SW Engineering CSC 648/848 [2] - Fall 2016
"myPlace" Web Application
Team 14 *(Local)*

Sophia Amin - samin25@mail.sfsu.edu
Ilya Ivanenko
Daniel Nguyen
Patrick Aung
Jimmy He
Ulises Martinez

"Milestone 4 Beta Launch"

December 1, 2016

Submission Date	**History**
10/06/16	M1 First Revision
10/11/16	Recommended Revision
10/25/16	M2 First Revision
11/1/16	Recommended Revision 2
11/25/16	M3 Meeting Feedback
12/1/16	M4 First Revision

Product Summary

myPlace is an apartment rental web application that is built by San Francisco University (SFSU) students, for SFSU students. *myPlace* leverages itself over its competitors by limiting its rental services to SFSU students only. *myPlace* will include basic features such as search, filters, maps, apartment listing, and communication. However we highlight these features by basing the design similar to official SFSU websites; thereby creating a welcoming and native feel familiar to SFSU students as well as an extension of SFSU's online identity. By emphasizing simplicity and intuitive design, we can serve clients of all ages and focus on delivering the one desire of our SFSU students: finding a place to rent.

Guests and students shall be able to view apartment listings provided by landlords and filter results to better find exactly what they are looking for. After finding an appropriate living space, the student can contact the landlord to get more information and setup appointments.

Landlords have the ability to create apartment listings that can be seen by both guests and students. Landlords can provide information such as images, address, available times, and etc, which will allow students to better filter for their own apartment.

All postings shall be moderated by the sites administrators who can remove inappropriate or no longer relevant apartment listings. Administrators can also remove user accounts that are unproductive or toxic for the website's community.

myPlace's purpose is to deliver a simple and reliable apartment renting service to SFSU students. Although this type of service is not new, there has yet to be a renting service that targets SFSU students adequately.

As a small startup of SFSU students, we believe that *myPlace* will not just provide a better renting service, but also foster a better relationship between SFSU students, SFSU itself, and local landlords. We know the difficulties and obstacles in finding the perfect apartment and can provide competitive resources so that each user has the best overall experience.

Website URL: **http://sfsuswe.com/~iivanenk/MyPlace/**

Priority 1 Functionality

Priority Level: (1 - *must have)*

Status: ***IP:** in-progress;* **√**: *accomplished;* ***I:** incomplete;* **UI:** user interface

	Priority 1:	
	Data Definition	**Function**
√	***Website*** shall have:	Use of filters, search bar, and sort by features
IP	*(encryption & validation check)*	Login and SignUp
√		Thumbnail of Apartments & Featured Apartments
√	***Guest*** shall be able to:	Register for an account (only as SFSU student or Landlord)
√		Browse apartments
√		Filter, search, or sort apartment results
√	***Student*** shall be able to:	Login account with mail.sfsu.edu email
√		Filter, search, or sort by apartment results
IP		Message Landlord
√	***Landlord*** shall be able to:	Offer (multiple) apartments for rent
IP		Edit apartment (ie. description, images, delete, etc.)
√		View messages from interested students
√		Have guest view of apartments
√	***Admin*** shall be able to:	Use SQL Workbench to delete user accounts and/or apartments but not edit them
IP	***Apartment*** shall have ***data*** including	Title
IP		Description

√		Price
√		Zip Code
IP		Maps using Google Maps API
√		Images (optional - max 10)
√		Thumbnail
IP		Posted Date
√		Pet Friendly
√		Parking Available
√		Laundry Available
√		No Smoking
√		Shared Room
√		Furnished
√		Wheelchair accessible

Usability Test Plan

1. **Test Objectives**

   Finding apartments fast and easy is an important aspect of *myPlace*. The following tests focus on finding an apartment that meets exactly what students are looking for and if this function is easy to use. Students can search apartments by typing attributes and the zip code that they prefer in an apartment. Users can also search apartments with the search engine by just filtering and sorting the amenities of apartment details to find the living space that accommodates their needs.

2. **Test Plan**

- ***System Setup:*** Computer that is running a recent version of Safari, Chrome, and Mozilla.
- ***Starting Point:*** Homepage of myPlace.
- ***Task to be Accomplished:***
  1. Use search engine to find apartments in database
     a. Find apartments in zip code 94132
     b. Find apartments near coffee houses
  2. Filter apartment results
     a. Find apartments with 3 bedrooms
     b. Filter apartments that are pet friendly and no smoking
- ***Intended User:*** Registered user, SFSU student.
- ***Completion Criteria:*** User searched and filtered apartment listings and found an apartment that met their criteria. The returned values are accurate. And as a result, users are able to search and find an apartment or a website response at each step.
- ***URL of the system to be tested:* http://sfsuswe.com/~iivanenk/MyPlace/**

3. Questionnaire

Please circle the answer that best describes your feelings to the following statements.

Question 1: *It was easy to navigate through the homepage to find apartments.*

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree

Comments:

Question 2: *The returned search engine results were accurate/useful***.**

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree

Comments:

Question 3: *The filters on the site were convenient to find an apartment.*

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree

Comments:

Question 4: *I would recommend myPlace to other SFSU students.*

Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree

Comments:

QA Test Plan

1. Test Objectives

Ensure the functionality of searching apartments and filtering results to refine the list of apartments available to rent.

2. HW & SW setup

myPlace supports the current browsers: Safari, Chrome, and Mozilla. An Internet connection is mandatory in order to access and interact with the web application.

3. Feature to be tested

Repeat tests for the search engine and filter component on all browsers to ensure the stability, usability and functionality. The feature to be tested shall return the results according the user input. For example, when the user searches for a specific number of bedrooms, it shall return only the apartments that meet the requested specifications. In addition, the user could filter all the apartment results that display it respectively to what is stated in the filters. As a result, as users search for a particular apartment that meets their criteria in mind, the user could expect precise search results.

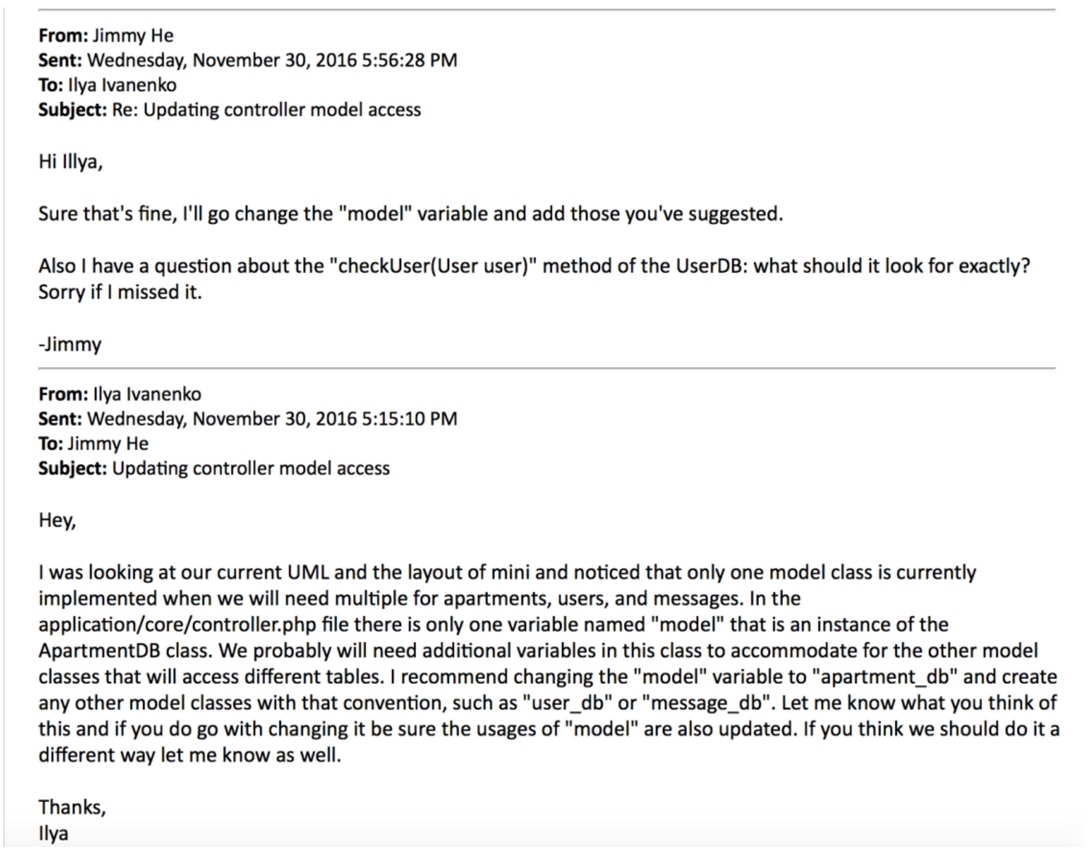
4. Actual Test Cases

Note: Displaying all apartment results take approximately 2-3 seconds to load

Test No.	**Description**	**Test Input**	**Expected Output**	**Pass/Fail**
1	Open webpage: http://sfsuswe.com/~iivanenk/MyPlace/ Hit "All Apartments" to view all available apartments	Click on the "ALL APARTMENTS" button	16 total apartments	PASS
2	Following Test #1 Mark the filters checkboxes: No smoking; shared room; pet friendly. Then "Filter" button	**Search:** " " **Filter:** Pet Friendly; No smoking; shared Room	Apartment that meet the tested filtered criteria. **1 Result** - 2 bdrm apt $5000	PASS
3	Open webpage: http://sfsuswe.com/~iivanenk/MyPlace/ Go to search engine and type area code	**Search:** "94132"	Apartments that meet the searched criteria **2 Results** - 2 bdrm apt $2500 - 1 bdrm apt $2900	PASS
4	Following Test #3 From the filter section, sort the sorted results by Price: High to Low	**Search:** "94132" **Filter - Sort By:** Price: High to Low	Apartments that met the searched and sort criteria **2 Results** - 1 bdrm apt $2900 - 2 bdrm apt $2500	PASS
5	Open webpage: http://sfsuswe.com/~iivanenk/MyPlace/ From the filter section, enter price range	**Filter - Price:** Max: $3000 Min: $4800	Apartments that met the price range **8 Results**	PASS

Code Review

The team is using Netbeans as an IDE for the project. The coding style we agreed on is that we are making sure we follow naming conventions, proper indentation, readability, and ensuring there are comments in our source code. We settled on those type of styles because it serves as a purpose to have a consistent look for the code so code review can be productive. The team code review sessions were in person because we all came to terms that it is much more helpful to do this face-to-face. That way there is less miscommunication, the process is agile and as a result the feedback with one another is accurate. Figures 1 - 6 displays conversations between our tech-lead, Ilya and our back-end developer, Jimmy clarifying questions after their in-person code review conference. They both discussed beforehand what their goals are and then evaluated each other code. After their meeting they continued working on their own time and when anyone of them hit a roadblock, they emailed each other for help as well as updating their tasks. Their dialogue was friendly, and intended to help and educate. This peer review did not consist of any criticism therefore it helped with the development of the whole system.



From: Jimmy He
Sent: Wednesday, November 30, 2016 5:56:28 PM
To: Ilya Ivanenko
Subject: Re: Updating controller model access

Hi Illya,

Sure that's fine, I'll go change the "model" variable and add those you've suggested.

Also I have a question about the "checkUser(User user)" method of the UserDB: what should it look for exactly? Sorry if I missed it.

-Jimmy

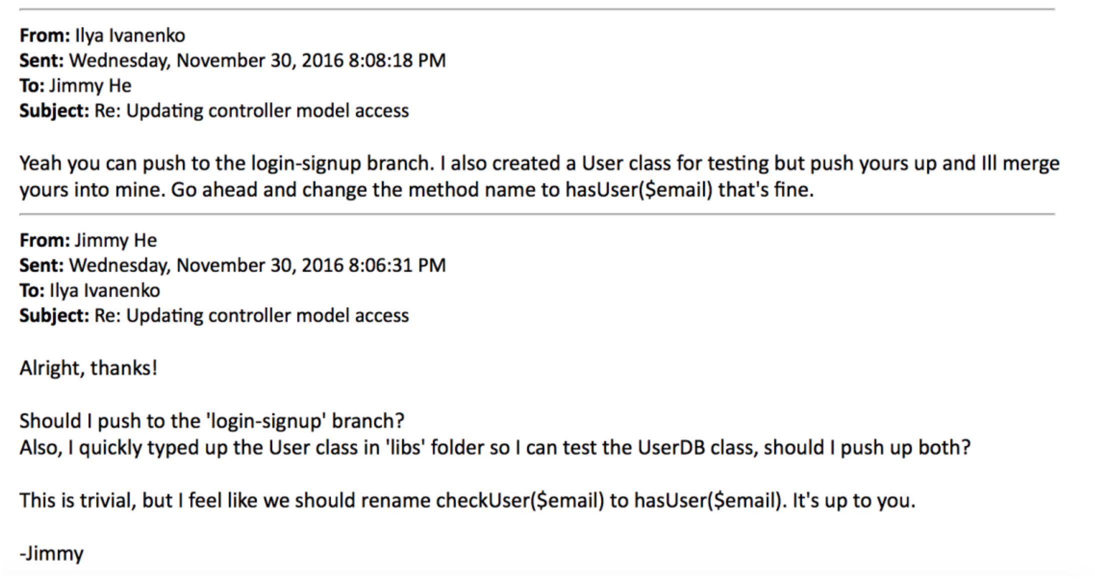
From: Ilya Ivanenko
Sent: Wednesday, November 30, 2016 5:15:10 PM
To: Jimmy He
Subject: Updating controller model access

Hey,

I was looking at our current UML and the layout of mini and noticed that only one model class is currently implemented when we will need multiple for apartments, users, and messages. In the application/core/controller.php file there is only one variable named "model" that is an instance of the ApartmentDB class. We probably will need additional variables in this class to accommodate for the other model classes that will access different tables. I recommend changing the "model" variable to "apartment_db" and create any other model classes with that convention, such as "user_db" or "message_db". Let me know what you think of this and if you do go with changing it be sure the usages of "model" are also updated. If you think we should do it a different way let me know as well.

Thanks,
Ilya

Figure 1 - Database Discussion (between Ilya and Jimmy)



From: Ilya Ivanenko
Sent: Wednesday, November 30, 2016 8:08:18 PM
To: Jimmy He
Subject: Re: Updating controller model access

Yeah you can push to the login-signup branch. I also created a User class for testing but push yours up and Ill merge yours into mine. Go ahead and change the method name to hasUser($email) that's fine.

From: Jimmy He
Sent: Wednesday, November 30, 2016 8:06:31 PM
To: Ilya Ivanenko
Subject: Re: Updating controller model access

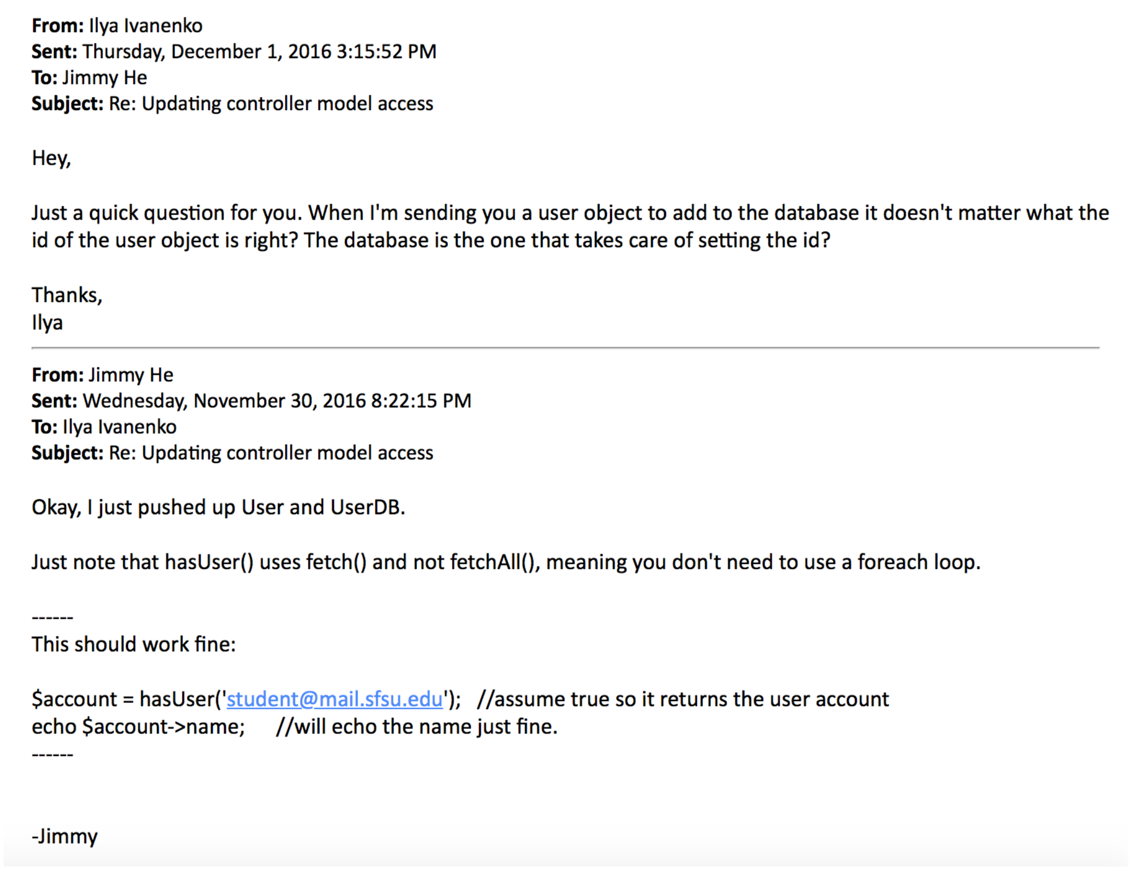
Alright, thanks!

Should I push to the 'login-signup' branch?
Also, I quickly typed up the User class in 'libs' folder so I can test the UserDB class, should I push up both?

This is trivial, but I feel like we should rename checkUser($email) to hasUser($email). It's up to you.

-Jimmy

Figure 2 - Merge Login-Signup Branch Request (between Ilya and Jimmy)



From: Ilya Ivanenko
Sent: Thursday, December 1, 2016 3:15:52 PM
To: Jimmy He
Subject: Re: Updating controller model access

Hey,

Just a quick question for you. When I'm sending you a user object to add to the database it doesn't matter what the id of the user object is right? The database is the one that takes care of setting the id?

Thanks,
Ilya

From: Jimmy He
Sent: Wednesday, November 30, 2016 8:22:15 PM
To: Ilya Ivanenko
Subject: Re: Updating controller model access

Okay, I just pushed up User and UserDB.

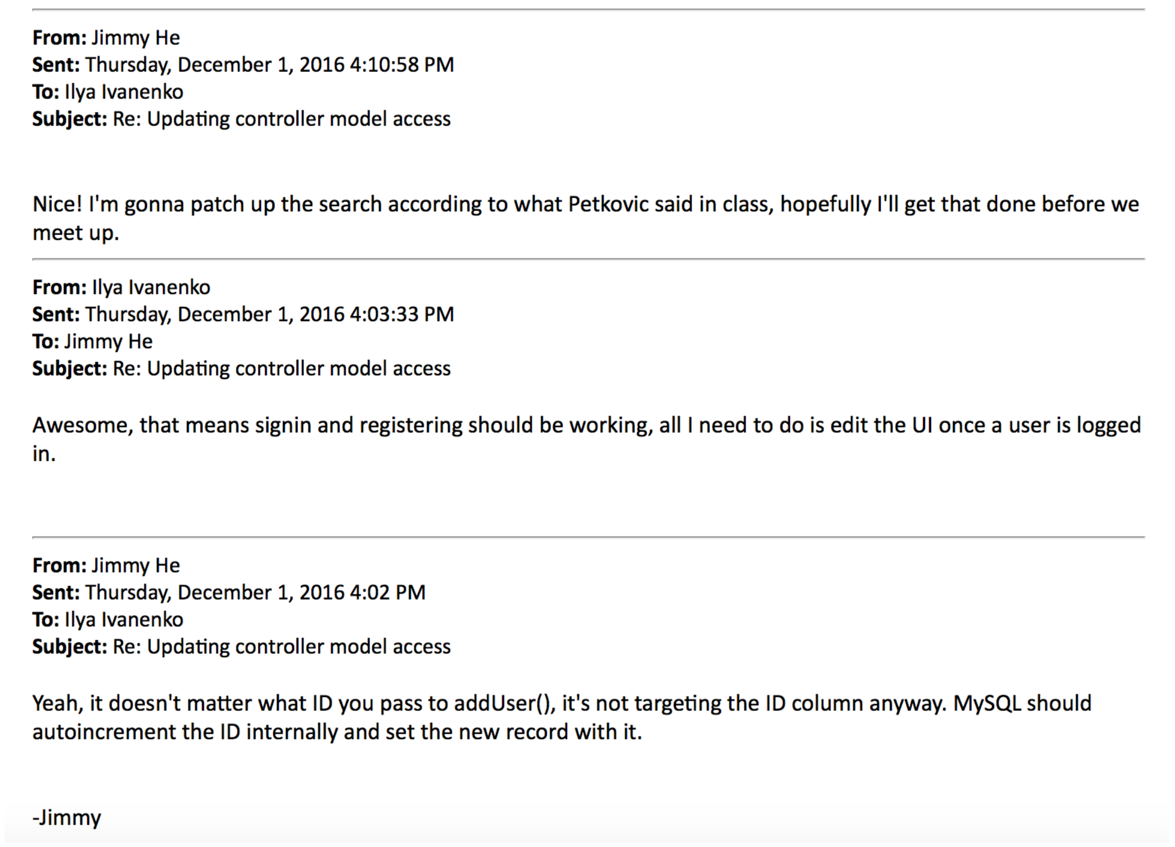
Just note that hasUser() uses fetch() and not fetchAll(), meaning you don't need to use a foreach loop.

This should work fine:

```
$account = hasUser('student@mail.sfsu.edu');   //assume true so it returns the user account
echo $account->name;     //will echo the name just fine.
```

-Jimmy

Figure 3 - Code Review for Database (between Ilya and Jimmy)



From: Jimmy He
Sent: Thursday, December 1, 2016 4:10:58 PM
To: Ilya Ivanenko
Subject: Re: Updating controller model access

Nice! I'm gonna patch up the search according to what Petkovic said in class, hopefully I'll get that done before we meet up.

From: Ilya Ivanenko
Sent: Thursday, December 1, 2016 4:03:33 PM
To: Jimmy He
Subject: Re: Updating controller model access

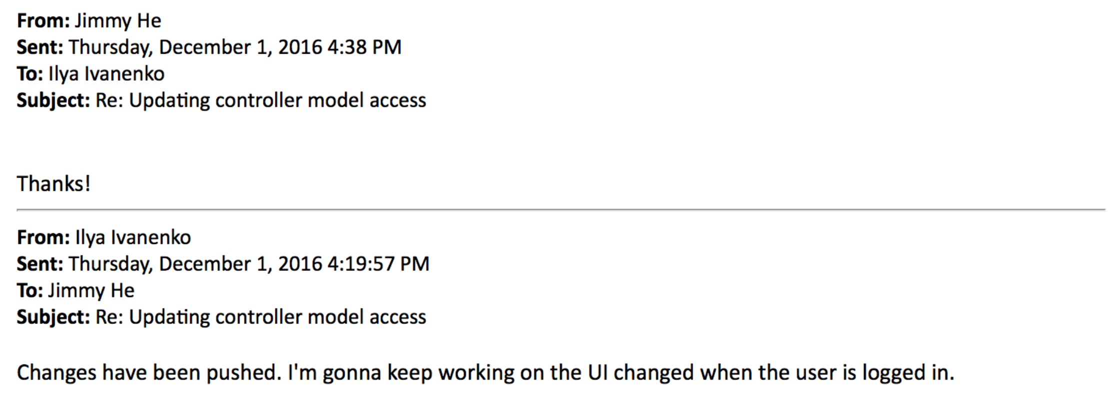
Awesome, that means signin and registering should be working, all I need to do is edit the UI once a user is logged in.

From: Jimmy He
Sent: Thursday, December 1, 2016 4:02 PM
To: Ilya Ivanenko
Subject: Re: Updating controller model access

Yeah, it doesn't matter what ID you pass to addUser(), it's not targeting the ID column anyway. MySQL should autoincrement the ID internally and set the new record with it.

-Jimmy

Figure 4 - Feedback (between Ilya and Jimmy)



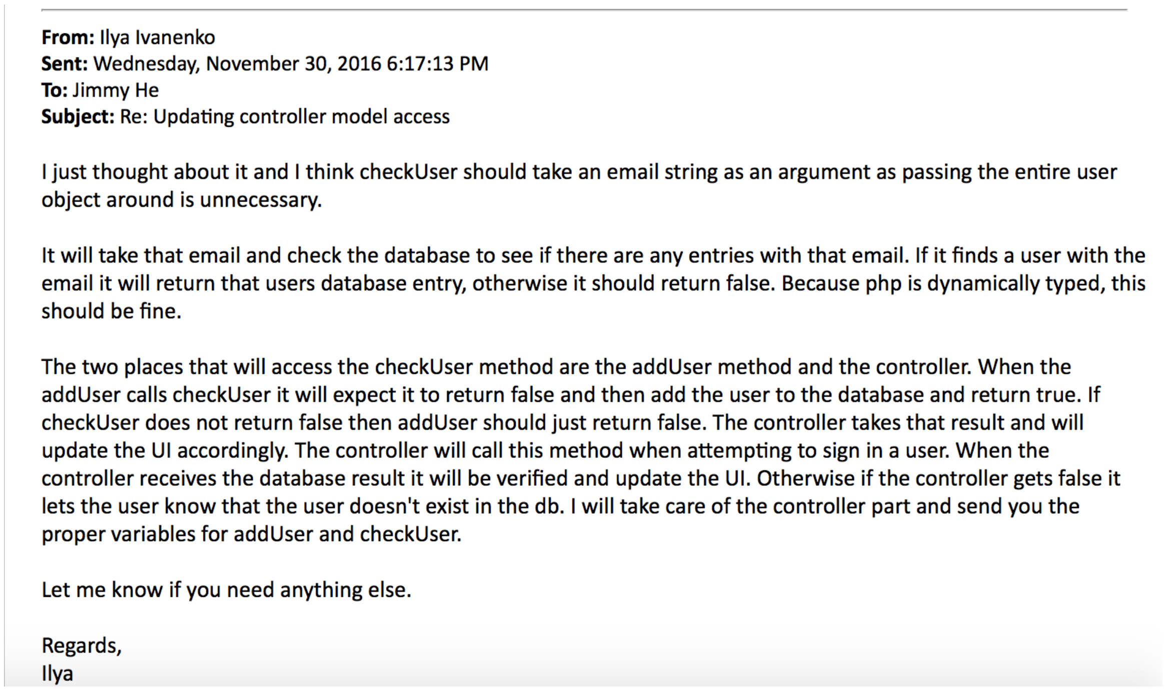
From: Jimmy He
Sent: Thursday, December 1, 2016 4:38 PM
To: Ilya Ivanenko
Subject: Re: Updating controller model access

Thanks!

From: Ilya Ivanenko
Sent: Thursday, December 1, 2016 4:19:57 PM
To: Jimmy He
Subject: Re: Updating controller model access

Changes have been pushed. I'm gonna keep working on the UI changed when the user is logged in.

Figure 5 - Updates (between Ilya and Jimmy)



From: Ilya Ivanenko
Sent: Wednesday, November 30, 2016 6:17:13 PM
To: Jimmy He
Subject: Re: Updating controller model access

I just thought about it and I think checkUser should take an email string as an argument as passing the entire user object around is unnecessary.

It will take that email and check the database to see if there are any entries with that email. If it finds a user with the email it will return that users database entry, otherwise it should return false. Because php is dynamically typed, this should be fine.

The two places that will access the checkUser method are the addUser method and the controller. When the addUser calls checkUser it will expect it to return false and then add the user to the database and return true. If checkUser does not return false then addUser should just return false. The controller takes that result and will update the UI accordingly. The controller will call this method when attempting to sign in a user. When the controller receives the database result it will be verified and update the UI. Otherwise if the controller gets false it lets the user know that the user doesn't exist in the db. I will take care of the controller part and send you the proper variables for addUser and checkUser.

Let me know if you need anything else.

Regards,
Ilya

Figure 6 - Follow up from face-to-face Code Review (Ilya to Jimmy)

Self-check on Best Practices for Security

1. **Protected assets**: User’s information; Landlord's contact information and messages, and databases.
2. **Confirm password is encrypted in the DB:** (DONE)
3. **Input data validation:**
    a. Check if fields are empty and/or proper format
    b. Check if email are validated
        i. Proper format
        ii. Check if email exists
        iii. Check if password is correct
        iv. Prevent SQL injection
    c. Search Engine checks input
        i. Checks for symbols
        ii. Checks if invalid information written
        iii. If invalid it will not allow search and will prompt user

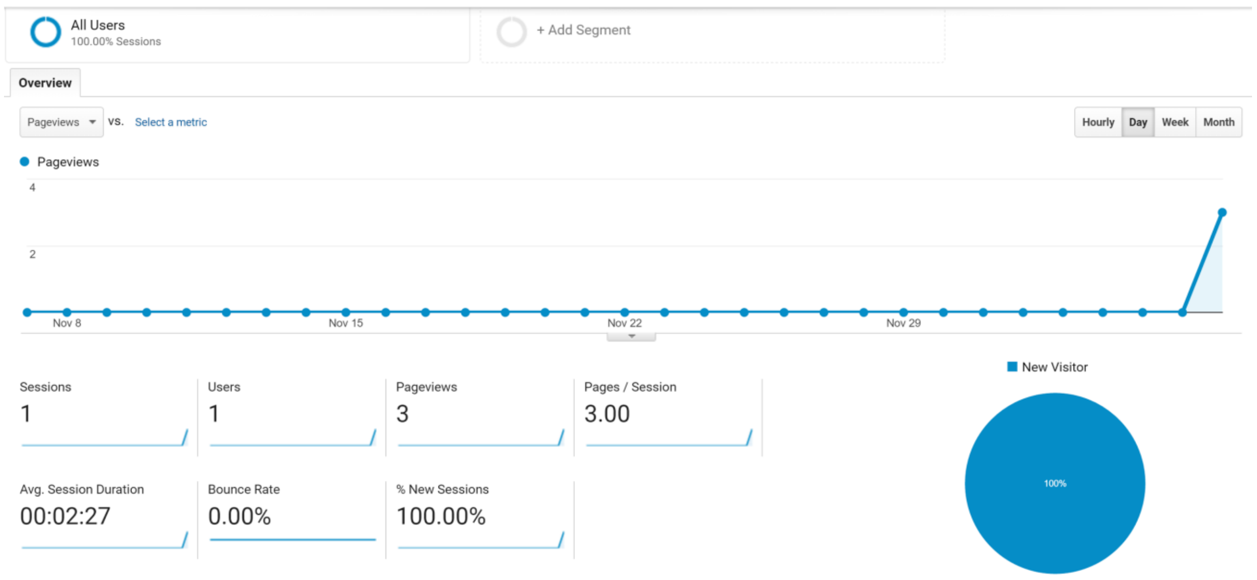
Adherence to Original Non-Functional Specs

(√) ***: "DONE";*** **(~)** ***: "ON TRACK";*** **(!)** ***: ISSUE;*** **(N/A)*****: Not Available***

√	**1.** Application shall be developed using class provided LAMP stack
√	**2.** Application shall be developed using pre-approved set of SW development and collaborative tools provided in the class. Any other tools or frameworks shall be explicitly approved by Marc Sosnick on a case by case basis.
	○ We shall use the following approved tools: JQuery Bootstrap, PHP, MySQL Workbench, Netbeans.
√	**3.** Application shall be hosted and deployed on Amazon Web Services as specified in the class.
√	**4.** Application shall be optimized for standard desktop/laptop browsers, and shall render correctly on the two latest versions of all major browsers: Mozilla, Safari, Chrome. It shall degrade nicely for different sized windows using class approved programming technology and frameworks so it can be adequately rendered on mobile devices.
	○ The application shall utilize Jquery to target Bootstrap's predefined CSS classes to customize our web design.
	○ The application shall utilize responsive web design features provided by the Bootstrap framework to maintain the application's design layout.
	5. Data shall be stored in the MySQL database on the class server in the team's account
√	**6.** Application shall be served from the team's account.
N/A	○ The application shall be deployed on the sfsuswe.com server, under the account of f16g14.
	○ *N/A: Discussed with CTO & CEO that group server is down*
	7. No more than 50 concurrent users shall be accessing the application at any time
	8. Privacy of users shall be protected and all privacy policies will be appropriately communicated to the users.
√	○ Landlord apartment listing details shall be hidden to guest users.
√	○ The privacy policy shall be presented to the each user upon registration and it must be accepted before an account is created.
	9. The language used shall be English.
	10. Application shall be very easy to use and intuitive. No prior training shall be required to use the website.
√	○ The application shall follow design properties of official SFSU websites, providing users who are familiar with SFSU to quickly grasp our design.
√	**11.** Google analytics shall be added for major site functions.

 √ ~ *N/A* √ √ √	**12.** Messaging between users shall be done only by class approved methods to avoid issues of security with e-mail services. **13.** Pay functionality (how to pay for goods and services) shall be simulated with proper UI, no backend. ○ The Payment interface shall have common transaction items such as usernames, pay amount, payment type (e.g credit card details), and the goods exchanged. A proper receipt shall be displayed after completing the transaction. ○ *N/A: Removed by management* **14.** Site security: basic best practices shall be applied (as covered in the class) **15.** Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development, and only the tools and practices approved by instructors. ○ Group 14 shall use email, Google Drive, GroupMe (texting service) for collaboration and feedback. **16.** The website shall prominently display the following text on all pages "SFSU/FAU/Fulda Software Engineering Project, Fall 2016. "For Demonstration Only". (Important so as to not confuse this with a real application).

Google Analytics



Sessions
1

Users
1

Pageviews
3

Pages / Session
3.00

Avg. Session Duration
00:02:27

Bounce Rate
0.00%

% New Sessions
100.00%