

CMPT 276: DiningPal

Requirements and Specification Document

(Iteration 3 – Final Report)

Git Link: <https://github.com/ZiyiAn/DiningPal>

Web Link: <https://diningpal-czhao.herokuapp.com>

Presented by Group: Segfault

Ziyi An	301371687
George He	301315033
Bowen Wang	301267523
Zihan Wang	301329429
Chen Zhao	301308092

Aug 1, 2019

Index

Index	2
Project Abstract	3
Competitive Analysis	3
Customer Targeting	4
User Stories Simulation – Regular Users	4
1) Register and Login	4
2) Logout	5
3) Google Map API	5
4) User Interface	7
5) Meal Plan && Dining Options	8
6) Smart Recommendations	9
7) Food Digger	9
8) Public Chatroom	10
9) Private Chatroom	11
10) Open Weather API	12
11) Static Calendar	13
12) Brief Overview and Checklist	13
User Stories Simulation – Administrators	14
1) Login	14
2) Add / Modify / Delete Users Information	14
3) Manage Financial Information	15
Scope and Velocity Discussion	16
Technologies Used	18
Final Word – Feelings from Segfault	18
Appendices	18

(Notation: Iteration 1, [Iteration 2](#), [Iteration 3](#))

Project Abstract

DiningPal is an online web application allowing multiple users to make plans of dining together. After sign-up/login, users can propose their favorite restaurant(s) (with ranks) and find friends (or strangers), who also prefer similar restaurants during that specific time to have dinner together. The features of this application will include locating users, positioning/ranking/auto-recommend restaurants, with future implementations including chat rooms, users filter system, map navigation, coupon search, weather condition check and so on. Though this application can be used to arrange other places of collective activities, such as gym, theaters, shopping malls etc., it is primarily designed to help people find diner mates in an easy and comfortable way.

Competitive Analysis

The following real-world problem is poorly solved: instantly finding someone to dine with is usually hard in real life because of finite friends and schedule constraints. Even if people find any friend to meal with, they sometimes end up struggling of choosing among different restaurants. There does not exist such a popular application to solve this problem perfectly, for example:

- Yelp focuses on offering recommendations of restaurants for single users, which lacks an interface for friends/strangers to dine together. The friend list works poorly and is basically static. Meanwhile, it does not make suggestions for users on how busy the restaurant usually is, how conveniently to reach there, etc.
- Facebook has pretty good functionalities for friends to chat on where to eat, finding recreation and vote for what they prefer. However, it performs poorly on dealing with conflicts (no recommendation) and communicating with strangers. We need to draw support from other platforms such as Google Map.

DiningPal aims to offer diners a sense of closeness and fewer chances to miss a great meal offer. Hence, our application concatenates main advantages from such indirect competitors and makes use of their useful functionalities. Through DiningPal, our specialties are:

- Make both contacting with friend(s) and finding stranger(s) to dine with possible. We introduce an innovative platform to close the distance between people.
- Design a smart recommendation for dining group to get over conflicts, based on their different dining habits and realistic conditions (e.g. restaurant busyness, traffic jam)

- Offer comprehensive utilities all in one: plan further leisure activities after meal, coupons or special discounts...these all could be done in our application and become simple and direct. Dramatically, our app could link to other groups' work like fitness services app.

Customer Targeting

Our group comes up with three targets that can be regarded as our potential main users:

1. High school / college / university students and junior employees ages from 16-30 years old, who like to dine out, feel open to new things and make new friends.
2. Travelers who are unfamiliar with the new places and want to meet up with local people/friends. Inversely, local people and guides can meet visitors through this app.
3. Families and friends who have problems on deciding what to eat. They may also need suggestions of following up leisure activities.

User Stories Simulation – Regular Users

Regular users include almost all of our main targeting customers, from friend group of students to travelers who feel alone. However, they have small differences in accessible functionalities, depending on their identities. Some current and future potential user stories are:

- 1) Register and Login to the User Page (Home Page and User Interface) – Iteration 1
 - a) Preconditions: For a new user, suppose he/she registers successfully in our home page. For existed users, he/she may try to login.
 - b) Postconditions: For new users, if the username and all information are legal (there should be more constraints on registration in later iterations), our application will go back to login page and note the user to login. The database should have successfully stored their information. For existed users, if they input valid login information, the page directs to the user interface.
 - c) Acceptance tests (Triggers):
 - i) If the tests catch any illegal input (including usernames that already exist), it shows “invalid username / email” or “no matched password”, does not process the query and go back to the Home Page.
 - ii) If the query does not process successfully, it generates to “error” page.
 - iii) If the username does not exist, it shows “Username does not exist Error”.
- 2) Logout Improvement – Iteration 2
 - a) Preconditions: There is a mistake that we make in iteration 1, which is “logout” is not simply going back to the home page. There should be some methods to monitor the log-in

- status of users and if the user logout, he/she could not go inside into the user interface anymore.
- b) Postconditions: In iteration 2, we introduce **cookies** to our application. It stores the status of users and performs the logout very well. Details are like:
 - i) **Log-in Status:** if the user login successfully, the cookies will store his/her information as session file for 1 hour. The user could refresh the page and it directs to user interface again (will not ask the user to login again). Even if users click “login” button again, it will go to the user interface page directly. After 1 hour, if the user does not perform any further action, he/she will automatically be log-out.
 - ii) **Logout:** For the performance of logout, now it can “actually” log out and remove all temporary information of the users. The users need to enter in information again to log-in and use our application.
 - c) Acceptance Tests:
 - i) If the cookies do not work well, it will stop the query processing and safely go to the original home page to protect the safety of user information.
 - d) Velocity Discussion:
 - i) Ziyi and Chen are responsible for building the cookies and improve this login process. As this functionality is accomplished in 2 days, we agree to assign 10 marks for this task, ~~corresponding by 6 for Ziyi and 4 for Chen.~~
- 3) **Google (JavaScript & Place) Map API – Iteration 1 (Improved a lot in Iteration 2, Iteration 3)**
- a) Preconditions: (Intention)

Google Map API is so much stronger than our group expect. It contains a package with 15 functionalities and could be used in almost every map-based thing such as locating, distance, display nearby stores, etc. For this iteration, we focus on some basic ones:

 - i) The users may want to see their current locations first.
 - ii) The users want to see nearby restaurants and recommendations. Here, users could be in either home page or user page.
 - iii) Additionally, the user may want to adjust the searching radius on the map to do a more specific or broader search on nearby restaurants.
 - iv) The users may want to do a search of some locations. For our application, it is likely to be a restaurant. The users need to know where it is.
 - v) The map should tell the users the shortest route between two locations. This feature could be used widely, such as from school to home or from home to a restaurant.
 - b) Postconditions:
 - i) **Map of SFU:** In the bottom of home page, there is a static map to (~~indicate where the users currently are, and to show nearby recommended restaurants~~) show the location of SFU, which is where our project team based on. It shows the nearby restaurants at SFU (not very useful and basically will not change).

- ii) **Dynamic Map:** In the user page, ~~there is a direction to the map in home page.~~ we add a dynamic map to show where the user currently is, using **Google JavaScript Map API**. It also automatically shows nearby restaurants based on location. Please note that the website will ask the permission of users to locate where they are.
 - iii) **Map Expansion:** Through **Google Place Map API**, if the users want to adjust the map and to see a different scope (for example, from radius 500m to 5000m), the user could input the number that he/she needs and click “confirm”. The map will adjust accordingly and deliver suggested restaurants for that map scope.
 - iv) **Show more restaurants:** we add a column list to show the name of nearby restaurants shown on the map as well. The user could click “show more” button to display more names in the list.
 - v) **Search:** Now, the map supports a functionality of doing search. Users could input locations and the map will automatically show where it is.
 - vi) **Shortest Route:** If users input two locations inside, the map will get their coordinates and calculate the shortest path between these two places. It could definitely help the users to estimate the cost on transportation.
- c) Acceptance tests: (iteration 2) As we use the Google map API, we should test the locating functionality works well.
- i) If users deny the permission of getting locations, the page will direct an error information for the user (basically the functionality of Google map itself).
 - ii) If there are no restaurants find or the user locates to an invalid location, the page will note the user to try again.
 - iii) If users just input one location, without another location to see the route, the map will instruct an “error” to ask the user to fulfill the information.
 - iv) If there is no matching result in searching, the map will show no result and ask users to try again. It will not direct the user to an extremely weird location.
 - v) We designed the other four tests to show errors: **Permission_Denied**, **Position_Unavailable**, **Timeout** and **Unknown_Error**. These four cases basically include all possible errors.
- d) Velocity Discussion:
- i) Ming and Chen are responsible for this important part of our application. As the map functionality is essential and kind of difficult to implement, we agree to assign 16 marks for the map functionality, ~~corresponding to 8 points for each person.~~ This task was continually contributed for about one week.
 - ii) Chen is responsible for these two features. Everyone has contributed in testing of map, as this is one of the most important part of our project. We agree to assign 10 marks for this functionality. Ming also worked with the real-time effort.
- 4) (Minor User Story) User Interface optimization – **Iteration 2**, **Iteration 3**

- a) **Preconditions:** More realistically, the registration page will include more information for users to input, such as gender, age, dining preferences, etc. Accordingly, the back-end database broadens. Meanwhile, there will be more functionalities such as adding reviews, setting dining preferences, etc.
 - i) **Easier Registration:** In iteration 2, we decide to make the registration process easier for users. Nobody wants to spend 5 minutes to fill in personal information and register, after all. Hence, we move the users profile displaying and editing options to the User Interface. In iteration 3, we plan to build databases to store the personal information of users and improve this functionality (editing profile). **(DONE)**
 - ii) In iteration 3, we add the functionality to store the users' information, accordingly broaden the back-end database. As for our application, the most important information is the dining preference and location, we decide not to add more general information such as age, gender, height, etc. Hence, we increase eight more attributes in the database, which are X, Y coordinates, login status and five eating preferences. Also, these types of information are stored in cookies.
 - iii) **Minor improvements:** in the homepage, now the "get started" button has some features, which is no longer a decoration.
- b) **Postconditions:**
 - i) **More Components:** we select a better format of user interface and make it look more beautiful now. There are now more components (for example, an elastic option bar, more check boxes, chart diagram, etc.) in the user interface. It looks more user friendly.
 - ii) **Profile page:** now there is a bar in the user interface to instruct users to store their preferences or change the password. Once there is change and being submitted, refresh the page and it will show the updated information for users.
 - iii) **Users nearby:** as the database stores the users' coordinates and login status, we designed a function to show the online users nearby. If users login in successfully, the database "login-status" will change to TRUE and the function uses his / her coordinates to calculate the distance. It is based on a circle. The user interface dynamically changes according to how many users nearby (in the circle).
 - iv) **Minor improvements:** in the homepage, now if you click the "get started" button, it will guarantee you to the login page. If the cookie detects you are already log-in, it will go to the user interface.
- c) **Acceptance tests:** This is just a structure improvement of the page, basically in HTML and CSS file. Hence there is not many tests actually implemented for this part.
 - i) If the user leaves empty in the change password option, the password will not change, instead of setting the password to be "NULL". Before it will cause the user to have no password which makes the account destroyed.

- ii) If there is an error in connecting to the back-end database and the change is not successfully stored, the page will direct an “error” message to tell the user to try one more time.
 - iii) If the users block the permission of getting the location coordinates, the database will store NULL inside. It will not output an error. Instead, in the “users nearby” it shows “undefined” to users.
 - d) Velocity Discussion:
 - i) Ming, George and Zihan are mainly responsible for the designing of a new user interface. This is the basis display of our application and we agree to assign 20 marks, ~~corresponding to 6 marks for Ming, 6 marks for George, 4 marks for Zihan and rest 2 marks for Ziyi and Chen.~~
 - ii) For iteration 3, George was responsible for modifying the html page and Ziyi worked for the .js function and testing. Chen worked for getting the coordinates and showing the people nearby. We decide to add 5 points for each person and totally 15 points for this feature.
- 5) Make a meal plan && Dealing with conflicts in dining opinions – **Iteration 2 and 3 (future)**
- a) Preconditions: In iteration 2, the main functionalities of our application should be set. Users can make an interaction with others and make meal plans together. When meeting conflicts of dining opinions, our application could give a smartest recommendation for users to choose, based on preferences weight.
 - b) Postconditions: In iteration 2, we put off the implementation of this functionality a little bit, as it is actually harder than our thoughts. For this iteration, our temporary partial improvements are:
 - i) **Gallery:** we add a gallery option bar to show the suggested restaurants. Basically, it is a static one now. For each picture, there is a hyperlink to the homepage of that restaurant. **In iteration 3, we hope to improve this so that it could connect to and automatically recommend the restaurants nearby (become dynamic). (Comment: DONE. Substituted with functionality 7)**
 - c) Acceptance tests: If the home page of the directed restaurant changes, the test will instruct an “error” instead of nothing or a wrong picture. Also, in the administrator page it tells managers to change it in time.
 - d) Velocity Discussion:
 - i) George is mainly for this part. We agree to assign 4 marks for this relatively simple functionality.
- 6) Suggest recommendations (help users to make decision) – **Iteration 2, Iteration 3**

- a) Preconditions: Like the button “I’m feeling lucky” in Google, we decide to add a functionality for users who sometimes feel hard to make choices. The recommendation is related to the users’ dining history and preferences.
- b) Postconditions: Similar with user story (5), this time this functionality is used by the “gallery” bar. It would be improved in iteration 3. (**Comment: Replaced with Food Digger – functionality 7**)
- c) Acceptance tests: There should be a test to make sure this button works well and deliver correct restaurants to users. More meaningful tests should be determined later.

7) Food Digger – Iteration 3

- a) Preconditions: As one of the most important features in our project, the building of “food digger” improved the practicability of our application a lot. This is basically used to help people decide where and what to eat. This feature is based on two APIs: Zomato API and a food recipe API.
 - i) **Restaurants Nearby**: Users may want to have a look on the amazing restaurants nearby. He / she could be unfamiliar with that city.
 - ii) **Cook by Yourself**: Users may feel kind of lazy to go outside. Hence, he / she need a recipe on food to help them cook with ingredients.
- b) Postconditions:
 - i) Once users enter the “Food Digger” bar, he / she will need to make a choice between eating at home (cooking by themselves) or dining out.
 - ii) **Recipe**: If users choose to eat home and show them recipe, he / she could input the food category that they are interested in. The Food Digger will show them different types of food that could be cooked easily.
 - iii) **Recommendations**: If users choose to show nearby restaurants, Food Digger will ask them to input the city that they are in. Basically, based on Zomato API, it contains almost all cities in North America.
 - iv) The users could adjust the number of results to show more restaurants nearby, which give them more alternatives.
- c) Testing:
 - i) If users input a very large number to show, the app will not show that many and instead, only return the first 20 restaurants nearby.
 - ii) If there is an error in loading recipes or restaurants, the page will instruct an “error” to tell users to try again.
- d) Velocity Discussion:
 - i) Zihan is responsible to build this feature. Chen helped with setting this functionality done. Everyone in our group contributed the testing job. Hence, we decide to add 15 points for this feature.

8) Chat room and friend / stranger lists – Iteration 2, Iteration 3 (first real-time feature)

a) Preconditions:

- i) As teenagers and young people are always curious about new things around the world, the users likely may want to talk with strangers. This is the initial intention of the construction of our public chat room. Certainly, this is typically used in almost all current applications.
- ii) For this iteration, we focus on building a public chat room, under the **node.js** and **socket.io** module. To achieve this functionality, we followed two useful tutorials from (http://www.cnblogs.com/Wayou/p/hichat_built_with_nodejs_socket.html) and (<https://itnext.io/build-a-group-chat-app-in-30-lines-using-node-js-15bfe7a2417b>). In iteration 3, it could become a private chat room, with a mature functionality of friend / stranger list. Also, there could be a direct method to add friends with each other through our chat room. (**Comment: Private Chatroom DONE. No friend list**)
- iii) After chat with the public, the users must have a way to go back to our homepage / user interface (if already logged in). In iteration 3, one improvement could be like building the chat room inside the user interface. (**DONE**)

b) Postconditions:

- i) In the user interface, there should be some checkboxes or menu lists for finding existed friends and looking for nearby stranger diners. They could also check preferences or other information with each other, which is more than the stranger level relationship. As well, the blocked users should be filtered automatically. In Iteration 2: we finished such functionalities for the chat room:
- ii) **Enter:** If the user is logged-in, he/she could go to the public chat room through clicking the option bar “chat room”. It directs the user to our chat room and instructs the user to input a valid “nick name”, as users may want to hide their usernames.
- iii) **Time:** The chat room shows when the users send their messages, which orders all messages in the chat room chronologically.
- iv) **System Message:** The chat room will show the welcome message to new users by their nick names. For example, “system (xx:xx:xx): User ‘Bobby’ joins”. It instructs the existed users that someone enters here. Inversely, if someone leaves the chat room, the system automatically shows “system (xx:xx:xx): User ‘Bobby’ left”.
- v) **Online Users:** In the first line, the chat room shows how many users currently online.
- vi) **Set the Color:** The users can set the color of what they want to send.
- vii) **Send Emoji / Picture:** there are different types of valid inputs available. We saved several emojis in advance in the database and offer them for users. The users could also send pictures if they want. In iteration 3, we could add more functionalities such as sending pdf files (as this is a dining app, this may not be very useful). (**Comment: Since this is not very necessary, we did not plan to add this feature**)

- viii) **Quit:** Once the user wants to leave for our application (user interface), he/she could click the button “DiningPal” to go back. The sever will detect the left of users.
 - ix) **Iteration 3:** Now, the public chatroom (basically for us to testing and learning) is set inside the user interface, which is no longer directing to another website.
 - c) Acceptance tests: To test it works well realistically, make two virtual users and test each functionality. Make assertions to indicate that something goes wrong. The tests we implement in iteration 2 (in hichat.js):
 - i) **Same Name:** If the user wants to make a same nick names which conflicts, the system will show “nick name existed” and tell users to try again.
 - ii) **Success of login/disconnect:** under socket module, there are tests for checking the success of connecting to the server like emit “loginSuccess” and “logout”.
 - iii) **PostMsg:** This test could make sure the message has been sent out.
 - iv) In iteration 3, there should be more tests to verify that two users are friends / strangers before the delivery of messages. Additionally, we could make sure that the chat room is available to our DiningPal users only (not officially public to everyone).
(Comment: Move to Private Chatroom)
 - d) Velocity Discussion:
 - i) Zihan is mainly responsible for whole chat room itself. We agree to assign 6 marks for this functionality. This is accomplished by learning from the tutorial for about 2 days.
 - ii) As to make the public chatroom embed is kind of an easy job, we just agree to assign 2 marks for this work, as an encouragement.
- 9) Private Chatroom (based on MongoDB) – Iteration 3 (second real-time feature)
- a) Preconditions: As this is a kind of important feature that we added, we plan to extract this specifically and talk about its functionalities. To achieve the functionality of one-to-one chatroom, there must be a database to store the information that users share. We could not use socket.io alone to transfer information between users. Hence, we learned a tutorial online, chose the MongoDB as our back-end database and created one application on Heroku separately. The core code is worth around 170MB, which could not be submitted with our DiningPal app together. The code has been uploaded separately.
 - i) The users may want to find people with similar flavors. There should be a “channel” to specify which dining category the room is serving for.
 - ii) The users may want to talk with someone directly, not in public. Also, the app should tell the user whether your friend is online or not.
 - b) Postconditions:
 - i) In the user interface, if the user clicks on the “private chatroom” inside the “component” bar, it will go to the private chatroom inside. To protect the privacy, it

may need the user to register again. Just like the operation pattern of Facebook and Messenger, we divide our app and chatroom separately to make it safer.

- ii) The cookie will store the users' information, if he / she has already registered accounts for this chatroom. They could login directly.
- iii) **Create a Room:** In the top left spot, the user could modify the "home" to change the whole chatroom to another theme. It creates a totally new branch.
- iv) **Create a Channel:** In the left bars, the user could add a new channel (basically, eating flavors) to wait users with similar hobbies to go in. Then they could chat with each other and decide where to eat together.
- v) **Talk to Someone Privately:** Down the channel bar, there shows how many registered users there are and how many users are actually online currently. The users could send messages to his / her friends directly.
- vi) **Delete Message:** The users could delete what he / she said in this chatroom. Then the information will no longer be stored in the Mongo Database.
- c) Testing: There are some simple tests to make sure the private chatroom works well
 - i) If the users just create a channel and do not send any information, the channel will be destroyed automatically, to avoid someone created channels illegally.
 - ii) Similarly, if the users do not send any information to someone else, the "friend connection" will not be built and nothing is stored.
- d) Velocity Discussion:
 - i) Zihan is basically responsible for learning the Socket and MongoDB tutorials as well as build the whole chatroom, we decide to allocate 10 marks for this essential functionality. Also, Ming worked this together with Zihan to write Socket functions.

10) OpenWeather API (In user interface - Dashboard)

- a) Preconditions: As the users' cultural plan is highly connected with the weather conditions, here the weather API is used to show the current weather conditions, based on the users current location.
- b) Postconditions:
 - i) Once the users allow to locate themselves, the weather API will automatically work and show these information: location of the user (i.e. Burnaby, BC), current degree and a description of weather conditions (i.e. sunny, cloudy, etc.).
 - ii) Actually, this weather API provides much more information than we need. In iteration 3, we could make use of this API and get more information. (**Comment: We just keep it the same in the user interface, like Iteration 2**)
- c) Acceptance tests:
 - i) If the user blocks the permission of getting locations, the Weather API will not work correspondingly and leave there a blank.

- ii) There is a test to verify that the information we get from the OpenWeather API is valid. If there is any invalid input, it would direct to the error page.
 - d) Velocity Discussion:
 - i) Ziyi and Ming worked for this functionality. We agree to assign 6 marks with 3 marks for each person of this relatively simple improvement.
- 11) Static Calendar (User Interface: Components - Calendar)
- a) Preconditions:
 - i) For regular users: there could be a calendar to mark the dining schedule.
 - ii) For administrators: use a calendar to show the working / improvement process
 - b) Postconditions:
 - i) There is a static calendar in the option bar. For **users**, users can use that and drag what and where they want to eat dynamically into the calendar. The selected bar will be duplicated and remain in the calendar
 - ii) For **administrators**, we could use that to trace the development of our application. The operations are basically similar.
 - iii) **Iteration 3: In the next iteration, we could create a database and realistically store what we want to add into the calendar into the database. Hence, this feature becomes part of the user's "profile". (Comment: Not yet finished)**
 - c) Acceptance Tests:
 - i) As it is just a simple static calendar at present, there would be some tests to verify that the blog that we drag into the calendar is actually added into the database. If any error (for example, does not match with database) gets caught, it would go to the error page. **(Comment: Still leave the calendar as static. It is kind of annoying to connect to database)**
 - d) Velocity Discussion:
 - i) As this is basically included inside the template, we won't include mark for anyone as our improvement is based on the content.
- 12) Brief overview for future minor user stories – **Iteration 3 (future)**
- a) Usage of API: increasing more APIs from Twitter, Yelp... **(DONE)**
 - b) Link to further leisure activities after a meal, such as fitness, net bars and libraries. **(DONE, could link to some popular websites)**
 - c) Edit, delete the account or friend lists. Create, edit and delete the reviews. **(REMOVED)**
 - d) Improvements of map API: calculate and show the shortest route for matched users to a restaurant. It may use complex algorithm as there are multiple places to connect in a map. **(Generally DONE)**
 - e) To add more in later iterations...

User Stories Simulation – Administrators

Administrators are special users who are responsible to maintain, monitor and update the application. They have special user-id and home pages to manage the whole system. Here, administrators including our group members. Certainly, the instructor and TAs can also ask permission to have the access.

- 1) Login to Administrative Page – Iteration 1
 - a) Preconditions: The administrators want to do some operations to manage the users (look / modify information / delete). Consequently, login is the first step. The system state is the initial page and actually looks the same as regular users' one.
 - b) Postconditions: If the login id and password match, it automatically goes to administrative webpage, which is invisible to regular users. It should look like a normal management page (probably contains a table, and methods button like “show”, “change”, “delete”).
 - c) Acceptance Tests: There must be a test to verify that the accounts information has been successfully transferred and matched to back-end database.
 - i) Naturally, if the username and password do not match, it won't go to the management page.
 - ii) If any error got caught, assert an error page.
- 2) Add / Modify / Delete / Block Users Account Information – [Iteration 1 and 2 \(future\)](#)
 - a) Preconditions: The user has successfully logged in as administrator and wants to do some operations on user information. The page is in management page.
 - b) Postconditions: The administrator could manually add a new user's information into the system. This action could be done in a query interaction to database. Simultaneously, in this adding step the administrator could set different priorities for users (for example, VIP users). It is also possible to change the information if any errors. Finally, if the administrator wants to block some illegal users, blocking or deleting are two options that could be considered.
 - i) [Iteration 2: Similar with the improvement of user interface user story that we finish this time, the user face of administrators has been improved as well. In the administrator page, there is one more bar “user table” to show how many users already in our system. Certainly, this page is visible to only administrators and not included in the regular users' page.](#)
 - ii) [Also, the users table contain the information whether the user is an administrator or not. In iteration 3, we decide to finish the functionality of adding / delete a user \(either a regular user or administrator\) in the administrator page.](#)

- iii) Modify the users: In iteration 3, we are also planning to finish a simple functionality of modification. This task has been started and could be accomplished soon, maybe in this week. **(Forecasted in Iteration 2)**
 - iv) **Delete users**: In iteration 3: now there is an option for administrators to delete users through the 'User Database' bar. To protect the privacy information, we delete the password shown in the table, as passwords could be available through the local database in an extreme situation.
 - c) Acceptance Tests (Iteration 2): The tests could be like checking whether there are duplicated users in the database (By declaring primary keys in the table query). Similarly, there should be tests to check that our operations work successfully. It could be like we assert the user does not have access into this application anymore.
 - i) Iteration 2: we set a test to verify that the table shows well in administrator page. The information must fit the table we designed successfully. If anything goes wrong, it instructs administrators there is an invalid input.
 - ii) Iteration 3: we set a test to verify that the login status is actually the administrator. Regular users could not simply type "/database.html" to go inside the administrator's page. If the cookies detect an illegal request, it will direct to the homepage and remove the account.
 - iii) If there is an error when connecting to the database, the website will instruct an "deleting error" to tell the administrator to try again.
 - d) Velocity Discussion:
 - i) Iteration 2: George and Ziyi are responsible for this work and we agree to assign 3 marks for each person. This task is finished in one day.
 - ii) Iteration 3: Ziyi takes the functionality to delete users and write the test to check the legal status of users. We decide to assign 3 marks for this improvement.
- 3) **Manage Financial Information / statistics – Iteration 3 (future)**
- a) Precondition: If the administrator has the interest in looking at some financial statistics, for example, users' preferences, activity levels and login history, you could log in into the management page.
 - b) Postconditions: The system will run some cookies (store the user information) / SQL queries to show the statistics. Likely they will be in diagrams.
 - c) Acceptance Tests: This should be a simple test to verify that the statistics are within the domain. **(Comment: REMOVED, as we actually do not have any statistics to manage)**

User Interface and Scope of Project (Iteration 1, 2, 3)

Main features for users include:

1. User login/sign up (depending on different user identities, show different features)
2. Locate users on a map and see nearby restaurants in a scope, show the shortest route
3. Match up users who are willing to dine together at a specific time range
 - 3.1 Users individually choose a(some) restaurant(s) they prefer and time for which they are available
 - 3.2 They could also choose the recommendations generated from our application, based on their dining history and preferences.

Some other accessible functionalities:

4. Friend with other users can check their preferences (Chat Room, both public and private)
5. Recommend restaurants to users in terms of their preferences and locations
6. Search and present coupons from target restaurants
7. Calculate and show the shortest route for matched users to a restaurant
8. Plan further leisure activities for users after the meal
9. Help user cook at home by recommending them recipes

Velocity and Scope Discussion:

For each story we finished (or, partially finished) in Iteration 2, we assign a relative reasonable mark for that work, based on the difficulty and time scope. Also, we included who is responsible for which. The final statistics is as follows:

- **Ziyi:** $6 + 2 + 3 + 3 + 3$ (for testing) = 17 marks
- **George:** $6 + 4 + 3 + 3 = 16$ marks
- **Chen:** $4 + 8 + 3 = 15$ marks
- **Ming:** $8 + 6 + 3 = 17$ marks
- **Zihan:** $4 + 6 + 3 = 13$ marks (Comment: No need to assign marks for each person)

Final Velocity (39 points / week):

- Total Stories Finished: 10 main stories (Make up by 27 minor stories)
- For week 9 (July 2 – July 9): 27 points
- For week 10 (July 10 – July 16): 51 points

Expected Velocity for Iteration 3:

- 40 – 50 points for each week with continuous contributions
- Finish or improve at least 15 main stories
- Each group member contributes similarly

Real & Final Velocity for Iteration 3:

- Total Stories Finished: 8 main stories (33 minor stories, include improvements)
- For week 11 (July 17 – July 23): 36 points
- For week 12 (July 24 – July 30): 40 points
- **Average Velocity:** 38 points / week

Analysis:

- **Good Effort Distribution:** From the productivity measure, a good thing is like for each group member, we are contributing kind of similar work to our whole group project. Although some people may not feel good at programming, at least he could finish some functionalities by himself through learning tutorials or asking for help.
- **Not Frequently Push:** Basically, we just finish our main functionality locally, test it works well and go beyond to the next. That is one of the reasons there are not that many “git push / commit” in GitHub. Also, we just push them to our own branch, not the master branch. As Bobby suggests, after accomplishing each functionality, it is better to push them immediately so that we could track the whole progress.
- **Could Improve in Scope:** One disadvantage that we should focus on is like, almost 2/3 work is done by the second due week. We contributed less work one the first week. In iteration 3 we should continuously contribute to the whole project and not accumulate. However, each group member has realized it and we agree to have another meeting by Friday to finish the databases design.
- **Much Better in Scope:** this time, we focus on doing projects together gradually a lot. We improved the database just a few days after iteration 2, finished some features during week 11 and finalized our application on week 12. It also gets verified from the commit history in GitHub. Proof by facts, this is really efficient as we do not need to rush up anymore. It leaves us more time to discover different functionalities.
- **More tests added:** through what we learned from this class and the presentation through Mr. Vincent Chu, we realized the importance of testing especially for a software or application. This time we paid much attention to every functionality, made sure they all work well and caught unpredicted errors by testing.
- **Good Effort Contribution Continued:** For each group member, we continue the good feature from last time, which is like contributing similar work to the whole project. Although some people feel stressed in the final week, they contributed in advance and got a free time later. The atmosphere in our group is consistently positive.
- **Shortage and Pity:** Although we have finished some features that we thought were really difficult, we still leave some functionalities incomplete and finally choose to give up, such as matching users in real-time on a map and add friend lists. That is kind of a pity. From the demo time, some groups achieved these features. Probably if we have more

time or make more effort, we could have rocked this. After all, for some features we were already very closed, just with some minor bugs not fixed.

Technologies Used:

- **Google Map API** – generally for the map
- **Zomato API** – for “Food Digger” recommendations of restaurants
- **Open Weather API** – for showing the local weather in the user interface
- **Edamam API** – for “Food Digger” showing the recipes
- **MongoDB** – for the back-end database of private chatroom (Project ID: 5d3e3cfdff7a25581eef49b1)
- **HTML, CSS, JavaScript, jQuery, Socket.io, etc.**

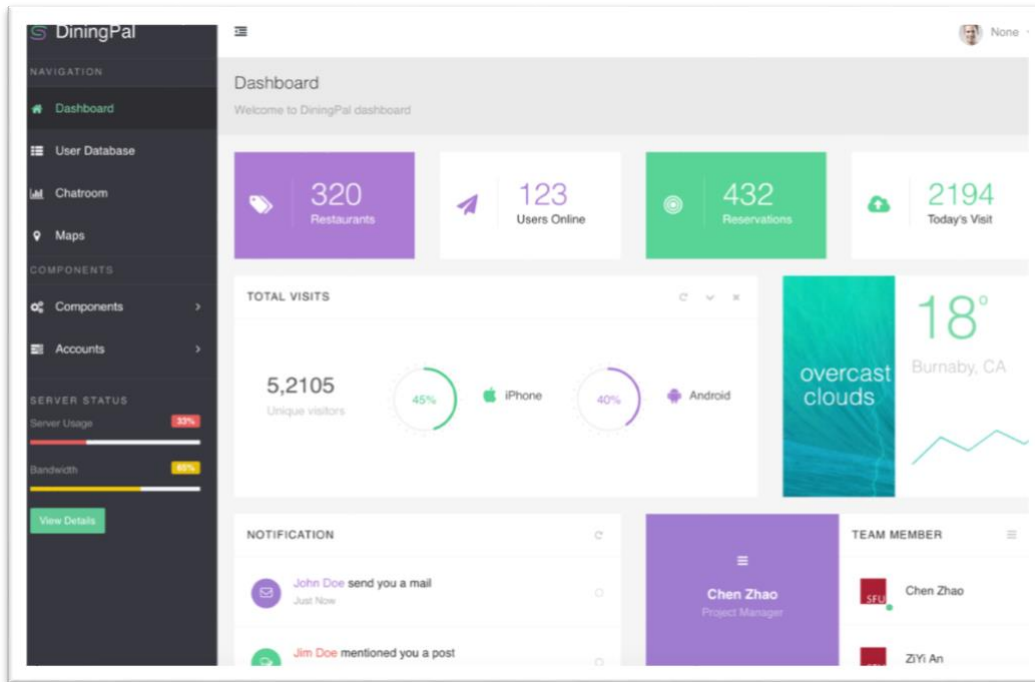
Feelings from the Group:

From this project experience, we definitely learned a lot. Not only technical knowledge like coding with different languages, but how to corporate with others and manage you time into a good scope. Testing is definitely an essential step for building an application. Although there are pities that we have not had a chance to fix, overall we feel proud of what we achieved --- DiningPal. We sincerely hope through DiningPal, not only friends but also strangers could make a meal together. We believe that the distance between people could be closer.

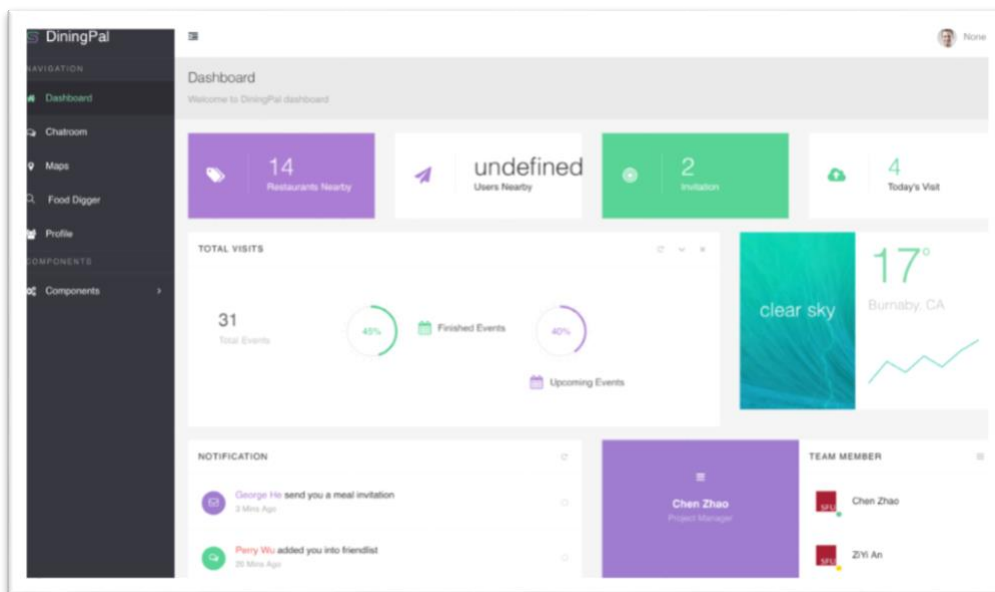
Appendices

Appendix A: Administrator account: cza94@sfu.ca // Password: segfault

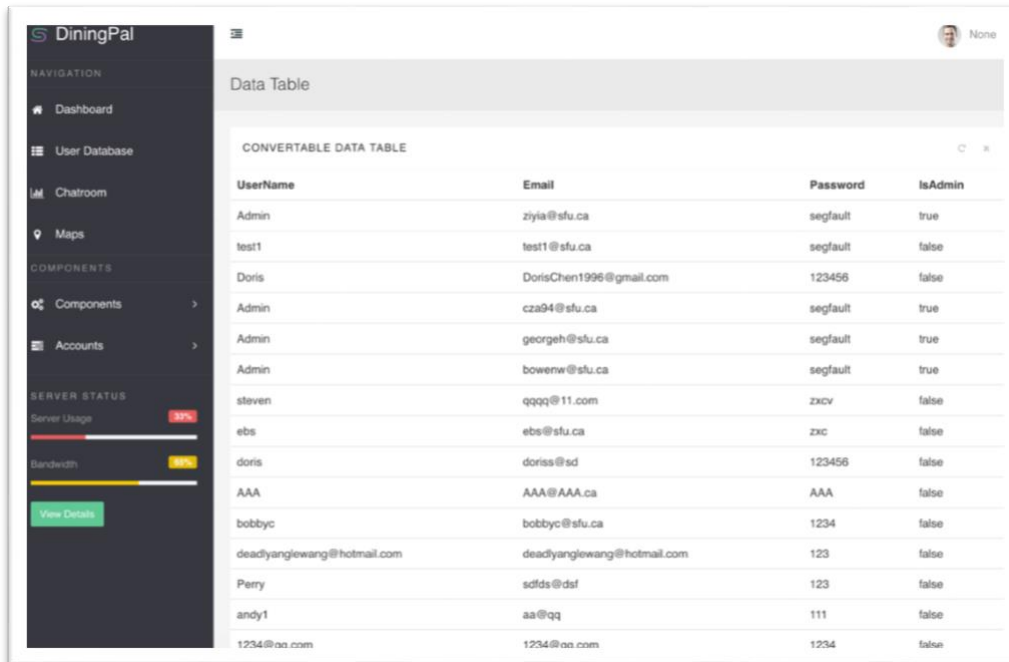
Appendix B: User Interface - administrator



Appendix C: User Interface – Regular users



Appendix D: Users Table – Administrators Page



The screenshot displays the DiningPal application interface. On the left is a dark sidebar with navigation links: Dashboard, User Database, Chatroom, and Maps. Below these are 'COMPONENTS' (Components, Accounts) and 'SERVER STATUS' (Server Usage at 30%, Bandwidth at 80%). The main area shows a 'Data Table' component with a 'CONVERTABLE DATA TABLE' header. The table has four columns: Username, Email, Password, and IsAdmin. It contains 15 rows of user data.

Username	Email	Password	IsAdmin
Admin	ziyia@sfu.ca	segfault	true
test1	test1@sfu.ca	segfault	false
Doris	DorisChen1996@gmail.com	123456	false
Admin	cza94@sfu.ca	segfault	true
Admin	georgeh@sfu.ca	segfault	true
Admin	bowenw@sfu.ca	segfault	true
steven	qqqq@11.com	zxcv	false
ebs	ebs@sfu.ca	zxc	false
doris	doriss@sd	123456	false
AAA	AAA@AAA.ca	AAA	false
bobbyc	bobbyc@sfu.ca	1234	false
deadylanglewang@hotmail.com	deadylanglewang@hotmail.com	123	false
Perry	sdfds@dsf	123	false
andy1	aa@qq	111	false
1234@qq.com	1234@qq.com	1234	false

Appendix E: Tutorial for “Food Digger”

Reference: <https://github.com/rposner16/Eat-Well>

Appendix F: Tutorial for the Public Chatroom:

Reference: http://www.cnblogs.com/Wayou/p/hichat_built_with_nodejs_socket.html

Appendix G: Tutorial for the Private Chatroom:

Reference: Made by Adam King, Copyright © 2016, licensed under the [MIT License]
<https://github.com/adamjking3/React-GraphQL-Subscriptions-Starter/blob/master/LICENSE>