Application Implementation Document

for

Fernwood Farmers' Market

Booth Scheduling System

or

FaBS

Table of Contents

1.0 Major Components

1.1 Registration:

Registration is essentially a form for the user to fill in. It validates the form through some some error checking regarding to provided input, and then creates the user. If there is an error, it provides an error message to the user, and they try to fix it. If registration is successful, users are redirected to the login page. The views for this section is under “views/pages/registration.html”. Registration also uses “userCreateController”, which is found in “js/controllers/userCtrl.js”. The controller calls the “js/services/userService.js”, which calls the API located at “server/routes/api.js” which in turn calls the user model present in “server/models/user.js”.

All HTML pages, controllers, services and models can be found in the same folder structure as mentioned above.

1.2 Login:

Login takes the user’s username and password, and checks if it exists within the “users” collection. if so, it will create a token for them, which holds their MongoDB object ID in the payload, and is used on the market page to extract user information. The login view for this page would be home.html, and the process uses “userLoginController” within “userCtrl.js”. This checks the token with the “AuthService.js” and if successful, users are redirected to “markethome.html”.

1.3 Book Booth:

Booking a booth is done on the “markethome.html” page. It uses the “marketController” in “mainCtrl.js”. That controller calls the “boothService.create()” function, which goes through the api to create a new booth with “booth.js” model. In the model, there is a pre save hook, which also loads itself into the “bookedbooths” array of the user booking the booth. As well as into the array of booths in the day model for its date.

1.4 Cancel Booth:

Canceling a booth is also done on the “markethome.html” page. it uses the “mainController” in “mainCtrl.js”. That controller calls the “boothService.delete()” function, which goes through the api to delete a booth from the booths collection. In booth model, there is a pre remove hook, which removes itself from the user’s “bookedbooths” array, and the array of booths from day model for its corresponding date.

1.5 Update User Profile:

Updating a user’s profile is done on the “profile.html” page. To get here, click on the Account button from markethome. It uses the “profileController” in “profileCtrl.js”. This controller is responsible for inputting pre existing data to the web page by the use of promises. The controller also checks to make sure all fields follow the same criteria as for registration. After validating the input fields, the controller makes a call to the “userService.js” file which essentially relays the information obtained from the “profileController” to the api.js file. “userService.js” then calls the .put function which updates the information on mongolab.

2.0 Design Patterns and Sequence Diagrams

2.1 The Design Patterns

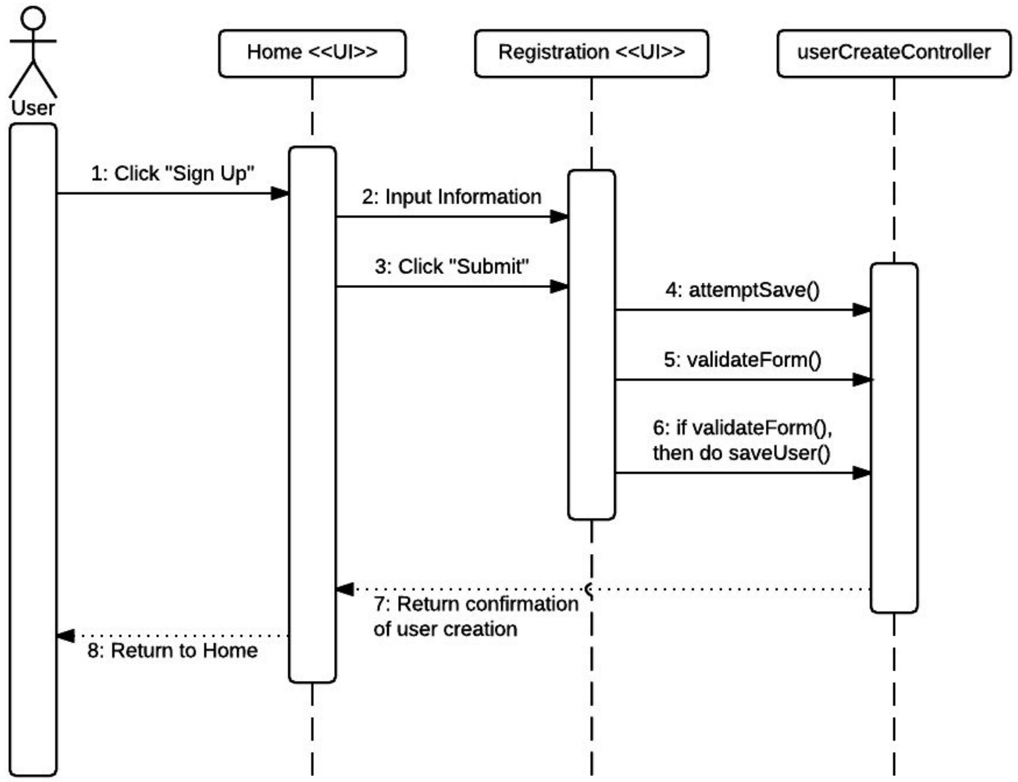
Our system primarily intends to use the Observer and Singleton design patterns.

The *Singleton pattern* can be associated with respect to our five models - Booth, Day, Support Request, and User, as there are only one instantiation of each model. These schemas can be viewed in their respective .js files in "./server/models”. Each Day object contains multiple Booth objects, and the User model consists of multiple Vendor objects which contain various information regarding user account details, biography and booking information. The Support Request model consists of a list of all support request objects that are created by vendors themselves. Since instantiations of these models represent all stored data on the system, we only intend to retain one set of data, thereby indicating to use the singleton design pattern to ensure only one instance of day, booth, user and support request models are created. However, due the behaviour of Mongoose, and the way JavaScript works, strict class instantiations of each model with their individual getInstance() functions have not been implemented, resulting in a rather incomplete textbook implementation of the Singleton design pattern.

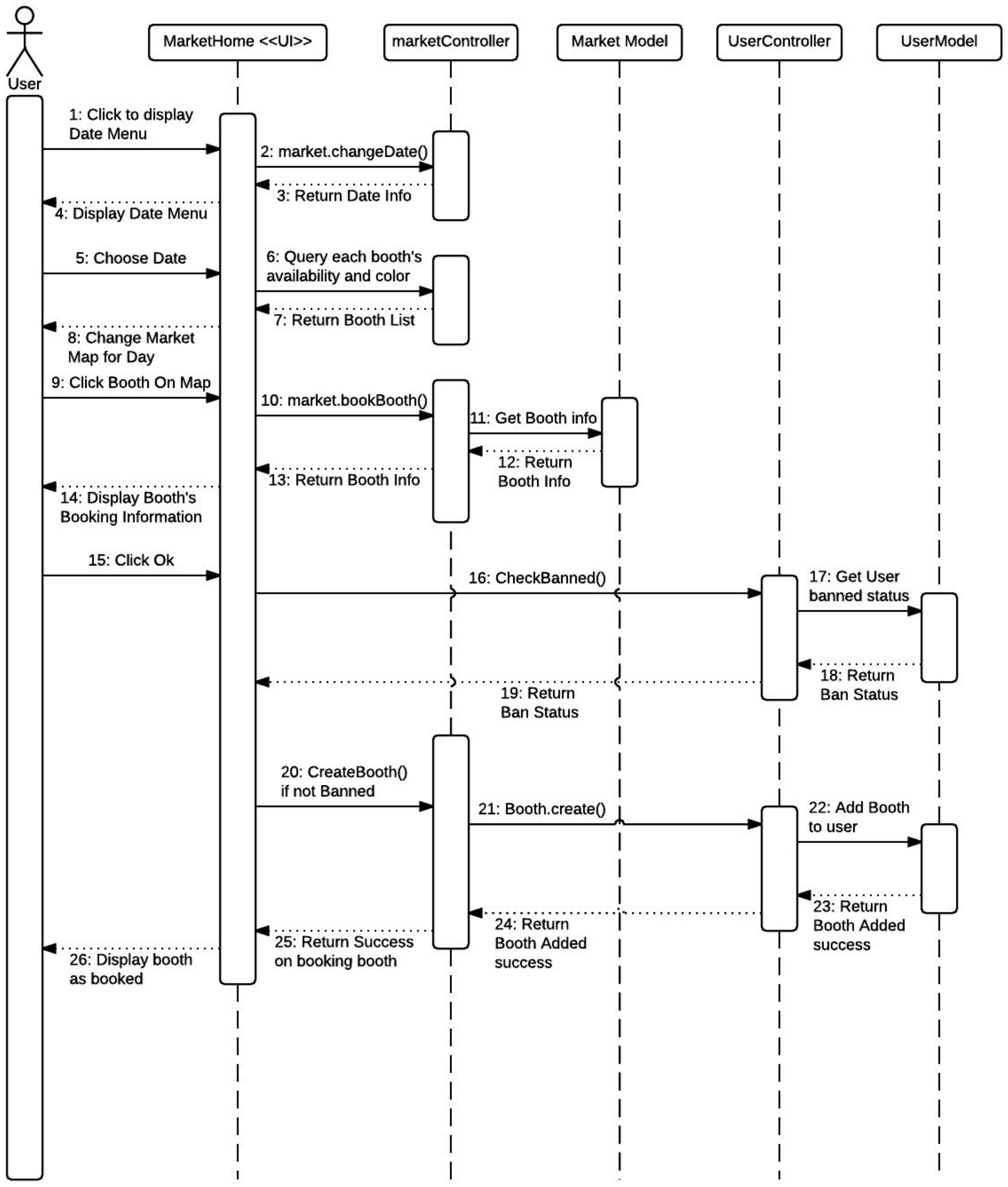
Effects of the *Observer design pattern* can be seen wherever the Angular directive “ng-model” has been used. Within adminmarket.html and markethome.html, the observer(vendor/admin) views of the market schedule are updated at real time whenever changes are made to the booth model due to cancellations and bookings of booths. A similar case can be seen with the ng-repeat directive by updating the views of the user bookings list and the support requests list whenever changes to their respective data models are made. Again, strict implementations of a subject class with a notifyObserver() method and a list of its concrete observers along with concrete observer classes with their respective notify() methods cannot be found within the code; again, due to the implementation restraints of JavaScript itself along with the inclusion of AngularJS as a front-end framework. Much of the functionality of the Observer pattern are provided by AngularJS with it’s MVC like application structure and most importantly, it’s two-way data binding functionality.

2.2 The Sequence Diagrams

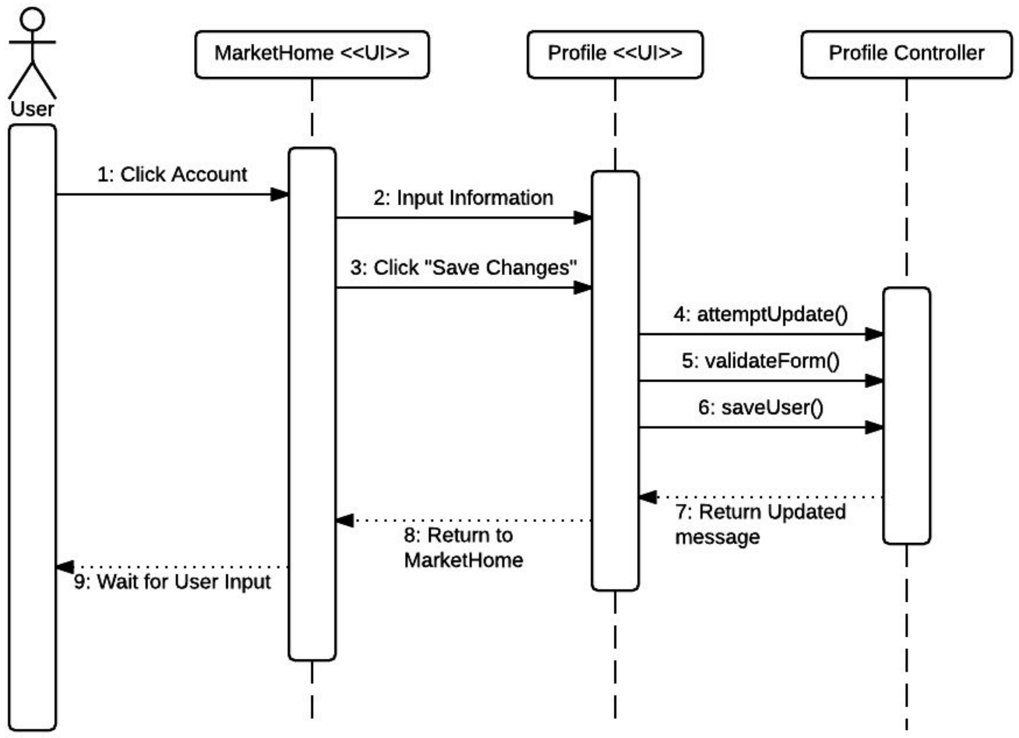
2.2.1 New Account



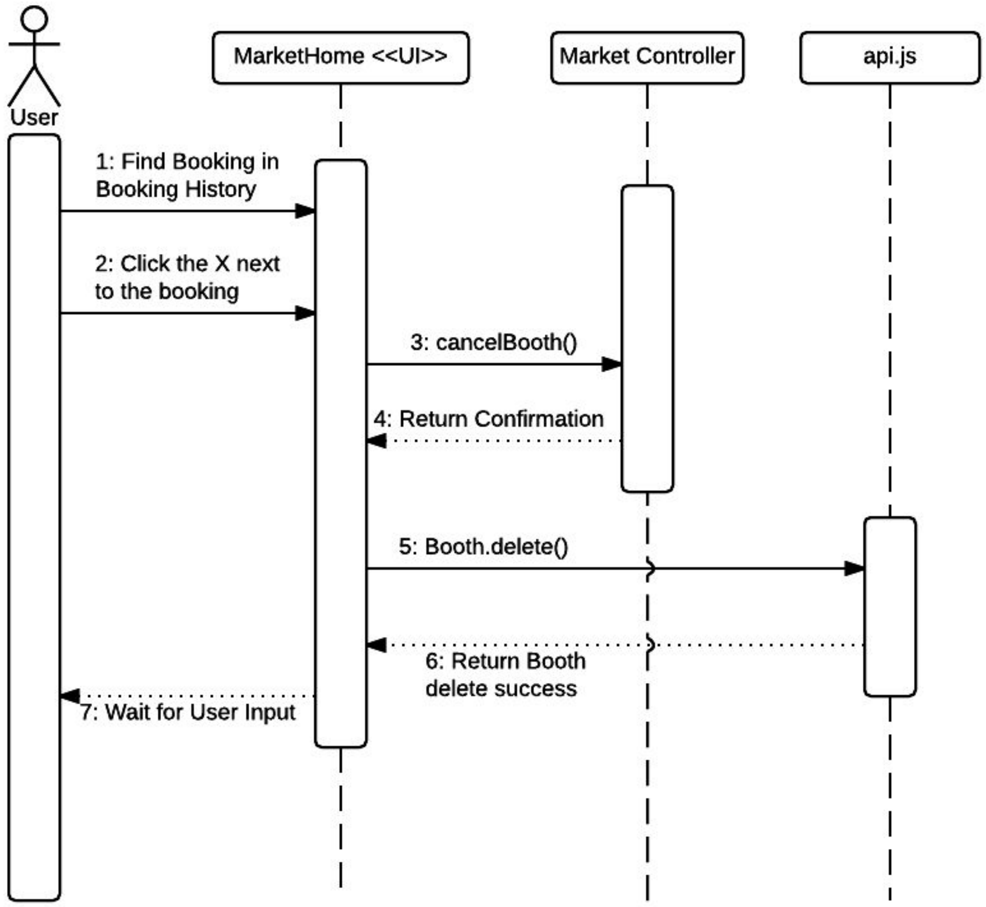
2.2.2 Book a Booth



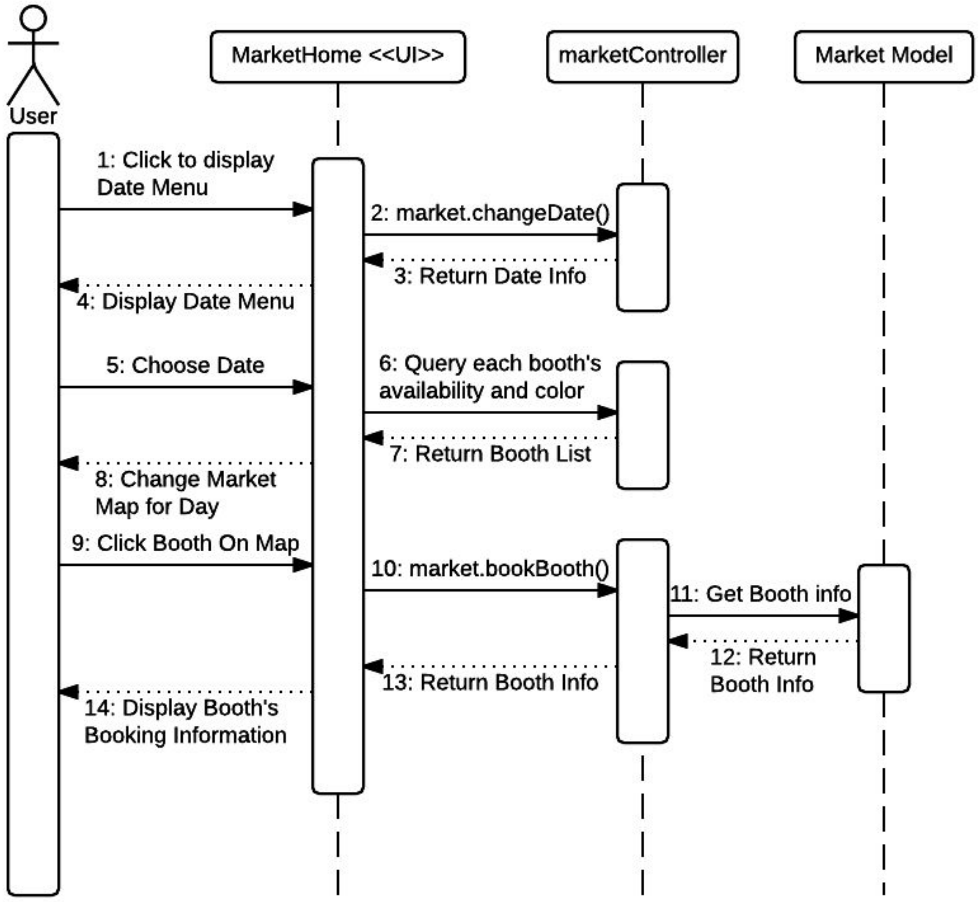
2.2.3 Edit Profile



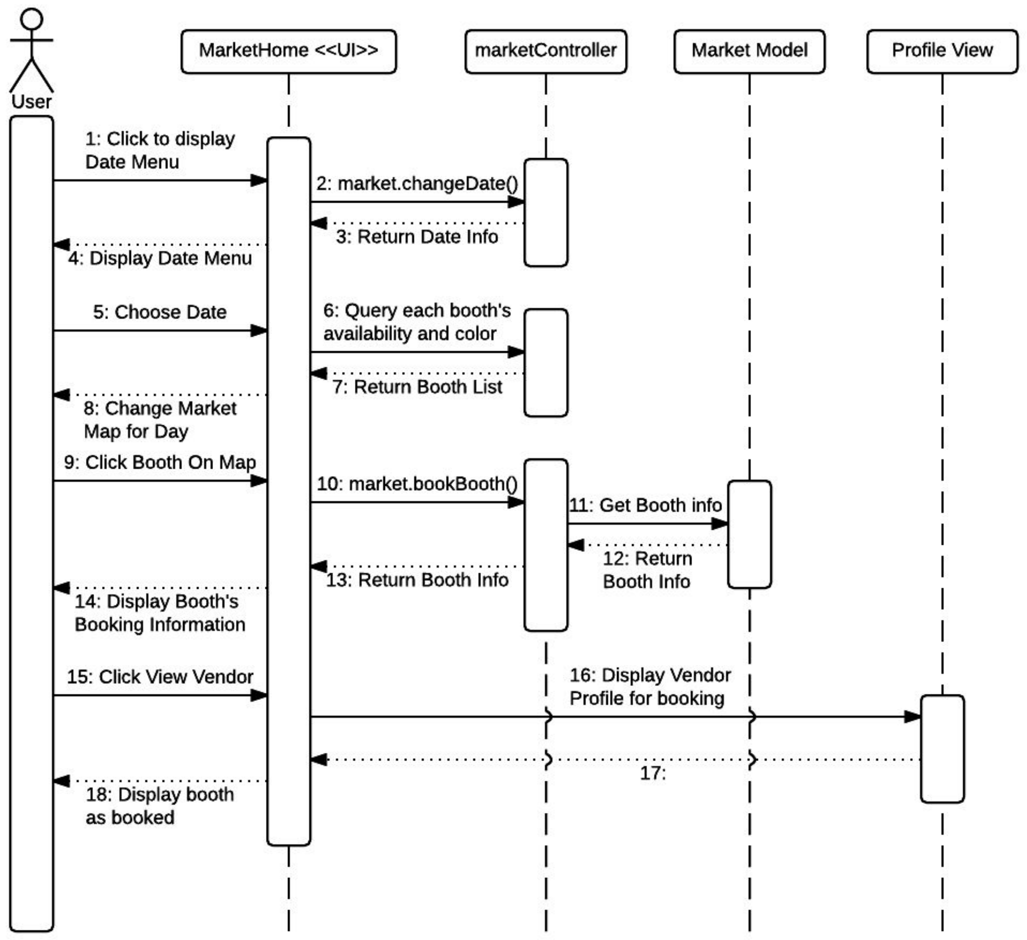
2.2.4 Cancel Booking



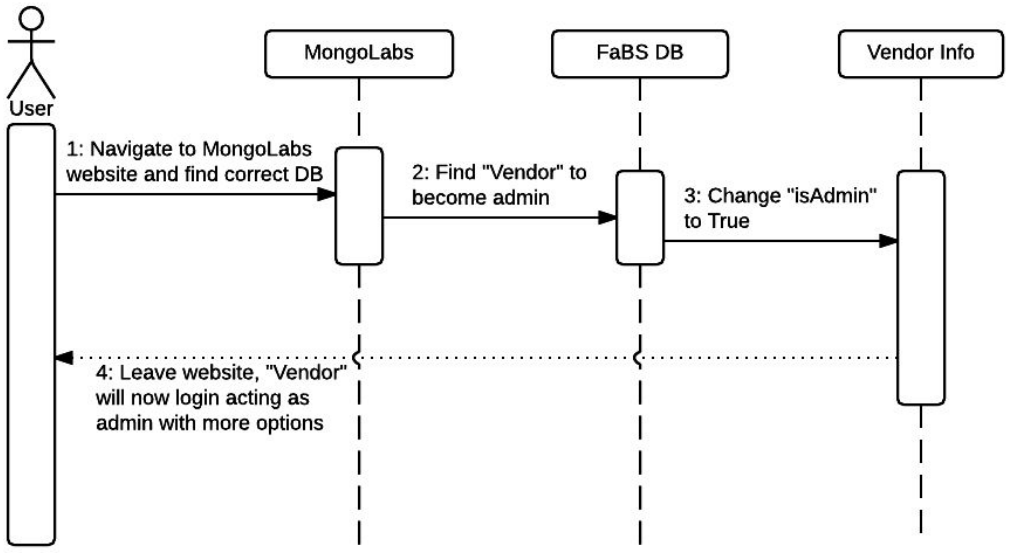
2.2.5 View Market Schedule



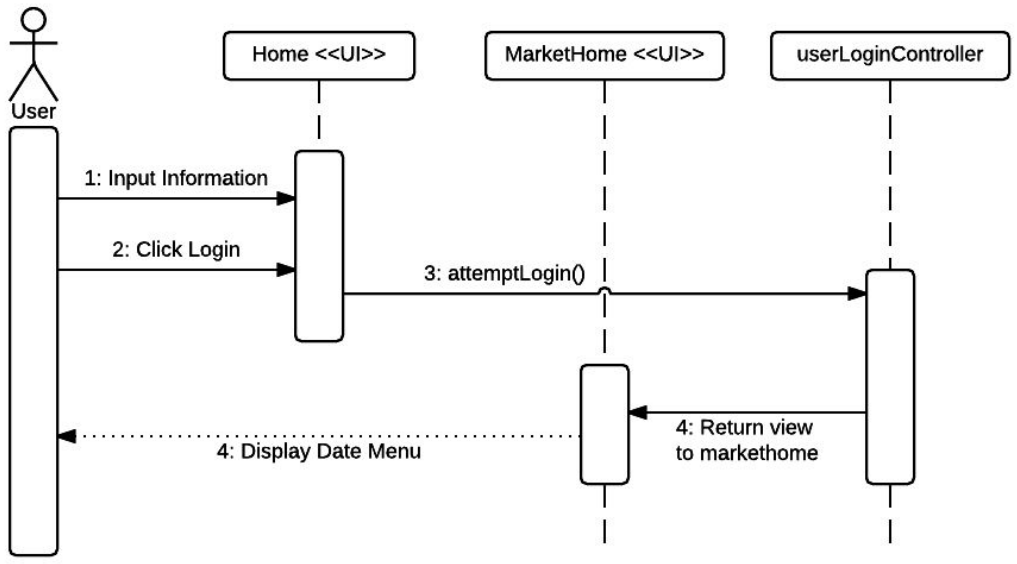
2.2.6 View Booth Information



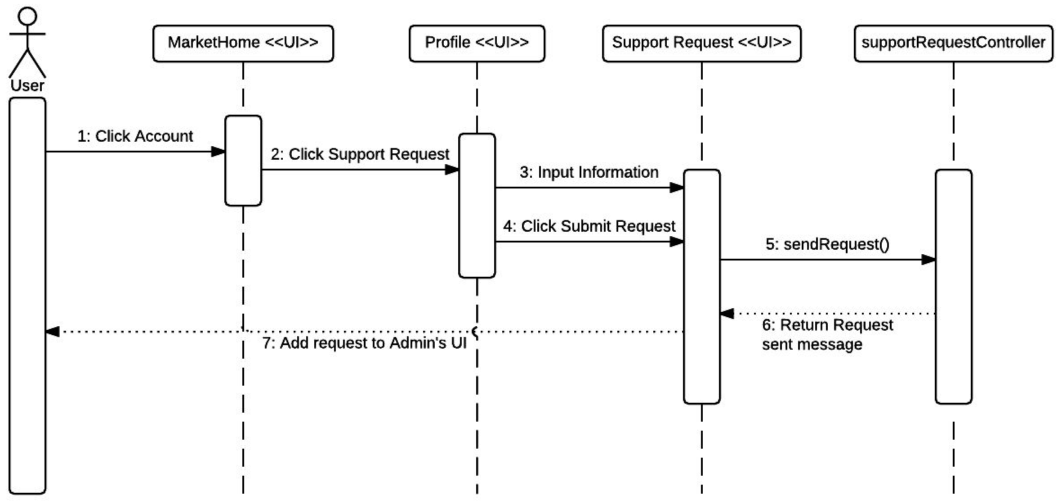
2.2.7 System Operator Creates Administrator


User
MongoLabs
FaBS DB
Vendor Info
1: Navigate to MongoLabs website and find correct DB
2: Find "Vendor" to become admin
3: Change "isAdmin" to True
4: Leave website, "Vendor" will now login acting as admin with more options

2.2.8 Login to the System



2.2.9 Submit a Support Request



3.0 Test Plans

Use case 1: New users register for an account

Destructive test: For all of these inputs, an error message should be presented and a should not be created.

1. Try providing incorrect values for each of the fields / leaving them blank.
2. Try providing mismatching passwords.
3. Try providing a username that already exists.

Expected case: A modal should pop up congratulating the new user and redirect him to the login page.

1. Fill up the form with valid information.
2. Click the submit button.

Use case 2: User books a booth for the market

Destructive test: Multiple users trying to book a booth for the same day at the same time. An error message should be prompted to the user.

1. Two users try clicking on an available booth to book it. One user clicks “confirm” to book it. The second user tries clicking “confirm” to book the same booth.

Expected case: User is given a confirmation pop up for successful booking, and then see’s their booking list updated.

1. Using the calendar, select a date.
2. Choose a booth to book and click on the corresponding time slot.
3. Confirm booking.
4. Receive confirmation message.

Use Case 3: User edits profile details

Destructive test: An error message should come up when invalid profile information is entered(eg. username is too short)

1. Try entering a username that has less than 4 characters.
2. Try entering a password that has less than 5 characters.
3. Try entering a password different from confirm password.
4. Try removing all information in the “Bio” field.
5. Try entering an email without a “@” symbol.
6. Try entering a phone number that has less than 10 digits.

Expected case: User makes changes to their profile details in their respective text fields. After saving, their changed profile details should be updated within the database.

1. Click on the “Account” button.
2. Make changes to text in the profile information/text fields
   2.1. Click on the “Show Additional Details” button to edit further details.
3. Click on the “Save Changes” button.
4. Receive confirmation of update.

Use Case 4: User cancels a booked time slot

Expected case: User is given a warning to make sure that the time slot for the booth to be cancelled is not within 24 hours of current time. Otherwise, a 48 hour booking ban is enforced on the user. If confirmed, a confirmation of cancellation is given.

1. Click on the x button to the right of a booking within the user bookings list.
2. Confirm cancellation.

Use Case 5: User views market schedule

Expected case: User logs into their account, and is shown the current day’s schedule. They can then change the date on the calendar to view the market schedule for a different day.

1. User logs in and is directed to the markethome page.
2. User uses the calendar to change date and view market schedule for another day.
3. User is displayed with the market schedule for the displayed date.

Use Case 6: User views a booth’s information

Expected case: User clicks on the “View Vendors” link within a booth marked red and is redirected to the profile page for the vendor currently booking the booth for that particular time-slot.

1. Click on “View Vendor” link on a booked booth(a booth coloured red).
2. A new page will appear showing information about the user that booked the booth.

Use Case 7: System operator creates a new administrator account

Expected case: User is given access to the mongolab account. They are able to create a new boolean variable called isAdmin which they set to true.

1. Login to mongolab with the system operator account (username:seng299g11 password:market123)
2. Click on Vendors under Collections
3. Click a user that you wish to turn into an administrator
4. Create new variable: "isAdmin": true,
5. Click Save or Save and go back

Use Case 8: Login to the system

Destructive test: error messages should be generated when an unregistered user is entered into the username field, and when an incorrect password is entered into the password field.

1. Try leaving the log-in fields blank.
2. Try entering a username that has not been registered.
3. Try entering a password that does not correspond to the entered user.

Expected case: No error messages pop up, user is redirected to markethome.html.

1. Enter username into Username text field.
2. Enter password into Password text field.
   2.1 (optional) check the “keep me signed in” box.
3. Click the “LOGIN” button.

Use Case 9: Submit Support Request to admins

Destructive test: Submitting empty requests should not be sent. Error messages should be prompted when this occurs

1. Try leaving subject field blank
2. Try leaving request field blank

Expected case: No error messages pop up, user inputs their request and request is saved into database.

1. Click on Account
2. Click on Support Request
3. Enter Subject field with text
4. Enter Request field with text
5. Click Submit Request