that they’ve received from others.

**Interacting with route card [Dao Zheng]**

Users are able to interact with these route cards in 3 ways.

Firstly, they can like and unlike a route to show support for routes created by other users.

Secondly, they can favourite a route. Favouriting a route will add the route to the “favourite” section on their User Page. This allows them to continue interacting with the route card later on.

They may also remove it from their favourite section by unlikely it later on.

Lastly, they can click on the map to view more details about the route. From here, the user can cycle a route created by others.

**Manual Route Creation [Dao Zheng]**

If the users wishes to create a custom route, he can select the create new route option. From this page, he can select a start point and an endpoint. He may also choose to increase the number of intermediate points by increasing the number of destinations. Once he’s satisfied with his selection, he may choose to cycle the route. Our application uses the OneMap API and various Data.gov.sg APIs to inform the user about the conditions of the chosen route. This includes distance, weather forecast, and pollution index of the route.

If the user is not satisfied with the route, he may choose to edit the route. For instance, this user wishes to cycle further than \_\_km. He can select the edit route option and continue to add destinations.

If the user if satisfied with the route, he can then choose to cycle the route. This route will then be added to the past routes tab on his user profile page.

**Automatic Route Creation [Hui Xiang]**

However, if the user does not wish to manually the points and simply wishes to get hop on to his bike and cycle, he can use our imfeelinglucky function instead.

The user can use his current location or select a custom starting point.

After which, he will enter the desired distance of his trip, and hit “Generate route”.

He will be redirected to a similar route description page.

**Progressive Web Application [Hui Xiang]**

However, our application is not merely a web application. We considered the fact that users may wish to access the application from their mobile phones and look at the map while they cycle. As such, we’ve decided to develop a progressive web application instead. Mobile phone users can simply click on this button to save the application to their mobile phones. Thereafter, they will be able to access Two Tyred like any other application.

**Presentation**

**Slide 6 (Evonne):**

For our system architecture diagram, it is the high level structure of our web application. And if you examine our class diagram, you can see that our architecture diagram is very similar to it, just that it is more macro. So allow me to take you through the different parts of the diagram. In designing this architecture, we used the loose coupling and high cohesion principle. You can see that our diagram has 3 different layers, namely application frontend which contains the boundary classes, application backend which contains the app logic and database which contains all the data of the users and routes. It adopts loose coupling as each layer can only communicate with the adjacent layer. For example, app frontend cannot communicate with database. This adds security to our app as well as allows for better scaling of it.

**Slide 7 (Ivan):**

On this slide, you can see our use case diagram which encompasses of all the use cases of our web app. For this presentation, we will only be presenting on 2 uses cases. Namely: route planning and Interact with Route Card

**Slide 8 (Ivan):**

Let’s us first talk about the route planning use case. The main functional requirements for this use case is the following:

* User can add extra destinations
* The weather and route information will be shown
* User can choose to select or reject the route

**Slides 9 (Ivan):**

On the left is the relevant class diagram for our Route planning function. The boundary class consists of the route planning page where the user will initiate the route planning function and route description page where the route will be displayed after generation. There is also the Route planning system which serves as the control class that calls the onemap API and generates the route based on User inputs. Lastly we have the Entity classes of route by destination and route by distance which are being used by the route factory to generate either of these two types of routes.

On the right is the sequence diagram for this use case. We can see that the user will first select “Create New Route” on the route planning page and this will in turn will call the planRoute method based on the chosen points for the user. In the event that the route was successfully created, it will then be stored into the database and upon successfully saving in the database, will show the completed route with its route description on the route description page.

**Slide 10 (Ivan):**

One of the designs we have chosen for this use case was to use strategy together with factory design to support envisioned further upgrades. While working on the route planning feature, we wanted our app to be more robust and have more options for users to create routes. To enable this, we decide to use the factory design for our Route planning interface to dynamically instantiate classes based on which route planning type was chosen. This works hand in hand with strategy design which encapsulates object creation and makes it easier to add new classes by implementing the same interface.

The implementation can be seen on the class diagram at the left where we have a routefactory that can generate a route. This routefactory is an abstract class that can generate either route by destination or route by distance, both of which being entity classes and stored in the database. Thus, depending on the input, only one class will be created.

**Slides 11 (Ivan):**

Lastly, we carried out black box testing using invalid and valid equivalence classes. We tested the route factory control class using invalid and valid start and end points and we achieved the results as shown in the table

**Slide 12 (Kai Jun):**

Next, let’s take a look at the Interact with Route Card use case

For this use case, there are 3 boundary classes, namely the Route Card Set, the Dashboard, and the User Profile. The route card set encompasses all the route cards, while the dashboard and user profile comprises of the route card set as seen from the class diagram. There are 2 control classes. The first is the application backend, which is involved in the CRUD functions, and the second is the Route Information Subject, which is used for Observer Pattern which I will elaborate later.

The main crux of this use case lies in the route card. The route card displays the route information such as the startPt, endPt, distance, route and likeCount. There are 3 main ways to interact with the route card: 1) like/ unlike route, 2) favourite/ unfavourite route, 3) to view more

As seen from the Sequence Diagram, when you click the like button, the likeRoute method will be called, and the backend will increase the likeCount in the database by 1. When you click the favourite button, the favouriteRoute method will be called, and the backend will add the routeId to the array of favouriteRoutes. On the other hand, when the user clicks into the route card to view more details, the backend will retrieve the route information from the database and reflect it on the route description page

**Slide 13 (Kai Jun):**

For this use case, our group adopted the Observer Pattern. In this case, the subject is the backend and the observers are the dashboard and the user profile, which both implements the abstract class Route Information. The reason for using observer pattern is because a change in the route data in the backend will need to be updated on both the dashboard and the user profile. Hence, by adopting observer pattern, the user profile and the dashboard will be automatically notified of the changes when the backend route data is updated. Doing so enables loose coupling between the route data and the dashboard and user profile

**Slide 14 (Kai Jun):**

Next, we carried out whitebox testing for the favourite route function. For this function, it is important to determine if the user id and route id exists, as well as whether the route has already been added to the array of favourite routes. Hence, as seen from the control flow, there 2 decision points given by the red boxes, and 3 basis paths shown in blue boxes.

**Slide 15 (Kai Jun):**

Lastly, this slide shows the list of inputs and outputs corresponding to the 3 flows mentioned earlier

**Slide 16:**

With that, we have come to the end of our presentation. Thank you for your attention!