4for4 - Second Iteration Doc

Haoran Geng (hg2496)
Jerry Lin (sl4299)
Yun Wang (yw3180)
Xijie Guo (xg2291)
Instructor: Prof. Gail Kaiser

User Stories:

1. As a traveler who doesn't want to spend so much time on route planning, I need a personal travel assistant to customize a travel agenda so that I can have a travel plan that can get me to selected places with least time spent on the road. My conditions of satisfaction are:
    - I can select spots I want to visit and the system will generate a travel plan for me.
2. As a traveler who prefers flexibility in transportation methods, I need a personal travel assistant to tailor the itineraries based on different modes of transport, so that I can have the quickest and the most flexible way to travel in the New York City. My conditions of satisfaction are:
    - I can obtain travel plans based on different modes of transport
    - I can prioritize the distance or travel time for my travel plan. Thus I can select the most suitable plan for me.
3. As a traveler who want to visit a lot of places, I need a system that can save all my potential travel plan, so that I can select the plan I want and do not need to fill in the travel information every time. My conditions of satisfaction are:
    - I can login the system with my username and password.
    - I can see my saved travel plan each time I login.

Use Cases:

1. Use Case Name: Select Plan

    Actors: Traveller (Has existing account), selection GUI(a system allow user to select condition), nearest neighbor shortest path algorithm(generate the travel plan)

    Precondition: Traveller login into the system with account.

    Basic flow:
    1. Traveller type one spot address he or she want to visit on selection GUI.
    2. If the address is valid, traveller can select the detailed spot address generated by the selection GUI and add it into the list on a GUI.
    3. Traveller repeat 1 and 2 to add more spots he or she want to visit into the list.
    4. After select all the spots, traveller can ask nearest neighbor shortest path algorithm to generate travel plan.

5. Nearest neighbor shortest path algorithm generate the plan from the list based on the travel time.
6. Traveller can save the plan (into the database) or delete the plan to start a new plan.

Alternatives:

1. If the traveller type in the invalid spot, the GUI will show the address is invalid and do not allow traveller to add the invalid address into the list.
2. If the traveller type in the spot outside of the New York City, the GUI will show the address is outside of the New York City and do not allow traveller to add the address into the list.
3. If the traveller's spots list is impossible to generate a travel plan, the GUI will show the spots list is impossible to generate a plan.

2. Use Case Name: Select Plan base on transportation

Actors: Traveller (Has existing account), selection GUI(a system allow user to select condition), nearest neighbor shortest path algorithm(generate the travel plan)

Precondition: Traveller login into the system with account.

Basic flow:

1. Traveller select the four transportations (walk, bicycle, drive and public transit) he or she want on the GUI.
2. Traveller select the distance or travel time he or she want the algorithm to generate based on.
3. Traveller type one spot address he or she want to visit on selection GUI.
4. If the address is valid, traveller can select the detailed spot address generated by the selection GUI and add it into the list on a GUI.
5. Traveller repeat 1 and 2 to add more spots he or she want to visit into the list.
6. After select all the spots, traveller can ask nearest neighbor shortest path algorithm to generate travel plan.
7. Nearest neighbor shortest path algorithm generate the plan from the list based on traveller's selection.
8. Traveller can see the distance or travel time between each spots
9. Traveller can save the plan (into the database) or delete the plan to start a new plan.

Alternatives:

1. If the traveller type in the invalid spot, the GUI will show the address is invalid and do not allow traveller to add the invalid address into the list.

2. If the traveller type in the spot outside of the New York City, the GUI will show the address is outside of the New York City and do not allow traveller to add the address into the list.
3. If the traveller's spots list is impossible to generate a travel plan, the GUI will show the spots list is impossible to generate a plan.

3. Use Case Name: Sign up and log in

Actors: Traveller (Has existing account), sign up GUI(a system allow user to sign up username and password), login GUI(allow user to login with password and username), database(store account information and plans)

Precondition: Traveller sign up and login into the system

Basic flow:

1. Traveller who without the account information will click the sign up button on the login GUI to get into the sign up GUI.
2. Traveller will type his or her account information on the sign up GUI to create an account.
3. Sign up GUI will save the account information into the database.
4. After complete the signup step, traveller can login into the system with username and password.
5. Traveller can begin using the system or check saved plans

Alternatives:

1. If traveller type in the wrong username or password, the login GUI will show the error to the traveller and forbid the traveller to login into the system.
2. If the user signup using invalid password or username, the signup GUI will show error and report to the traveller.

Test Plans:

Our tests includes:

1. Sign up test
2. Authentication test
3. Login test
4. Add spots test
5. Delete spots test
6. Save travel plan to database test
7. Travel plan generation test

Sign Up Test

Test signup method in LoginController class with two equivalence classes:

1. Sign up with a new user name which is not present in database: expect user to successfully sign up

2. Sign up with an existing user name which is present in database: fail to sign up and let user enter a new username

Authentication Test
Test authenticate method in LoginController class with three equivalence classes:

1. The username to be authenticated does not exist in database(no user); return -1 as the authentication status
2. Both the username and password exist and match with each other: return 1 as the authentication status
3. The username and password do not match with each other: return 0 as the authentication status

Login Test
Test login method in LoginController class with three equivalence classes:
First obtain the authentication result.

1. Log in with a username that does not exist in database: fail to login and let user retry
   Boundary condition: input can only be -1
2. Log in with invalid username and password that do not match with each other: fail to login and let user retry
   Boundary condition: input can only be 0
3. Log in with the correct username and password that are present in database: successfully logged in
   Boundary condition: input can only be 1

Add Spots Test
Test addSpot and other relevant methods in SpotCollection class with following equivalence classes/test cases:

1. Add a spot that is already in the list; fail to add and let user retry
2. Add an invalid spot that is not present on Google maps: fail to add into list and let user retry
3. Try adding a spot before clicking on Start a Plan button: invalid operation and no effect on user interface

Delete Spots Test
Test DeleteSpot and other relevant methods in SpotCollection class with following equivalence classes/test cases:

1. Delete a spot when the list is empty: fail to delete
2. Delete a spot when the list is not empty: successfully delete the last spot in the list
3. Try deleting a spot before clicking on Start a Plan button: invalid operation and no effect on user interface

Note: We might allow user to delete the spot that he/she chooses

Save travel plan to database test:

There are three equivalence classes so far:

1. Save an empty spot list to database: unable to save
2. Save a spot list containing invalid spot(s) to database: unable to save
3. Save a spot list with valid spots: successfully saved to database

Travel Plan Generation Test

Test ShortestRoute class and relevant functions in UIController class with following equivalence classes/test cases:

1. Try generating a travel plan with invalid spots(spots not in New York) or empty list: fail to generate the travel plan
2. Try generating a travel plan before clicking on the Start a Plan button: invalid operation and no effect on user interface
3. Generate distance-based plan in terms of different traffic modes respectively
4. Generate time-based plan in terms of different traffic modes respectively

Coverage:

In our daily development process, we used built-in coverage tool in Intellij to measure coverage. The tool allows us to accumulate coverage in all unit test classes. It indicates percentage of classes and lines are tested in all classes scope and percentage of classes, methods, and lines for each class.

For branch coverage tests in pre-commit and post-commit, we use jacoco (https://www.eclemma.org/jacoco/trunk/index.html) to automate the process. The workflow is built in to ant's build file (build.xml) and bundled with the unit test scripts. When unit tests are run, jacoco measures the coverage percentage and puts a log in the log folder along with static analysis reports. We achieved 70% coverage, but this number will increase as we continue to add unit tests.

Github repository link:

https://github.com/MousseKwok/Travel-Assistant