CS 307
Spring 2018

DESIGN DOCUMENT

2 Feb 2018

1ance

TEAM 29
Kenan Dominic
Shengqi Wang
Arnav Jain
Anjali Malik
Sanjit Sama

Purpose

College students have been known for their notorious saving tactics and penny-pinching expenditures. After being forced to adapt to a life of simplicity many are left craving a source of income. We at 1ance have recognized that though students are building their academic careers they already have a diverse set of skills and resources, viewed valuable by many of their peers.

Our project intends to bridge the gap between those who are looking to earn side income and those in need of services at a cheaper student friendly cost. Our intuitive web application will allow for two options. First of which will allow for students and professors to market their skills/resources on our platform specifying their price. The second feature will allow students to post requests if they have not been able to find someone already who has posted their skills/resource.

Design Outline

1Lance will be implemented using the client-server model. The server will respond to the client while parsing data using NodeJS. The server will also access data stored within a database allowing for multiple concurrent clients via our Web API. The client will interact with the server to send, retrieve, delete, and update the data stored in our MySQL database, and will display information back to the user accordingly, while securing an authentication connection to prevent DB injection.

1. Client

- The client is a web application, which will communicate with the Backend based on user's interaction of the application, and send requests to the server, and display appropriate response on the frontend.
- Users will be able to interact with the application easily by accessing the website online in a web browser.
- The client application will use HTTP requests to communicate with the server and pass data in form of JSON objects or query strings.
- The client application will also display the user data received from the server in a user-friendly format.

2. Server

- The server will be hosted using NodeJS, which will communicate and query with the database using a security protocol.

- The server will need authentication information from the backend of the client, to be able to accordingly make commits to our database system.
- The server will handle multiple clients and their requests simultaneously.
- The server will also return secure response strings including any necessary error messages, to the user.

3. Database

- The database system used for our application will be MySQL.
- The database will be responsible for securely storing user login information, profiles, service postings, request postings, activity information on discussion boards, etc.

High Level Overview

The Web API server form a many-to-one client-server architecture to serve multiple concurrent users, events, and communications. The usual flow of information starts with the client requesting some information from the server, and the server in response, queries the database for data and sends it back to the client. Similarly, client sends information to be stored or updated and the server writes that data to the database. As shown in the figure below, we will also implement automation testing, to continuously check the status of our server. Security being a critical element of 1ance, all information and connections will be secured and authenticated.

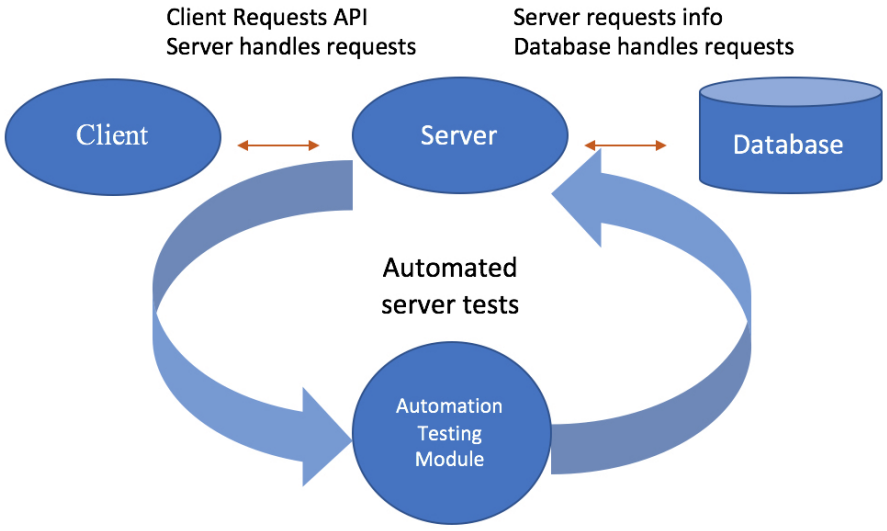

Figure 1: This figure demonstrates a high-level overview of our client-server system.

Design Outline

Design Issue # 1

How long shall requests/offers lasts for?

Option 1: Till the post is closed by the user

Option 2: Till the user has deleted their account

Option 3: For a user defined period of time

Decision: Often users forget tend to forget to close a post after they have received a service or are no longer interested in fulfilling one, leading to an accumulation of inactive Offers and requests on the database. Hence, we request the user to provide a deadline for their post and the post is automatically removed once the deadline is met.

Design Issue # 2

How shall offers/requests be organized?

Option 1: By date posted

Option 2: By popularity

Option 3: By relevance

Option 4: Point based system that factors in relevance, popularity and date.

Decision: Sorting posts will be by default based on a point system that will be calculated and updated constantly by a backend function. This is to give the user the best experience possible. However, the user will also have the ability to sort by any of other options listed above.

Design Issue # 3

What shall clicking on a post do?

Option 1: Redirect the user to a post-specific page

Option 2: Present a modal view with more specific post details to the user

Option 3: Have the card rotate presenting more details about the post

Decision: Redirecting the user to a new page is inefficient with the transmission of data from page to page. Users in general prefer to stay on the same page and not being forced to click back every time they want to read about a post. Having the card rotate is aesthetically pleasing but this results in having half the information on the other side. As a result, we decided on using a modal view that will pop up over the view on the current page and having all details about the post in one place.

Design Issue # 4

How shall users receive notifications from the application?

Option 1: Via Email

Option 2**:** Via Text

Option 3: On site pop-ups and badges

Decision: Email notifications are outright annoying and our users who are primarily students wouldn't want their work emails cluttered with email notifications. Text messages are often reserved for more urgent matters relative to service offers and requests. Although we do provide the option for a user to turn on text notifications they are switched off by default and notifications are primarily on the application itself in the form of badges and pop-ups.

Design Issue # 5

What shall deleting a user account do?

Option 1: Remove the user from the authentication database

Option 2: Option 1 and remove all user posts (both requests and offers)

Option 3: Option 1, Option 2 and undo all user actions such as comments, likes etc.

Decision: We decided to delete all posts by the user and remove them from the authentication database for obvious reasons. However, we think it's essential to maintain the users reviews, comments, likes etc as they can help other users make a more informed decision when choosing to request or offer a service.

Design Issue # 6

What technology will be powering our backend?

Option 1: Node.js

Option 2: PHP

Option 3: C#

Decision: Node.js uses an event driven, non blocking I/O model which translates to being able to execute multiple processes without causing a an input-output bottleneck in the server side script. It is also more efficient and easier to maintain and update than its competitors.

Design Issue # 7

What backend hosting service will we use to power our back end?

Option 1: Heroku

Option 2: AWS

Option 3: Firebase

Decision: AWS is a far more powerful service than the two other services we have considered. AWS also offers a unique service called EC2 (Elastic Cloud Computing)

which allows developers to run their web applications on a virtual computer whose computing power is scalable as per the developer's needs.

Design Issue # 8

Should we use HTTP or HTTPS?

Option 1: HTTP

Option 2: HTTPS

Decision: Security is crucial to our web application since we're dealing with sensitive user data. This data may if time allows be pulled from other social services like Facebook or LinkedIn using OAuth2.0 which ethically demands us, as developers to uphold similar security standards. HTTPS is the secure version of HTTP which ensures encrypted communications between our website and the browser, which is why we will be utilizing HTTPS for our service communication.

Design Issue # 9

What database will we use?

Option 1: MySQL

Option 2: Amazon RDS

Option 3: SQLite

Decision: We decided to use MySQL as it is a time tested RDBMS option and extensively documented too and almost all scripting languages are compatible with it SQLite is preferred for embedded applications and Amazon RDS has a learning curve that could hinder development.

Design Issue # 10

What frontend framework will we use?

Option 1: Angular

Option 2: React

Option 3: Bootstrap

Decision: We've decided to use a combination of both Angular and Bootstrap for our web application. We wanted to use bootstrap because it is great for creating responsive, mobile-friendly websites, whereas angular is powerful for its data-binding and dependency injection tools making it much easier to write code for components succinctly. Both of these together create beautiful and efficient UI, which is why we came to the decision of using them together.

Design Details

Class Diagram

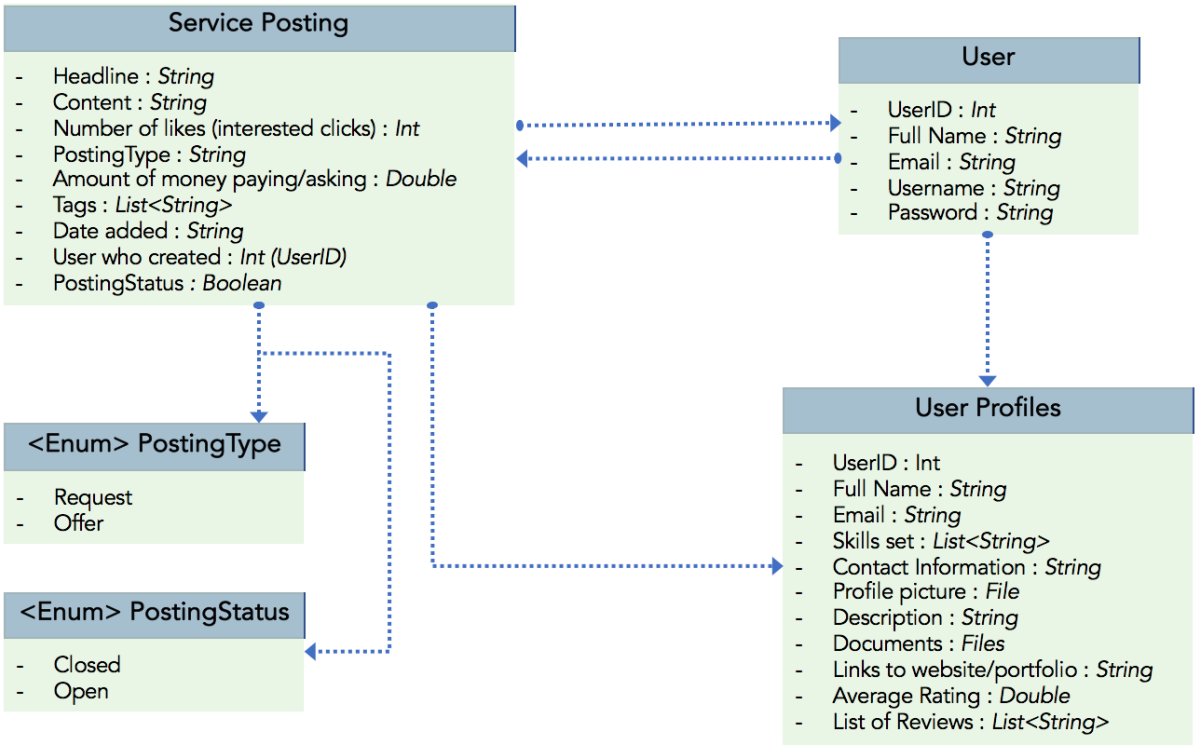

Figure 2: Class diagram illustrates types of objects in our project, and how they will be represented in the code as well as the Database.

User

- An instance of User is created anytime a person registers on our system.
- Specific information provided by them during the sign-up process is stored in this class, along with an automatic ID generated to uniquely identify each user in the system.
- This class will be central for authentication system in our code.

User Profile

- An instance of the User profile represents a unique user that is registered with our system.
- This instance will have a unique User ID, full name, and email from the parent User class.
- The user will provide additional information for sections: skill set, contact information, profile picture, description, documents, and links.

- User will have no access to editing their ratings and reviews which are another important part of the User Profile class.
- All of this information about the user will be stored in the User Profile table in the DB.

Service Posting

- All postings created by any user will make an instance of this class.
- As soon as a instance of Service posting is made, essential information such as the UserID of the user who created the post, and posting date will automatically be initialized.
- Users during creation of the post will provide information about the headline, content, posting type, and amount of money charging or willing to pay.
- Service postings will also have tags which will allow them to be categorized and filtered accordingly. These tags will be provided by the user posting it.
- The posting status initially will always be Open, until either user account is deleted or it is closed.

Navigational Flow Map

Our app will exploit Purdue's exclusivity, which essentially means that every user requires a Purdue email to sign up. We authenticate this by sending out an email post sign up with a randomly generated confirmation code. Once confirmed, it takes you to the homepage which is where the job feed lies. This job feed consists of request cards available for users to view and pursue. The users can then direct message the other user who posted the job card.

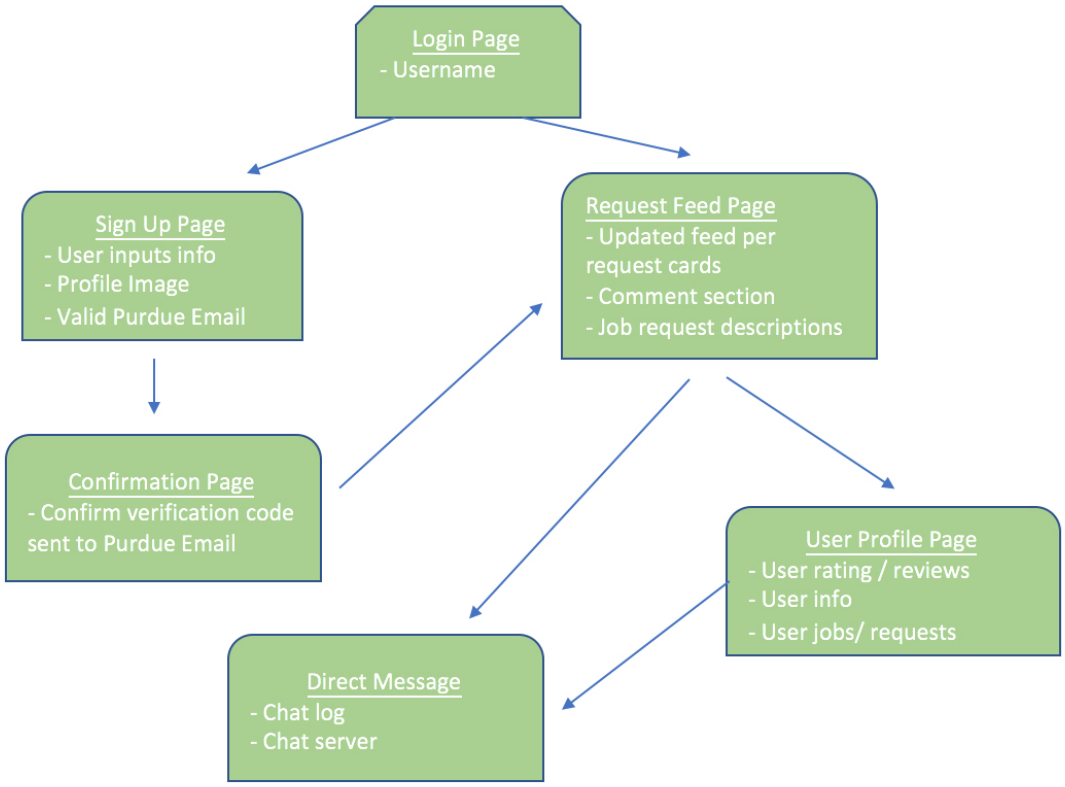

Figure 3: User work flow showing interactions between different pages

Sequence Diagrams

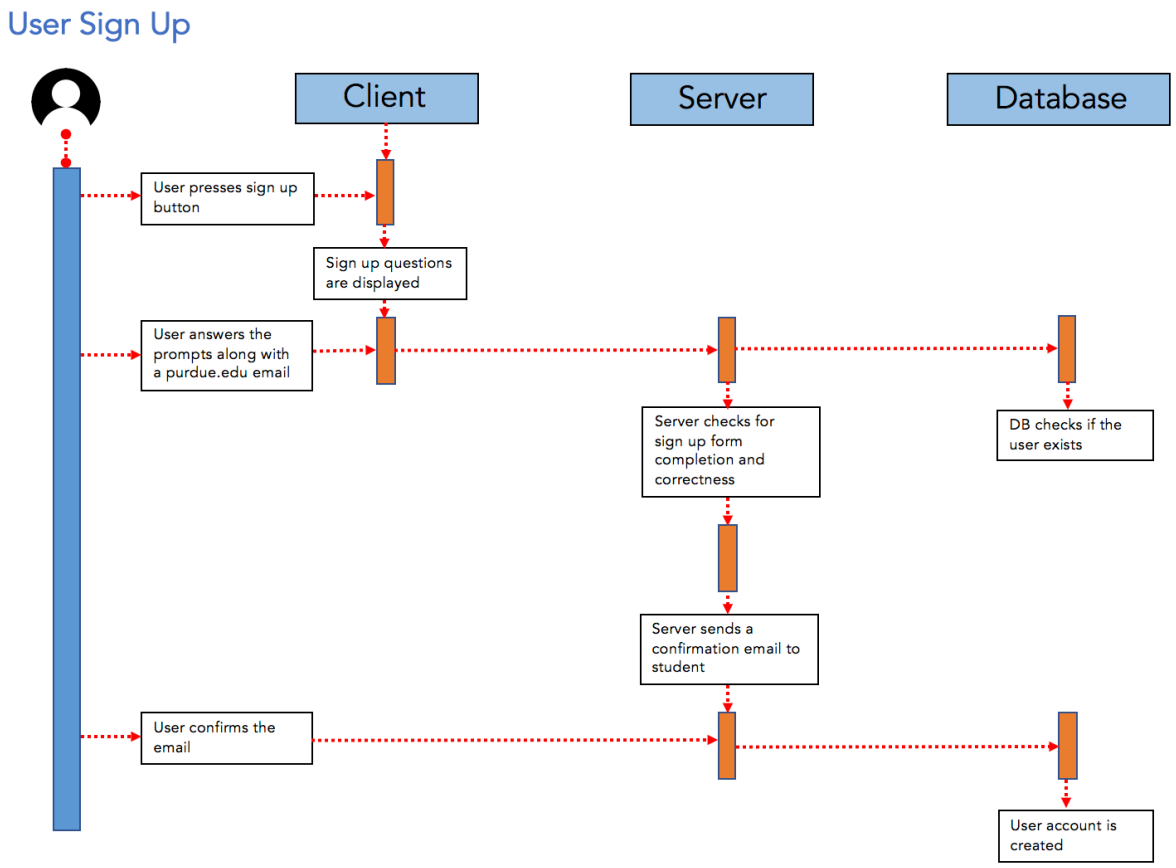

Figure 4: Sequence Diagram for User Sign up

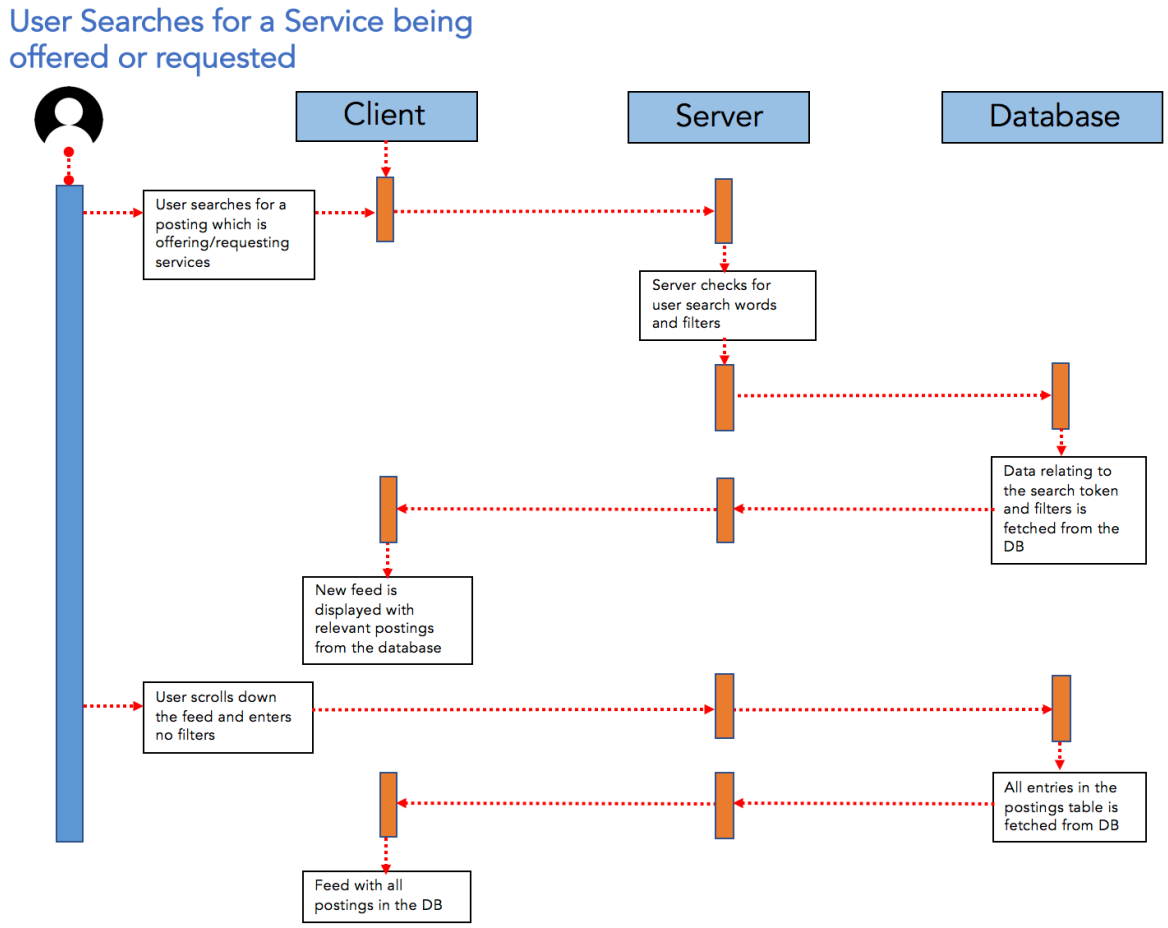

Figure 5: Sequence Diagram for User search in the postings feed

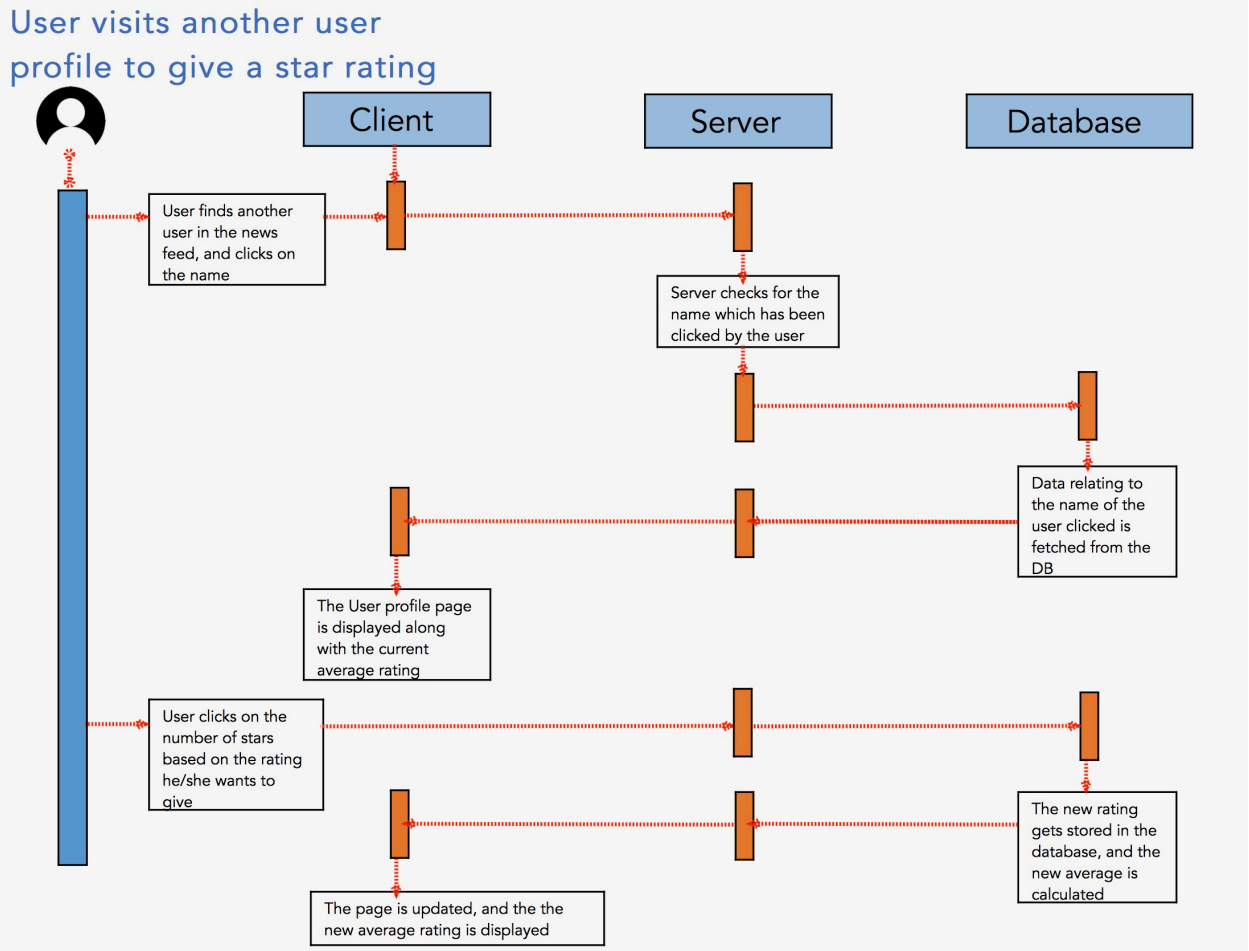

Figure 6: Sequence diagram for user giving a rating to another user

UI Mockups

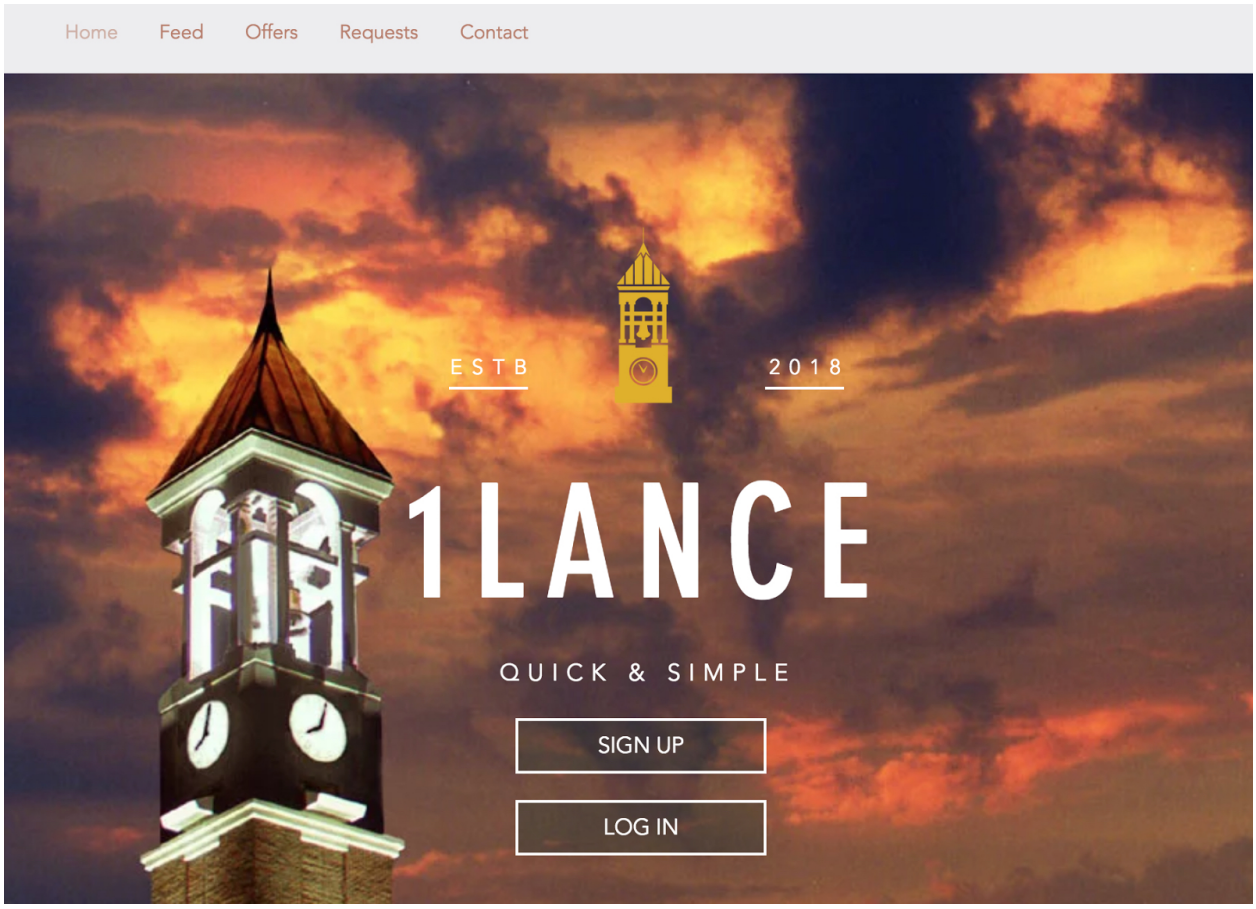

Figure 7: UI mockup showing the Login/Sign Up page for 1ance. Since this a Purdue based service, we wished to incorporate a Purdue cornerstone.

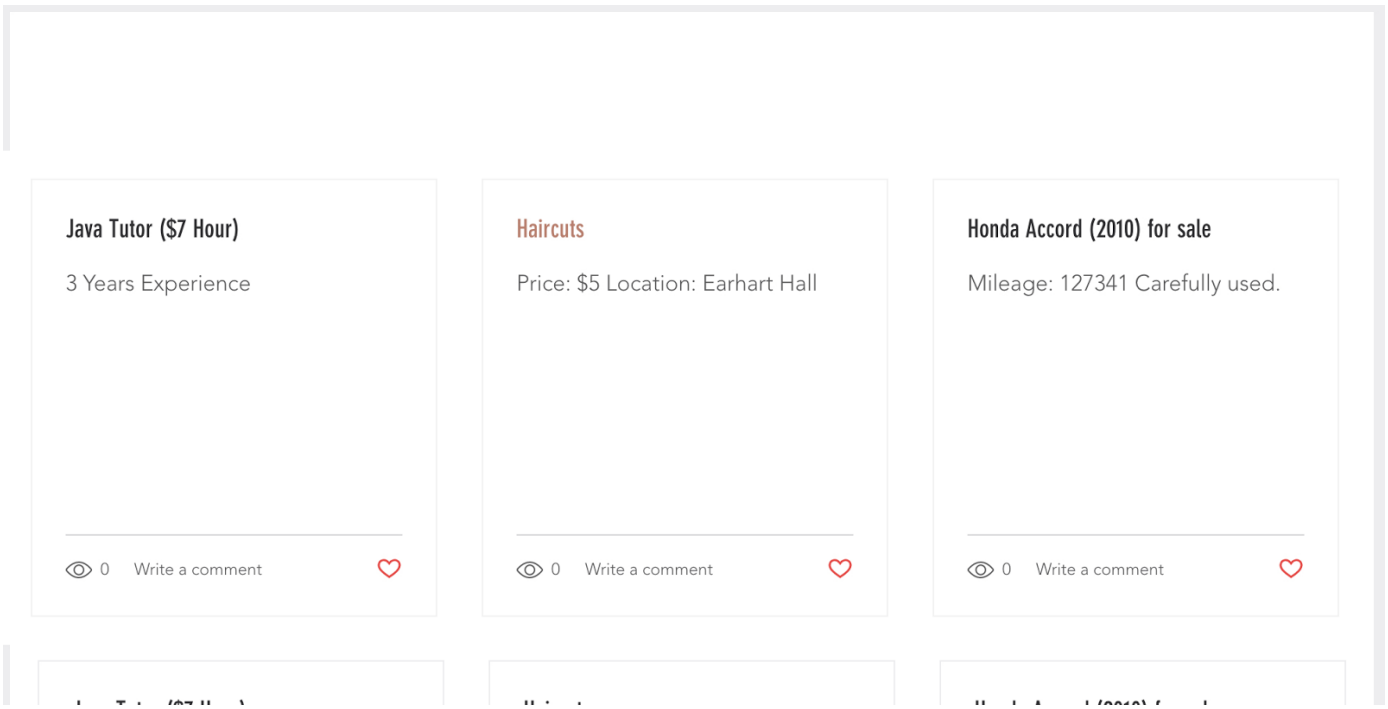

Figure 8: UI mockup displaying how user feed will look with posting cards. Users can either post a Request or an Offer. Users can request/Offer a range of items or services. They may include anything from a cheap Haircut to a Coding tutor to an used Car.

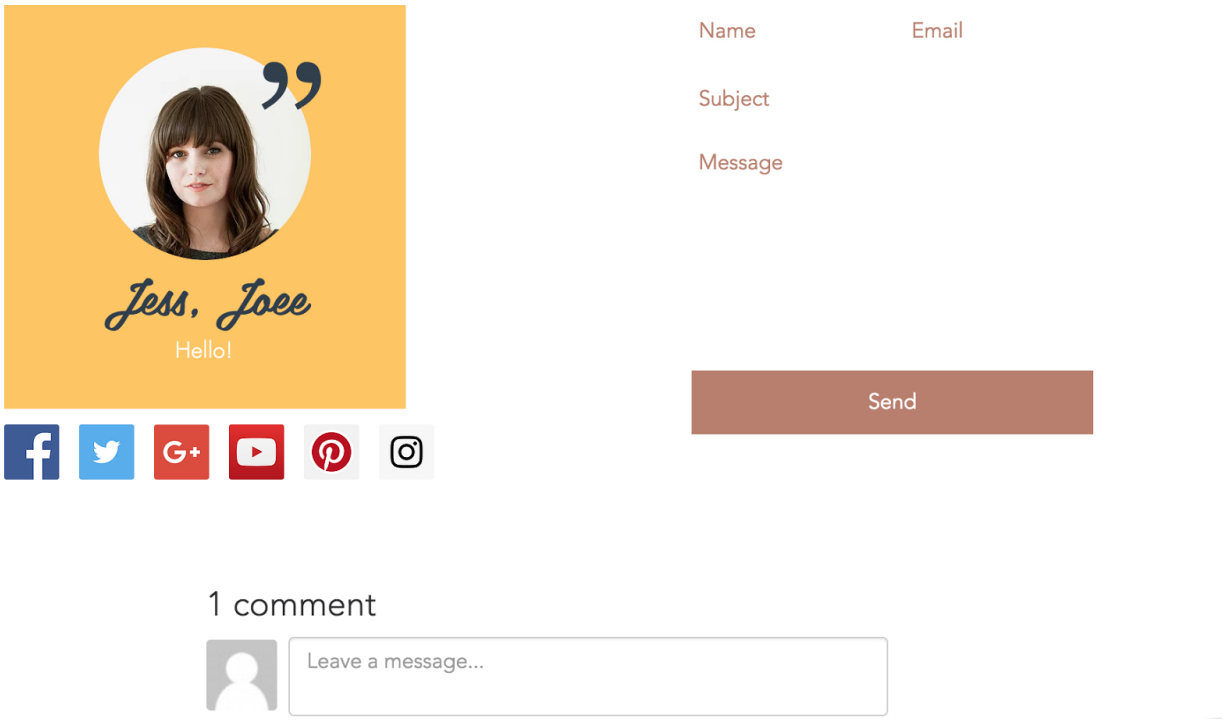

Figure 9: UI mockup showing starting of the profile page of a user. Since users will be dealing with strangers, profile clarity is very important. Which is why we have users fill in an about me, picture, and contact information.