**CMPE 273 – Lab 1**

**Sricheta Ruj (SJSU ID - 012527606)**

**(Part 1) Calculator-**

**Goal :**  To Develop a calculator.

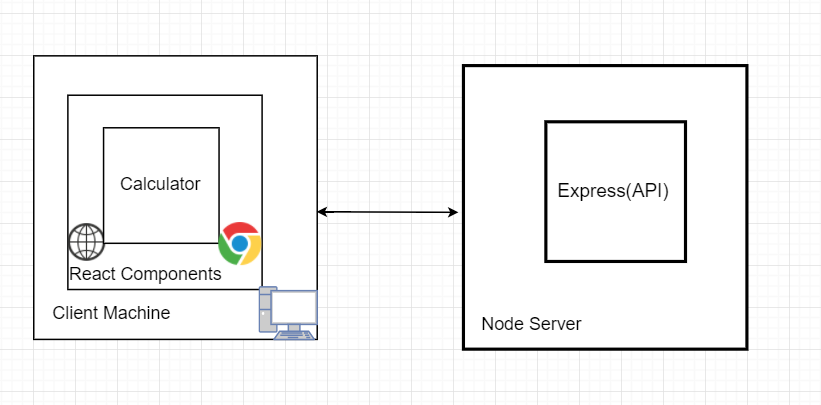
**Purpose of the system**: The purpose of the system is to develop a real-like Calculator. The Calculator includes following functionalities:

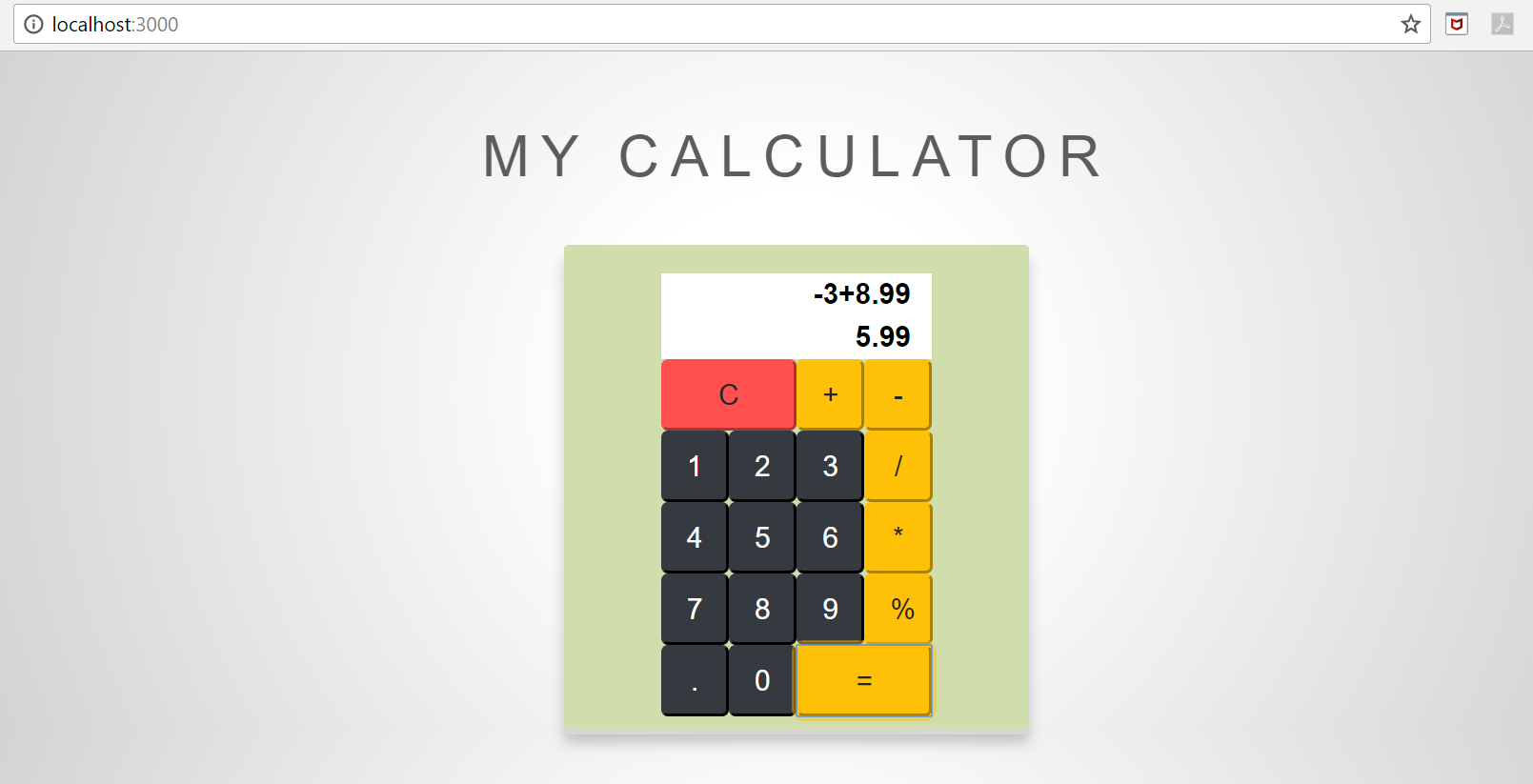
1. Addition
2. Subtraction
3. Division
4. Multiplication
5. Modulo
6. Clear button to clear the operation
7. Support of Decimal

**Technology Used**:

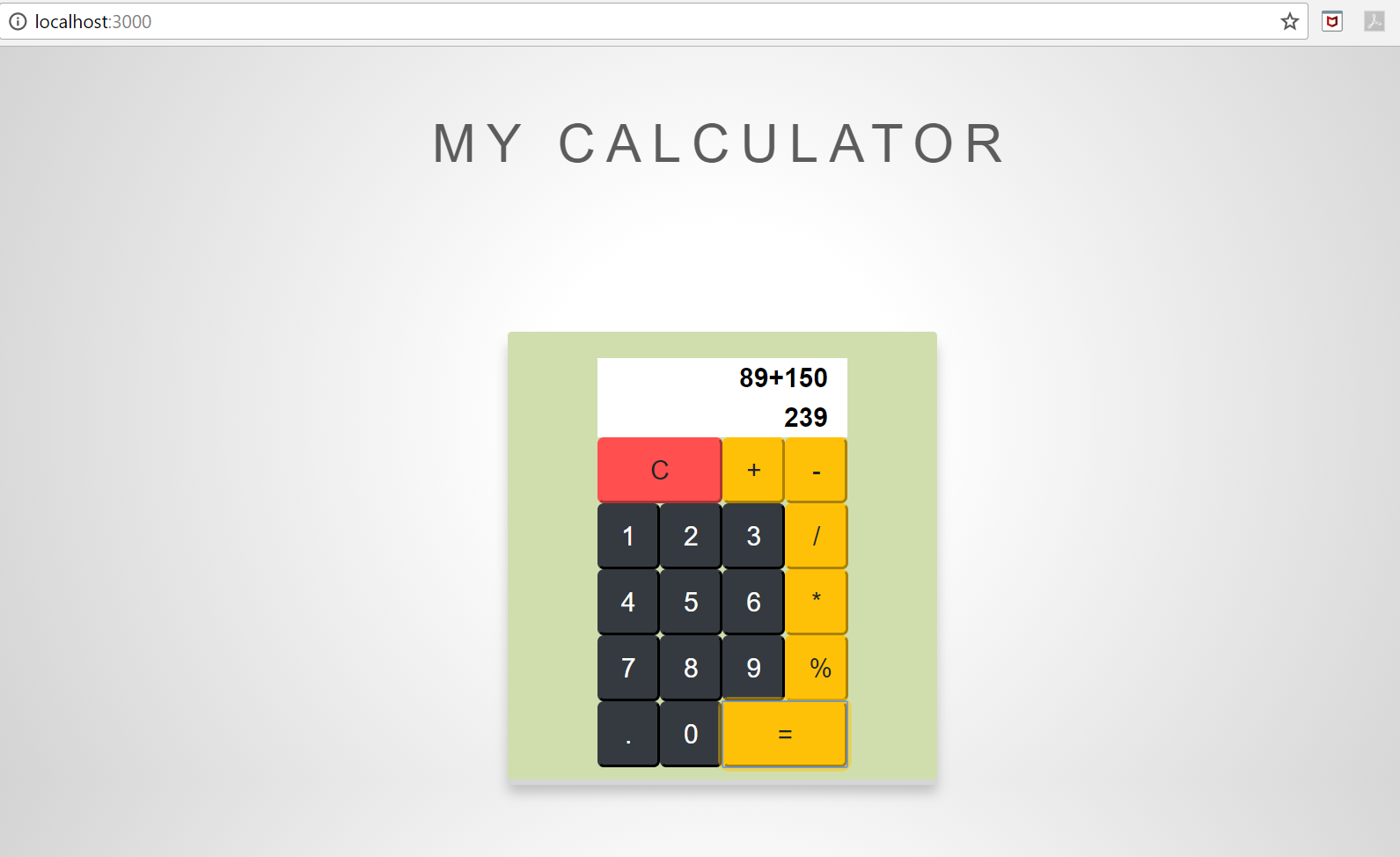
* React as View in MVC architecture
* Node JS as a platform to host rest services.
* ExpressJS as a web application framework.
* axios to call the REST service.

**Architecture Diagram:**

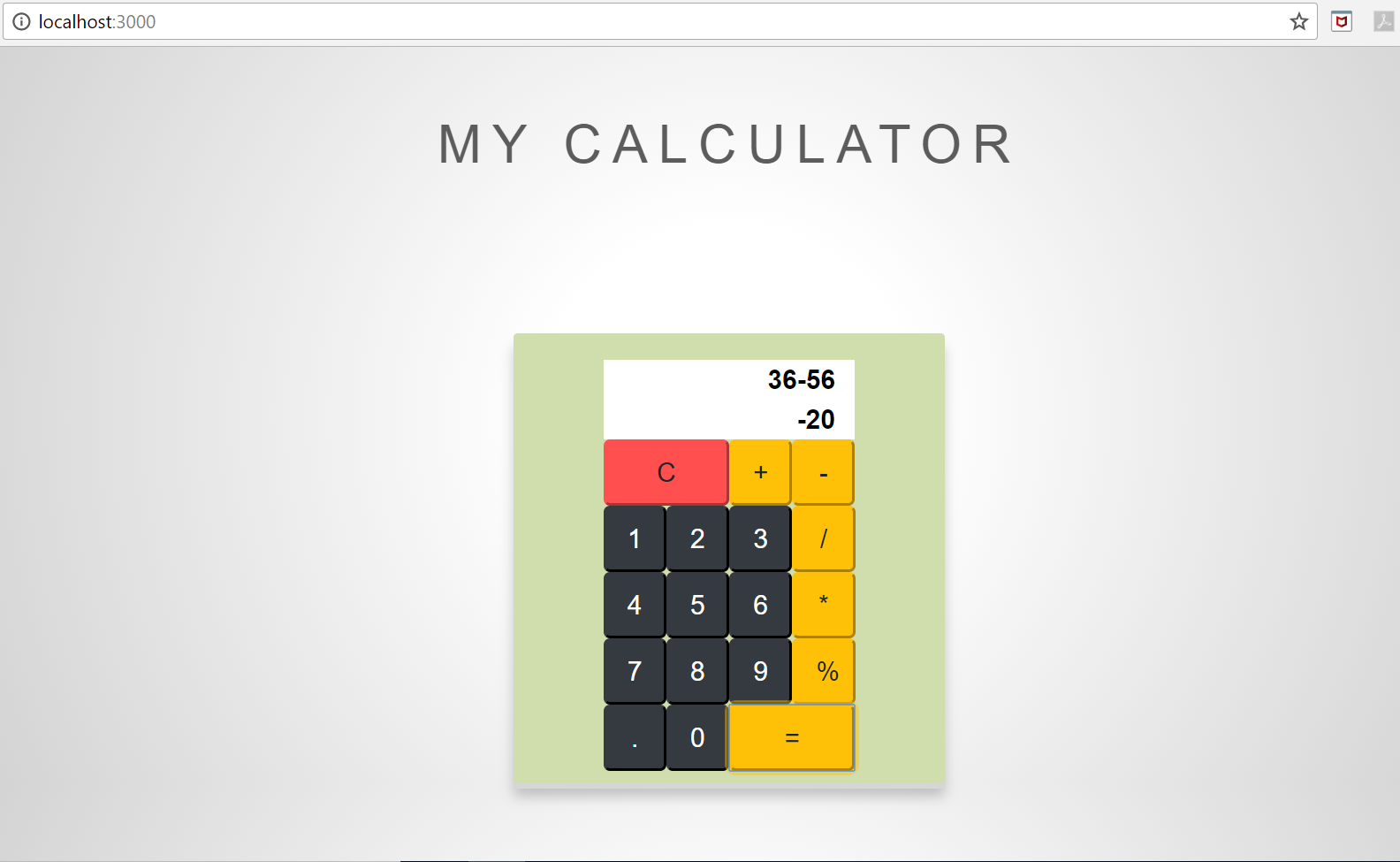




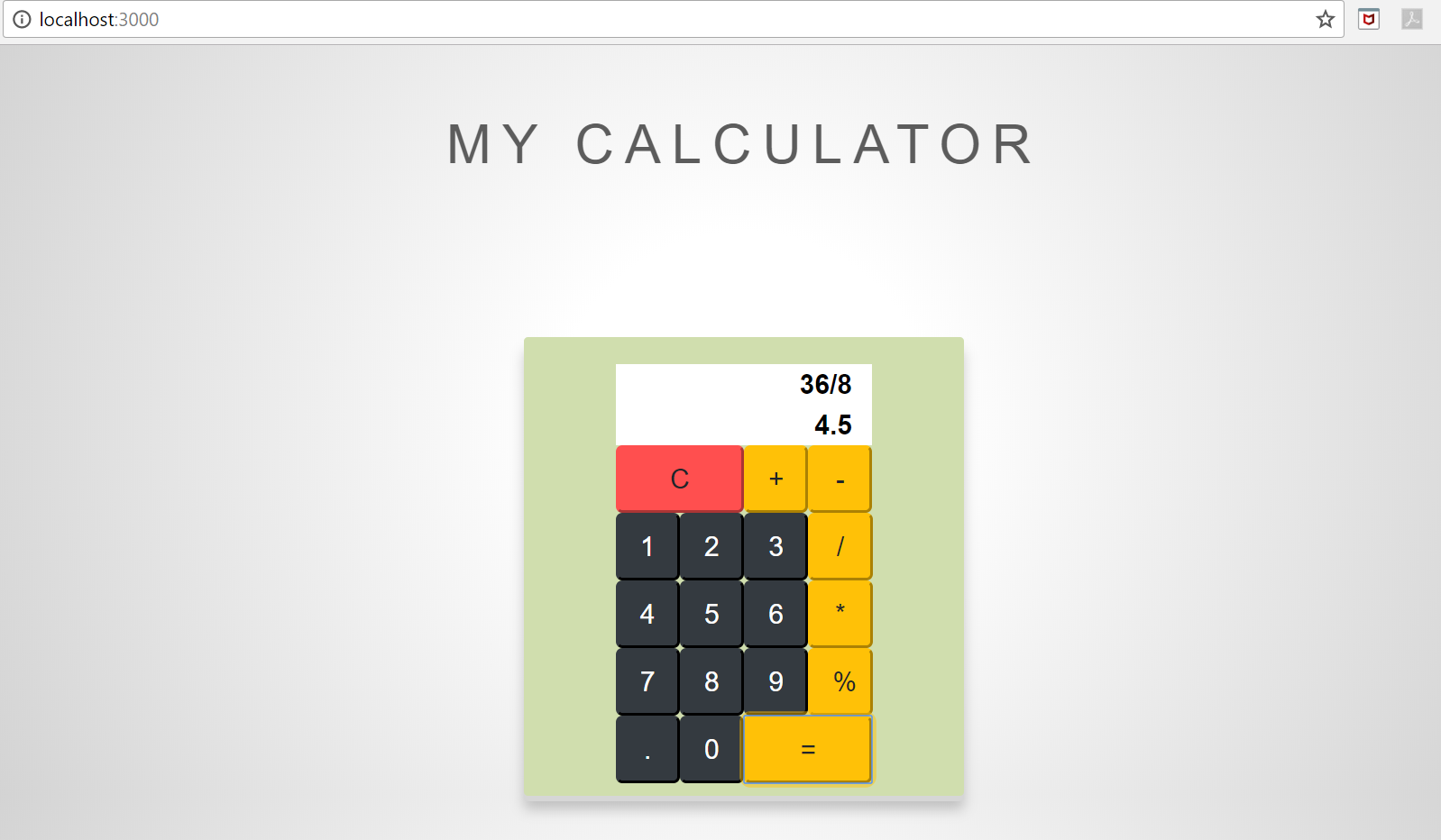
1. Addition Operation



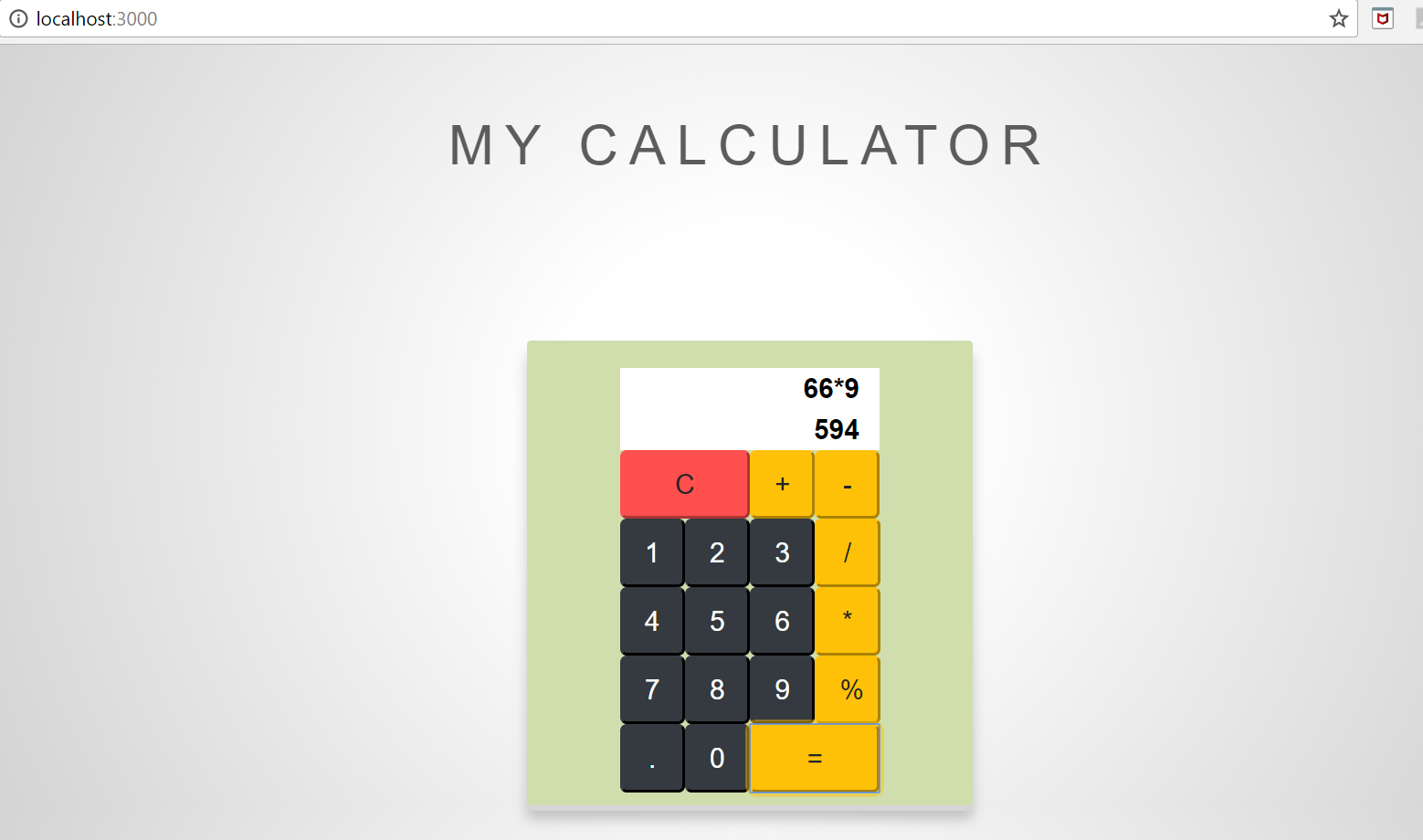
1. Subtraction Operation



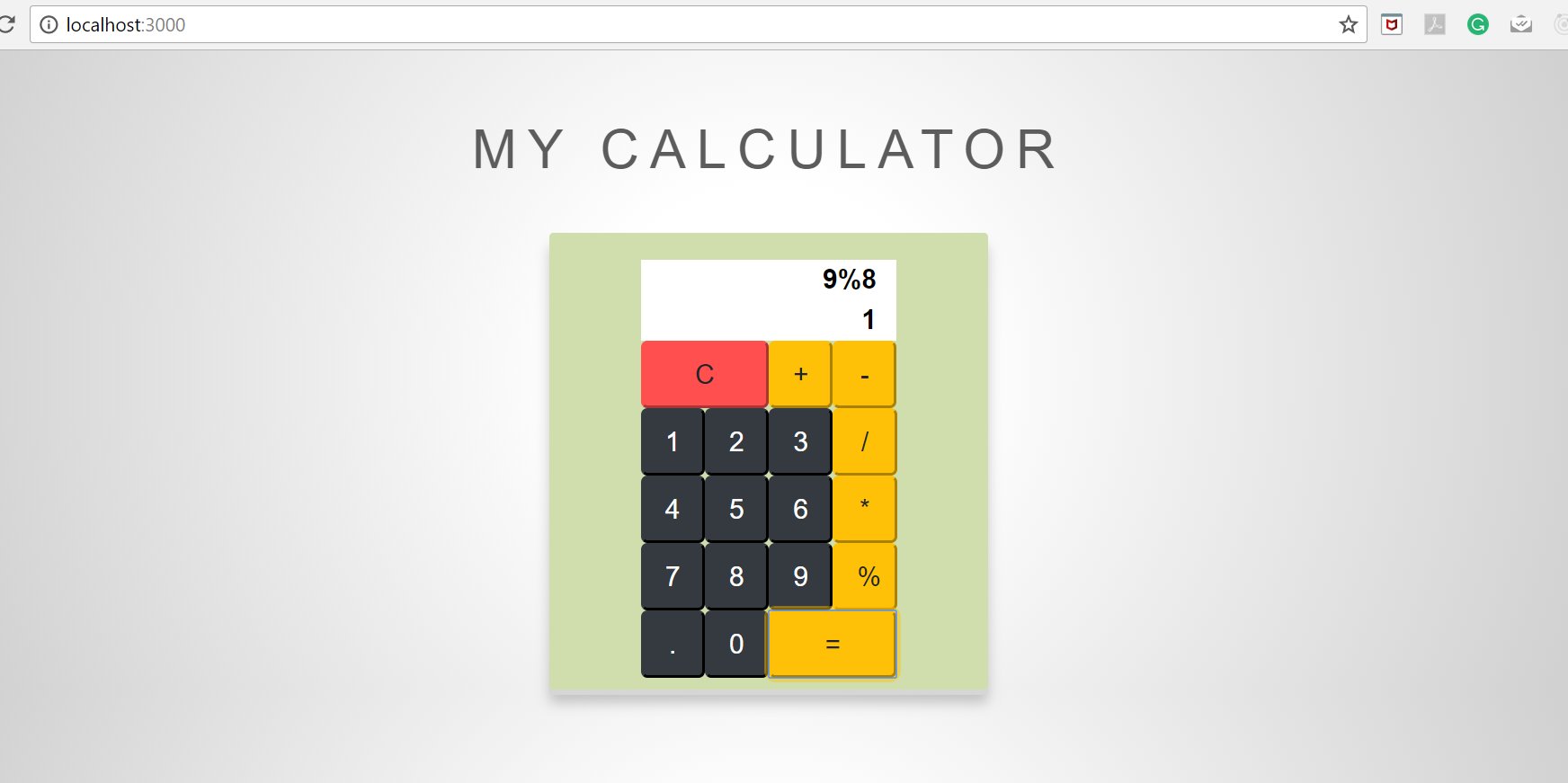
1. Division Operation



1. Multiplication Operation

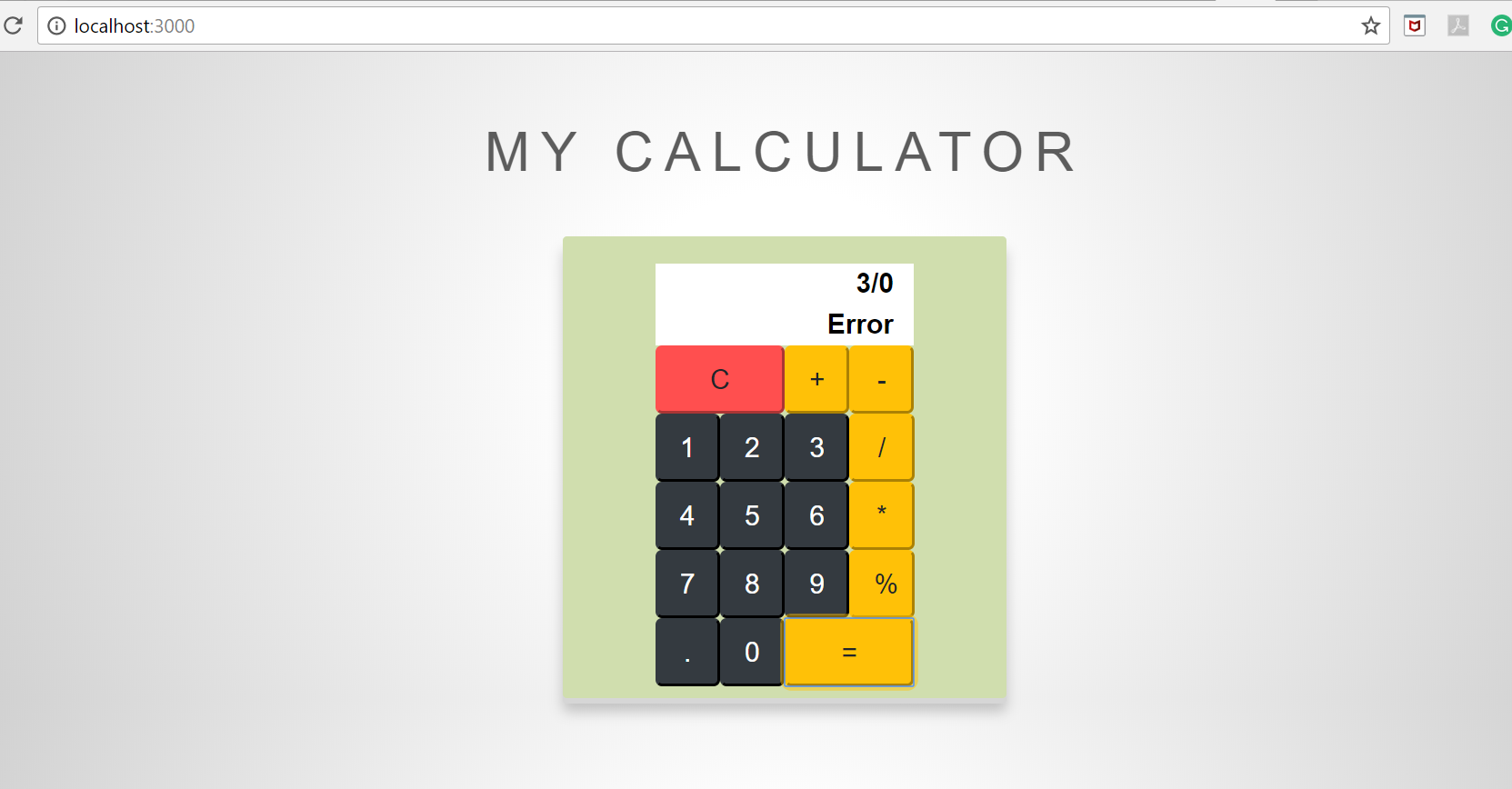


1. Modulo Operation



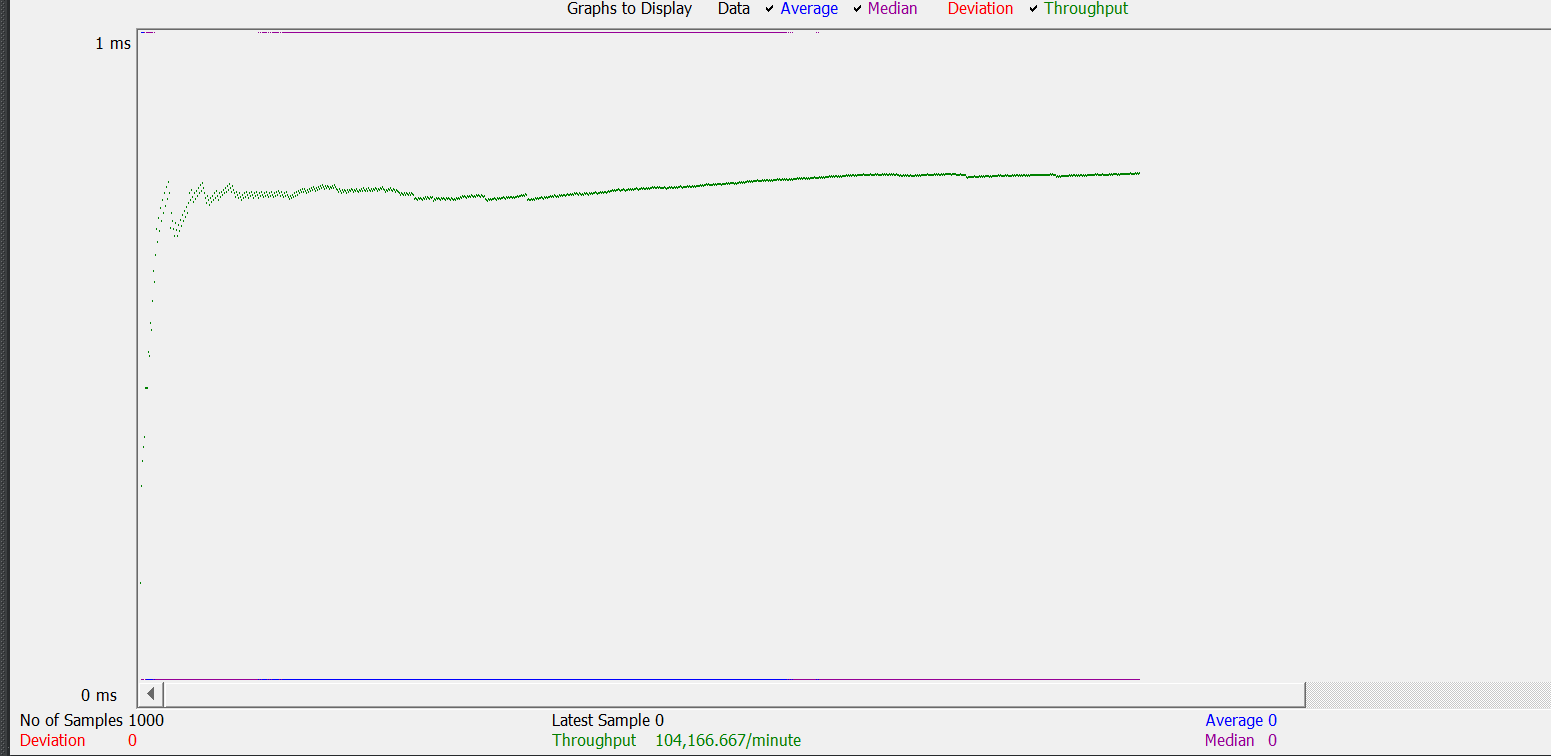
**Error and Exception Handling :**

* Return as Error when tried to divide any number by 0

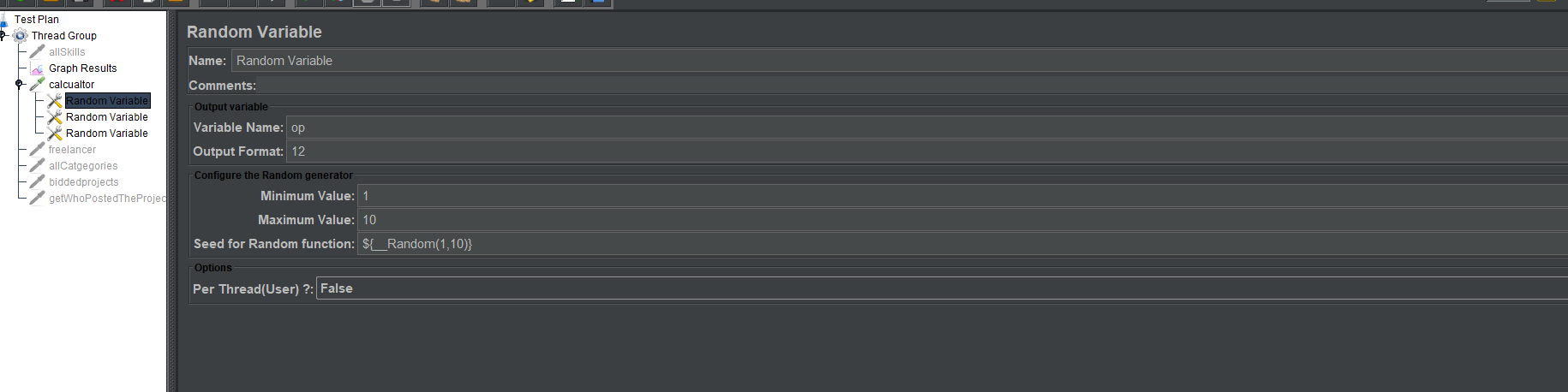


**JMeter Testing:**

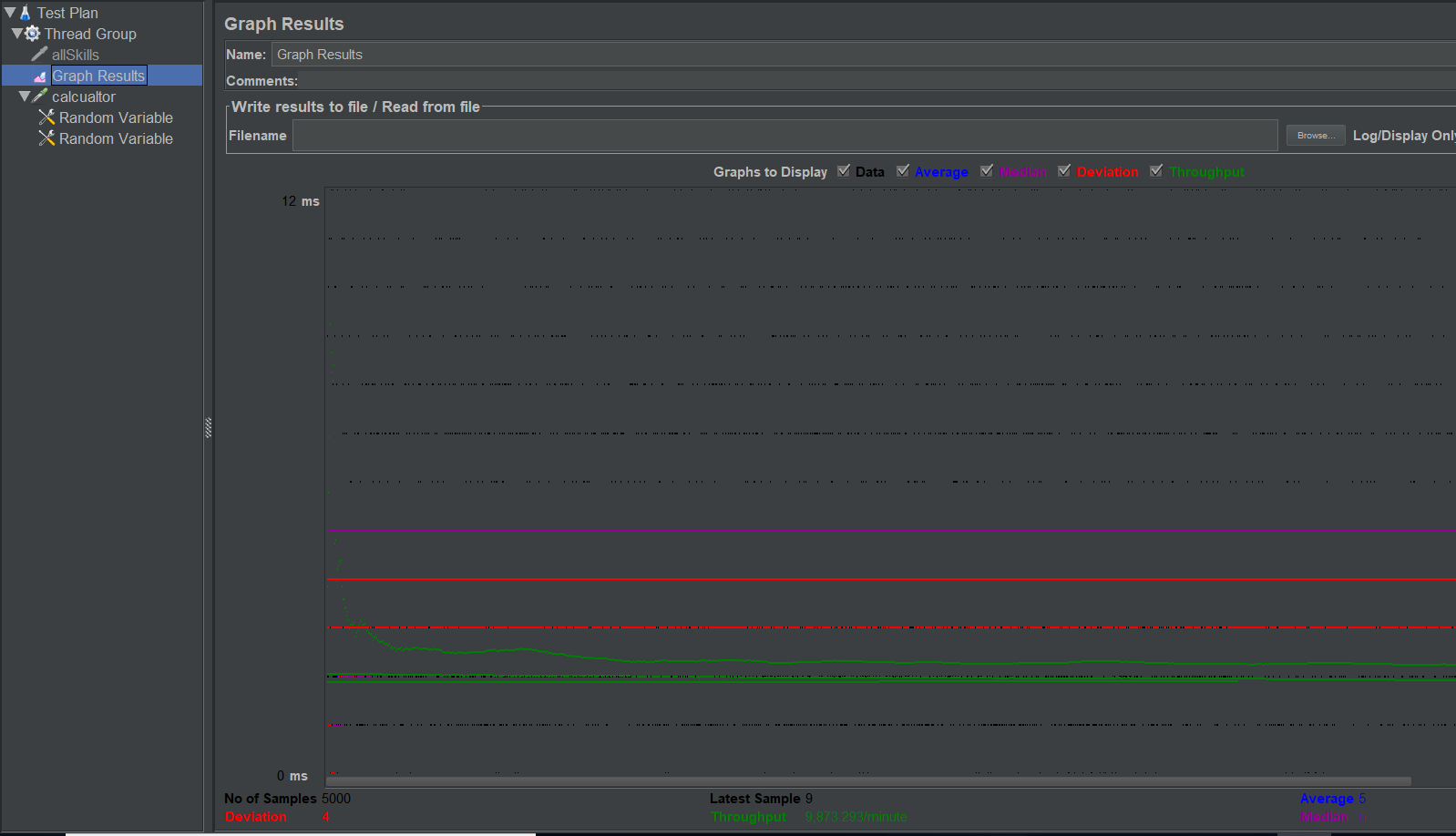
1. **1000 calls on randomly selected task (Average 0 )**



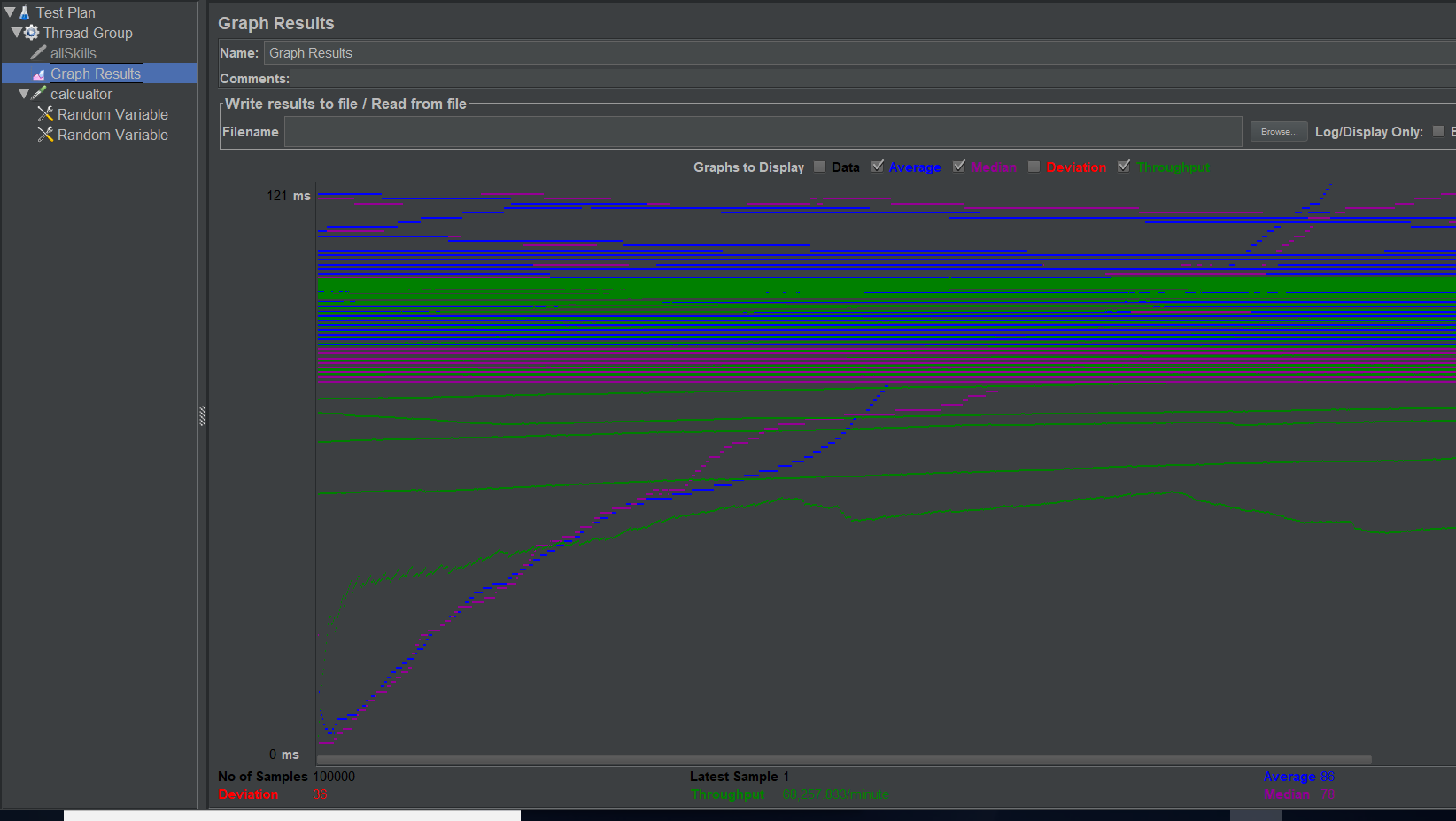
Random Variable Conf:



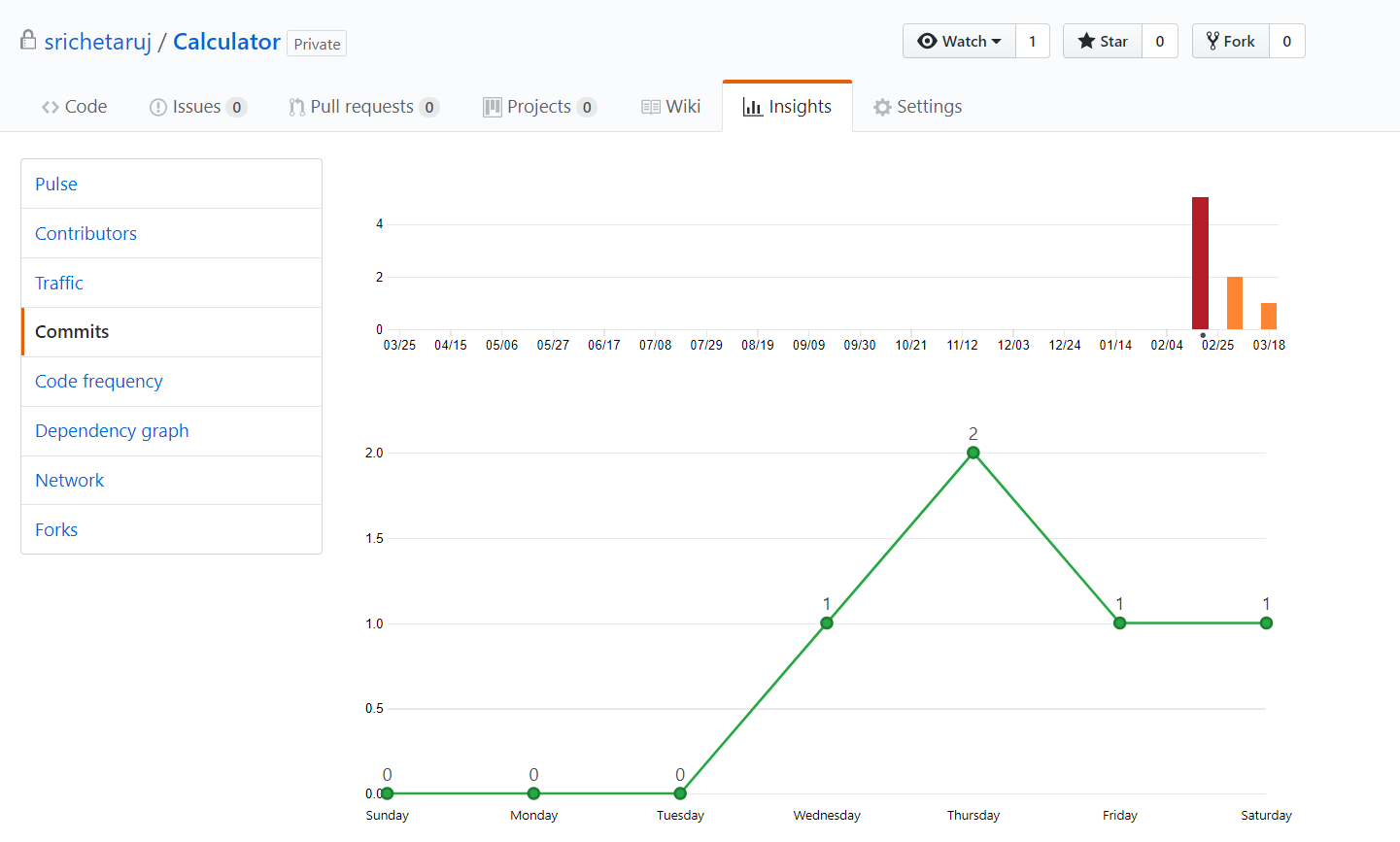
1. **5000 calls on randomly selected task (Average 5 )**



1. **100 threads with 1000 calls calculator random (Average 86)**



**Github commit history**



**(Part 2) Freelancer –**

**Goal:** To Develop Freelancer like scalable web application developed in MERN Stack

**Purpose of the system**: The purpose of the system is to develop a real-like Freelancer web application. The application helps professionals to get their work done by engineers simply by sitting at home. It has following features:

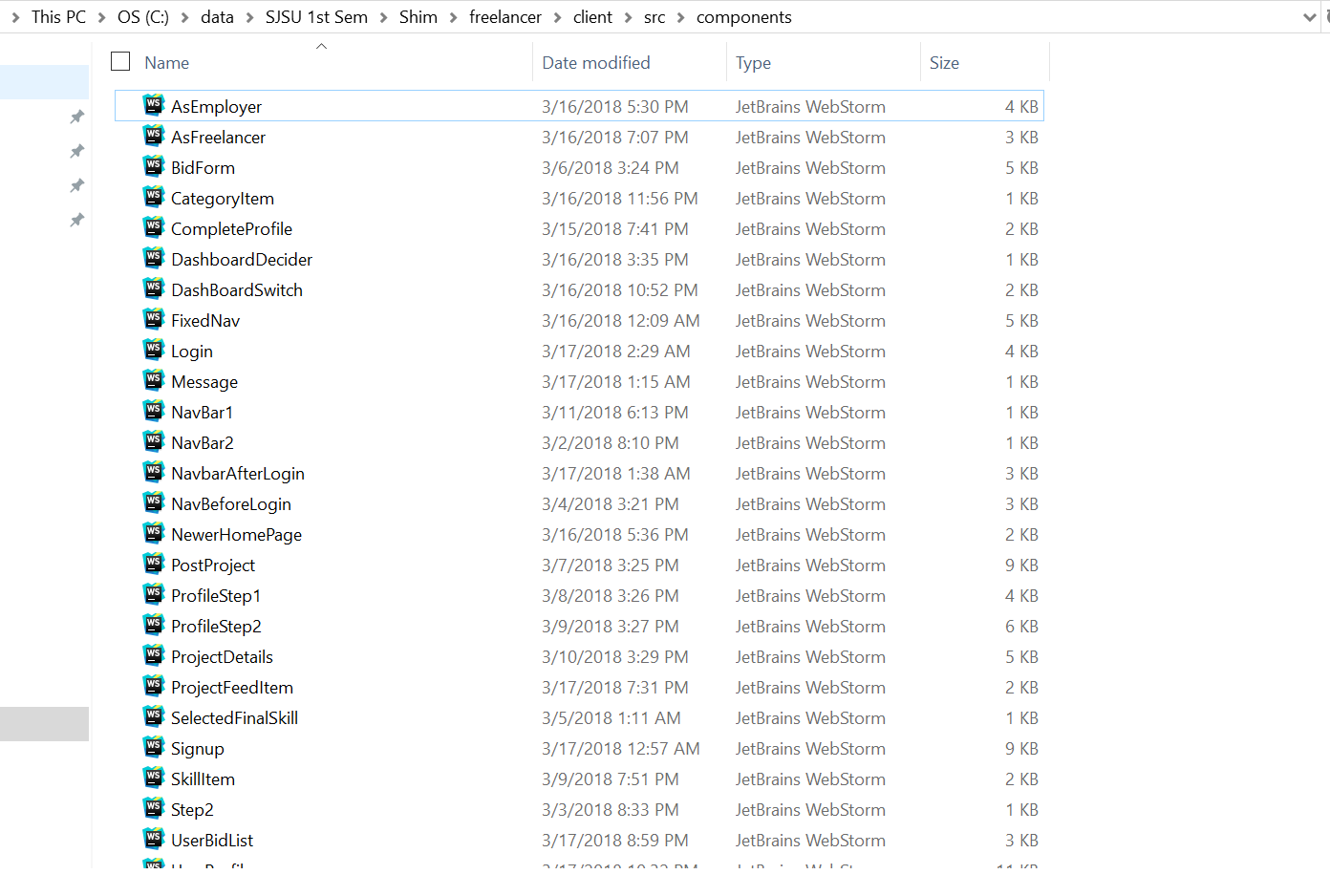
1. Signup as a Freelancer and as a Employer
2. Login
3. Set up profile
4. Dashboard (for Employer a well as Freelancer)
5. Post a Project (upload file)
6. Bid for a project
7. Home page View all open Projects

**Technology Used**:

**Client side:**

1. **React** as View in MVC architecture used reusable Components
2. **React-Bootstrap-Typeahead** to choose multiple skills
3. **Redux** for state maintenance.
4. Used **react-file-download** to interpret data stream.
5. **axios** to call the REST service.
6. Used **react-router-dom** to handle routing between different pages.
7. **React-StepZilla** to show progress view of complete profile page.
8. Used “**redux-thunk**” as middleware to support data flow between redux store and react components.

**Total of 26 Components Used :**



**Server side :**

1. Used **Mocha**, **Chai** and Supertest for unit testing of rest api’s
2. **Express** as web application framework.
3. Used **Apache JMeter** for load testing
4. **bcrypt** to encrypt password and store it in database
5. used **fs-extra** to read the file and send as a blob to client side to display image
6. Used **JWT** for session maintenance.

**Database Design:**

**Database used :** MySQL

**Tools used :** MySQL Workbench (Reverse Engineering to draw Database design diagram)

**Tables used:**

1. **User :** has all the information of a person to registers in the website
2. **project\_user :** has information of users associated with a project by what role (freelancer /Employer)
3. **skill\_user :** has information of users associated with a skill.
4. **project\_bid :** has information of users associated with a project on which they are interested.
5. **skill\_category :** has information of skills associated with a particular category.
6. **project\_document :** has information of files associated with project.
7. **skill :** has all the skills
8. **project\_skill :** has information of project associated with a skill.
9. **project :** has information of all the projects.

**The database design has been normalized up to 3NF.**

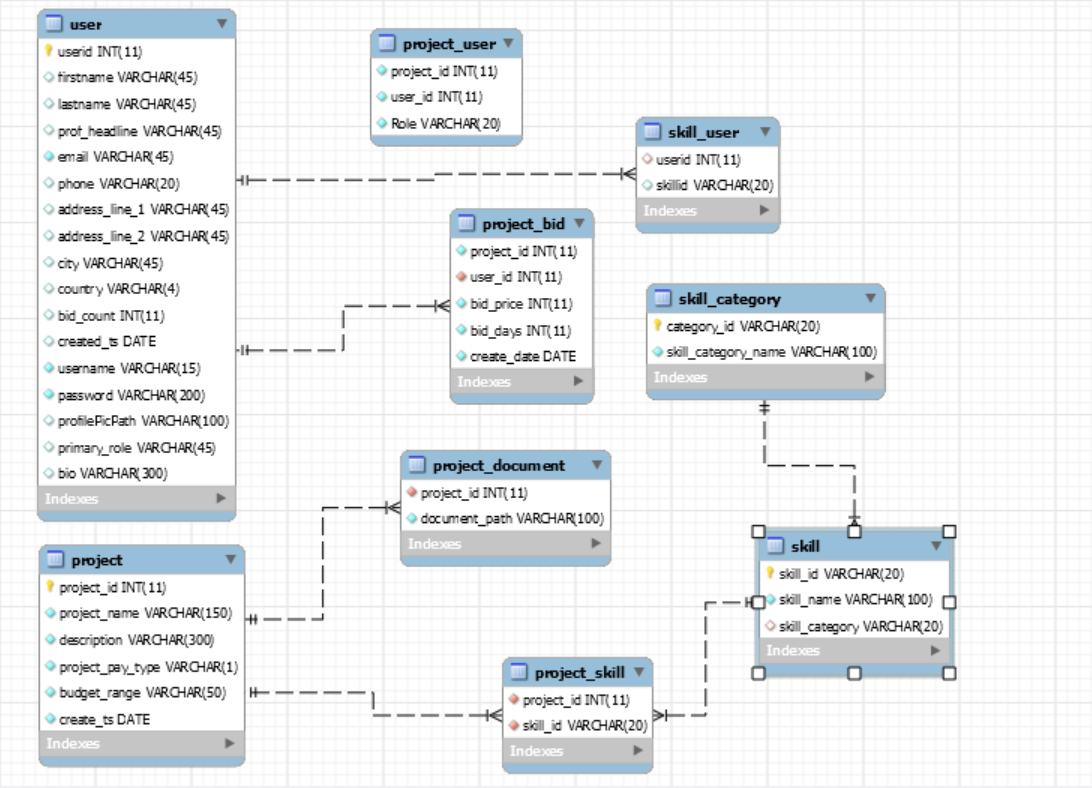
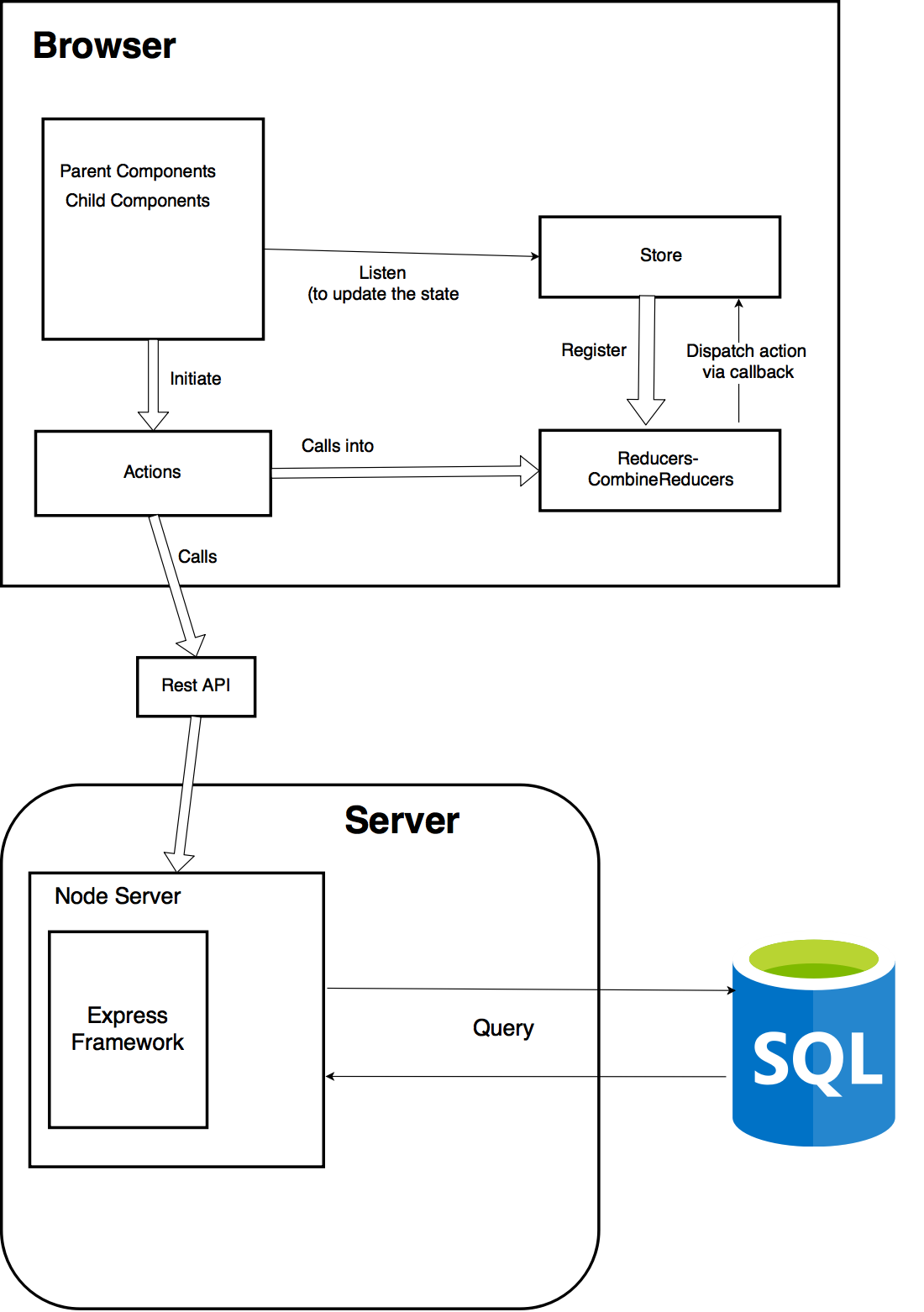


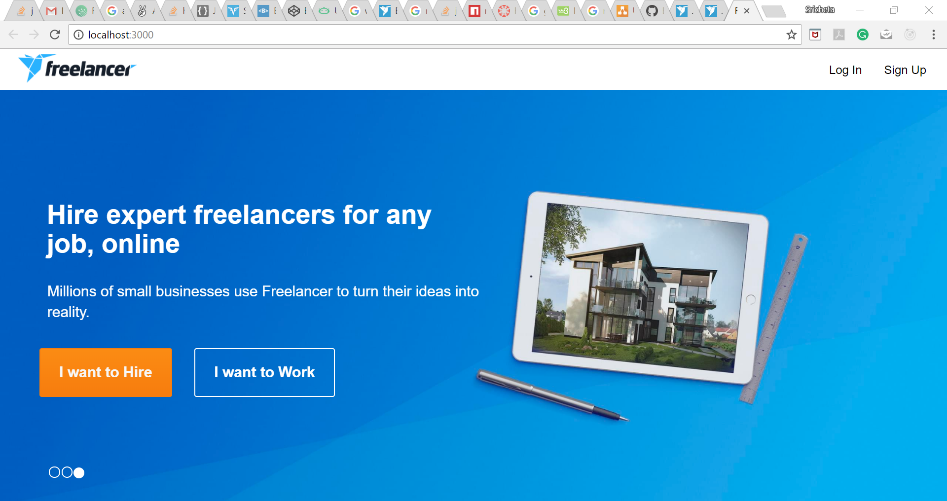
Figure 1 Made using Reverse Engineering tool in MySQL Workbench

**Architecture Diagram** (made using draw.io)

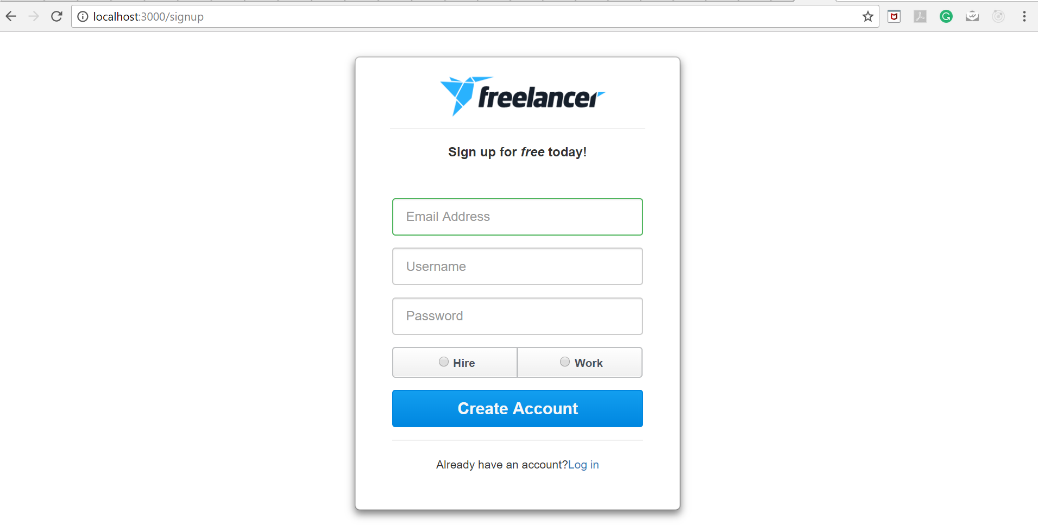


**Functionality :**

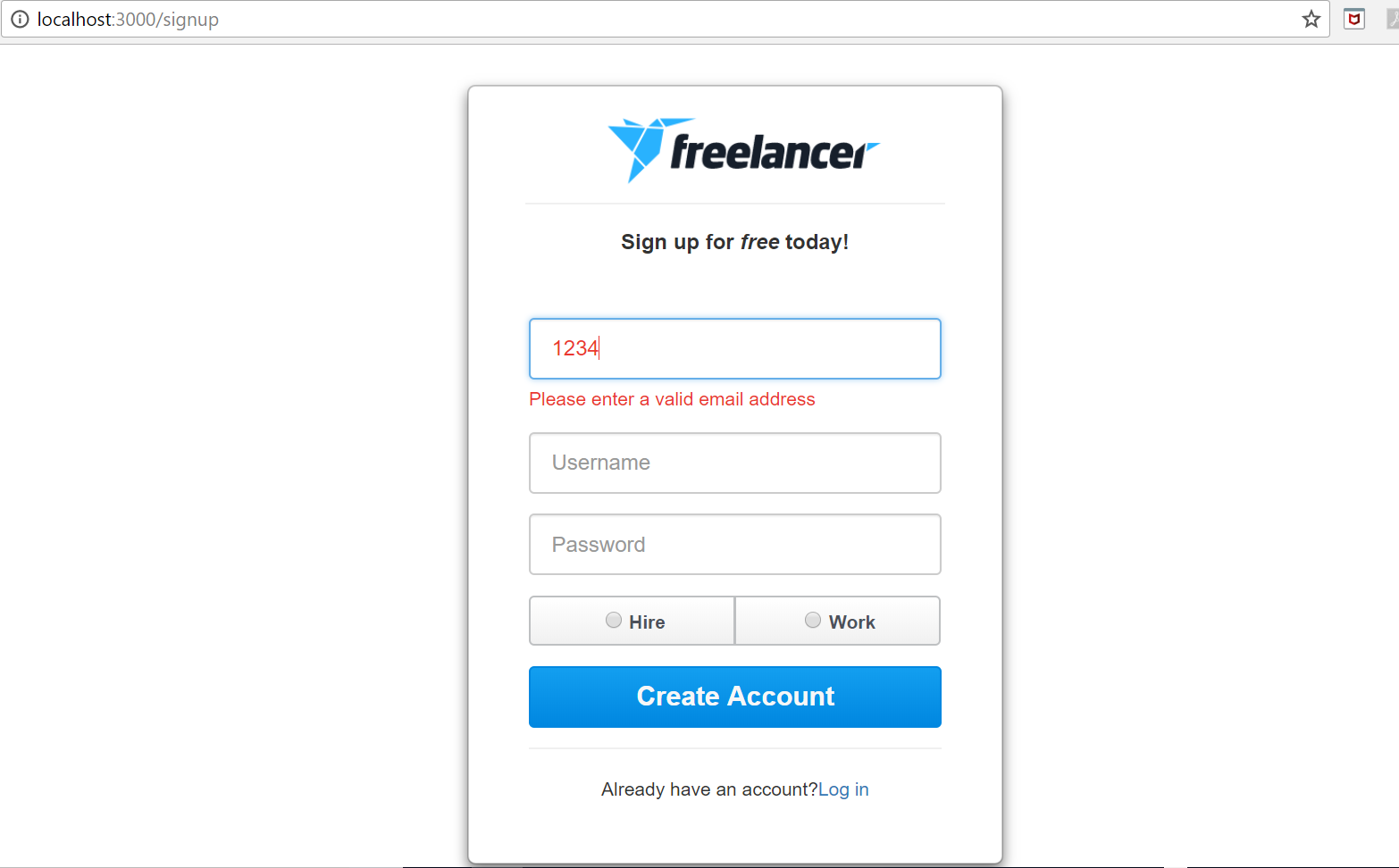
1. **Landing Page** has 2 options to login and signup



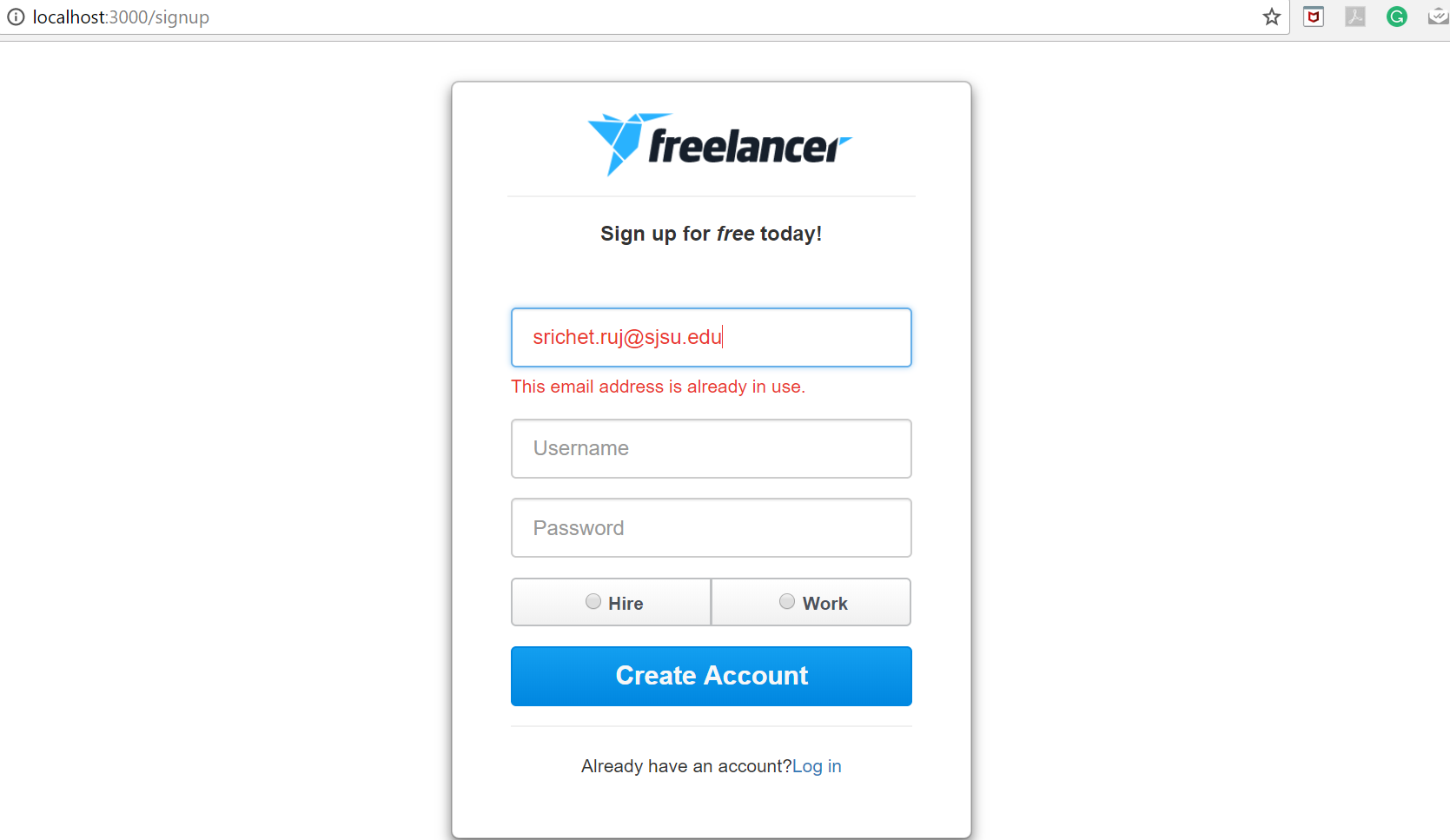
1. Signup page landing



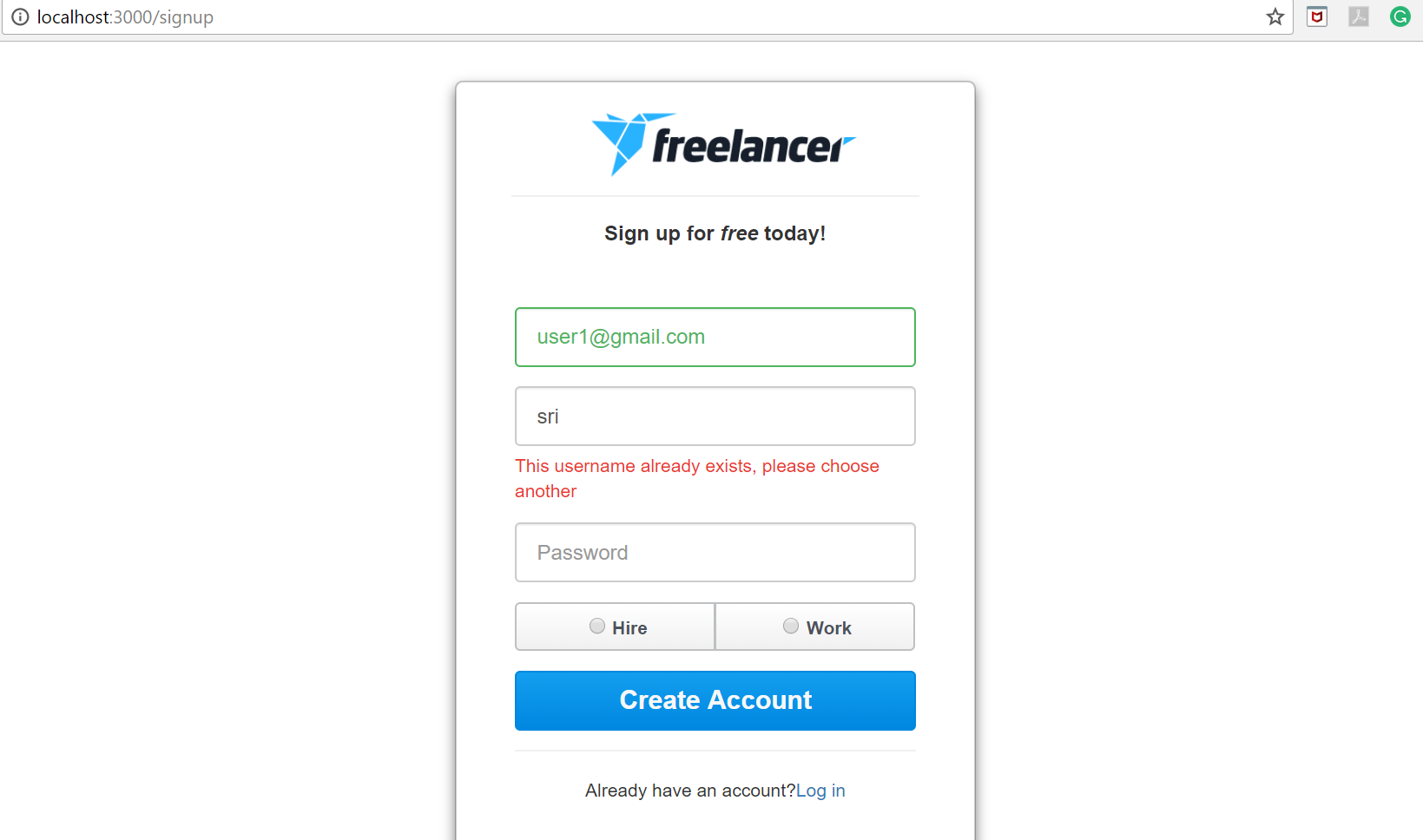
1. Signup page validation **(Invalid Email)**



1. Signup page validation **(Email already in use)**



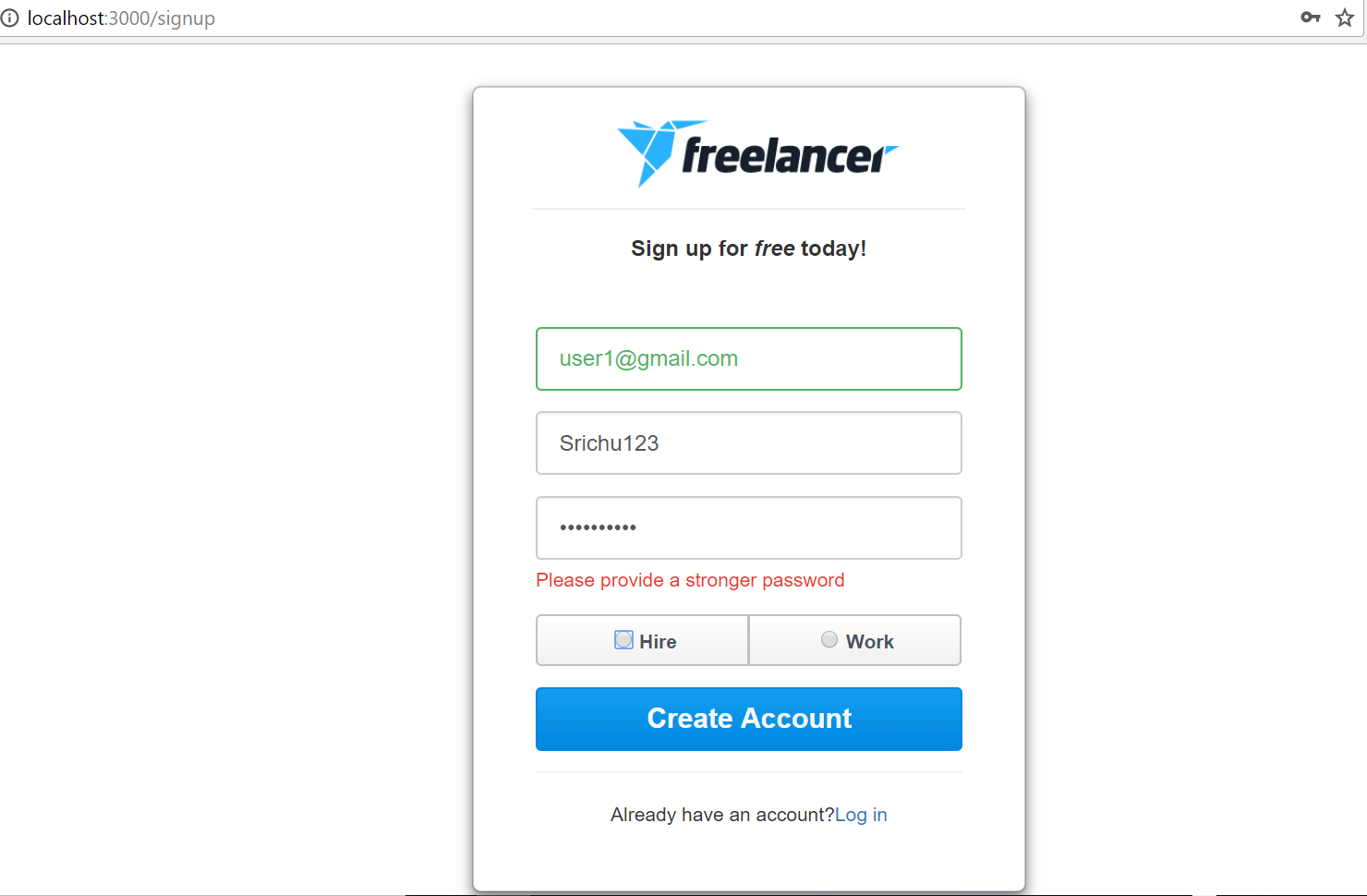
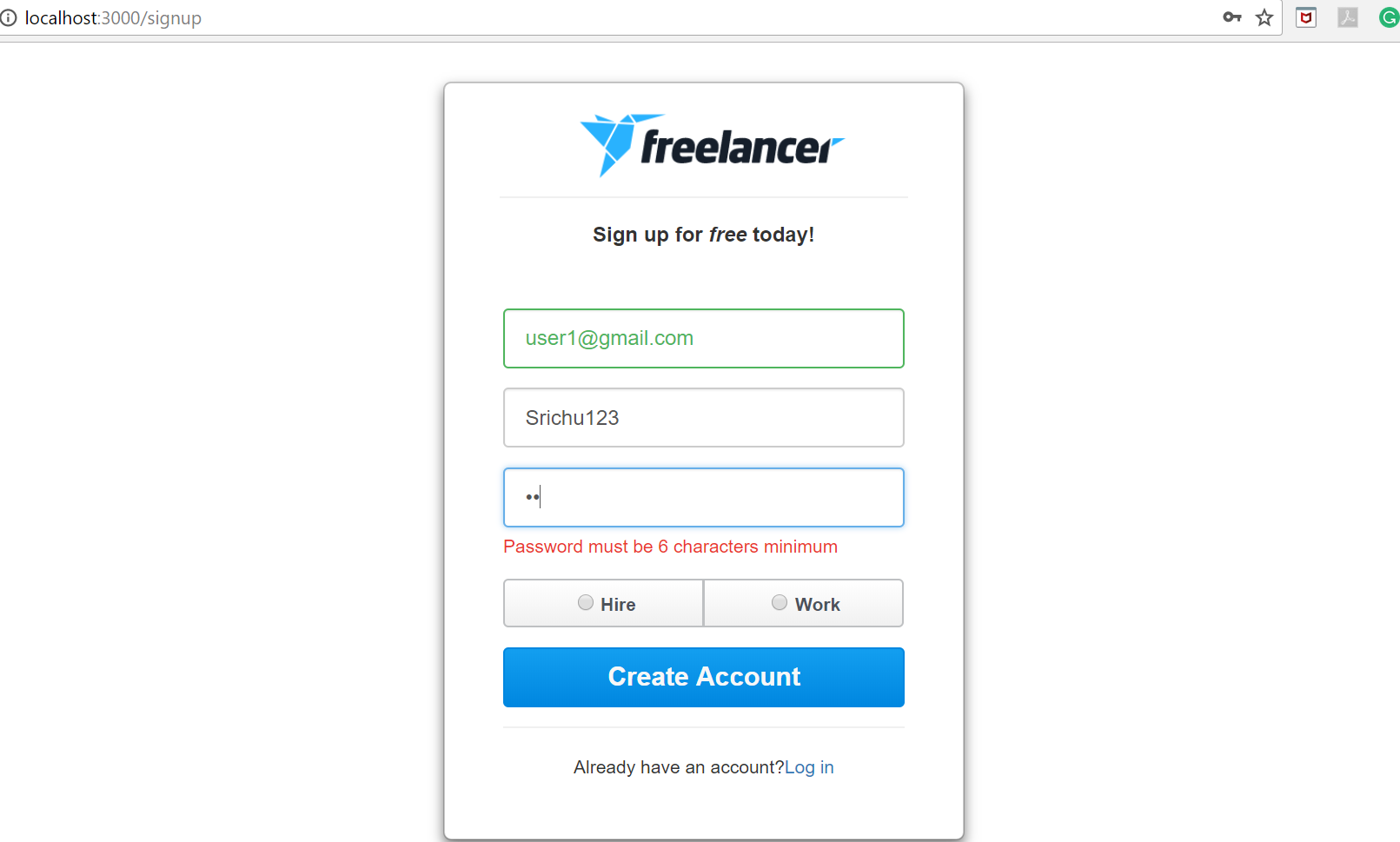
1. Signup page validation **(Username already in use)**



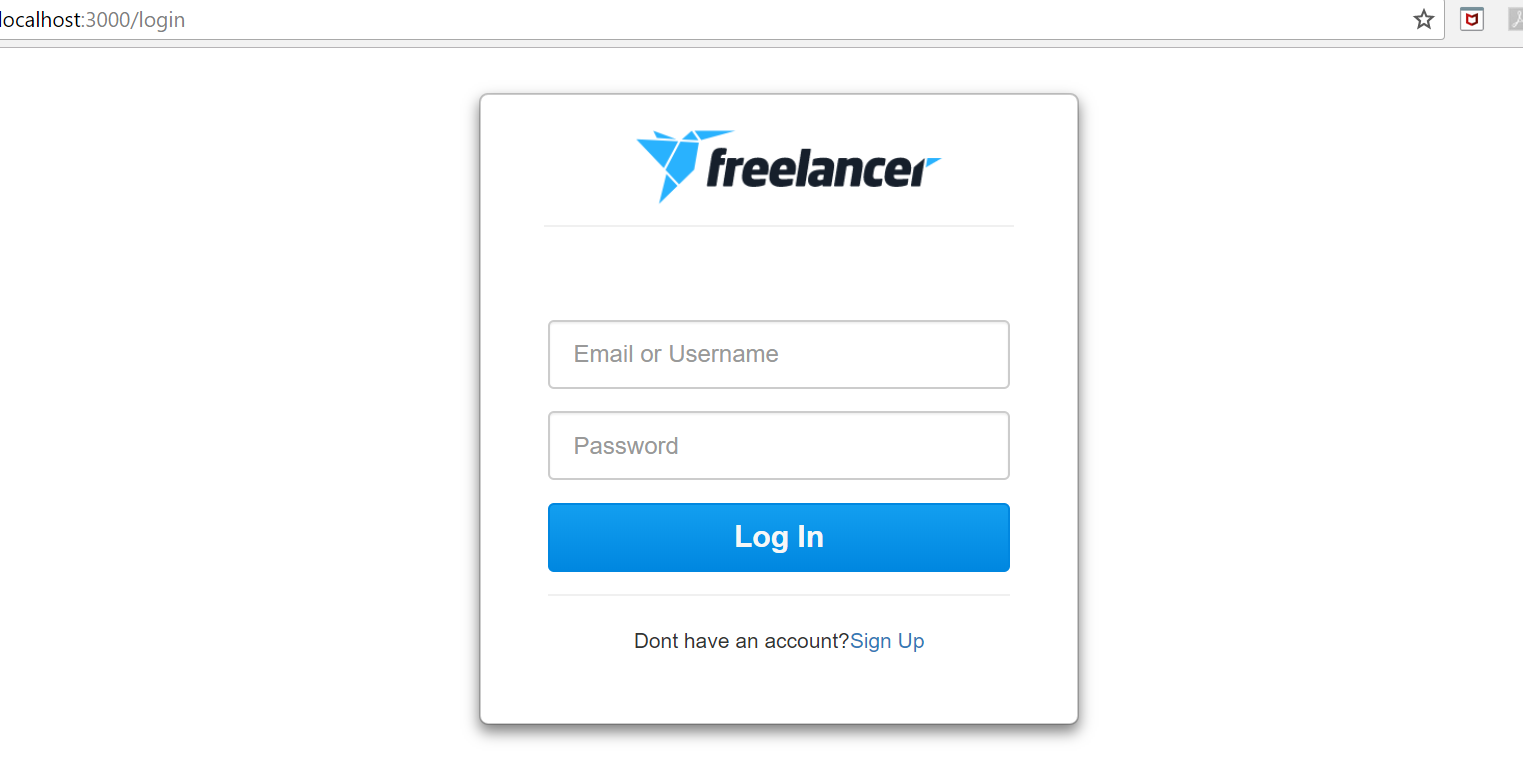
1. Signup page **(Username criteria validation)**



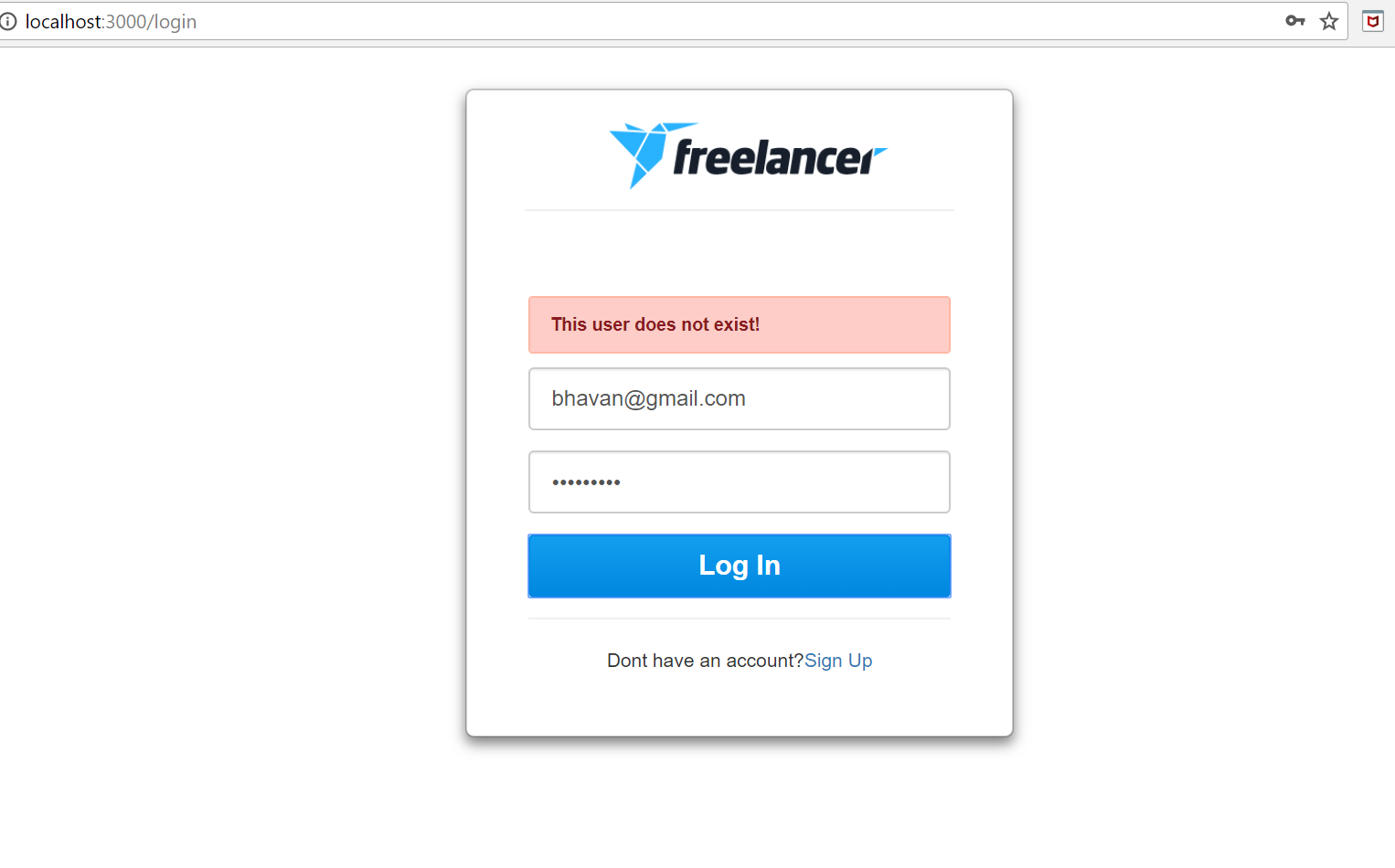
1. Signup **( strong password validation )**



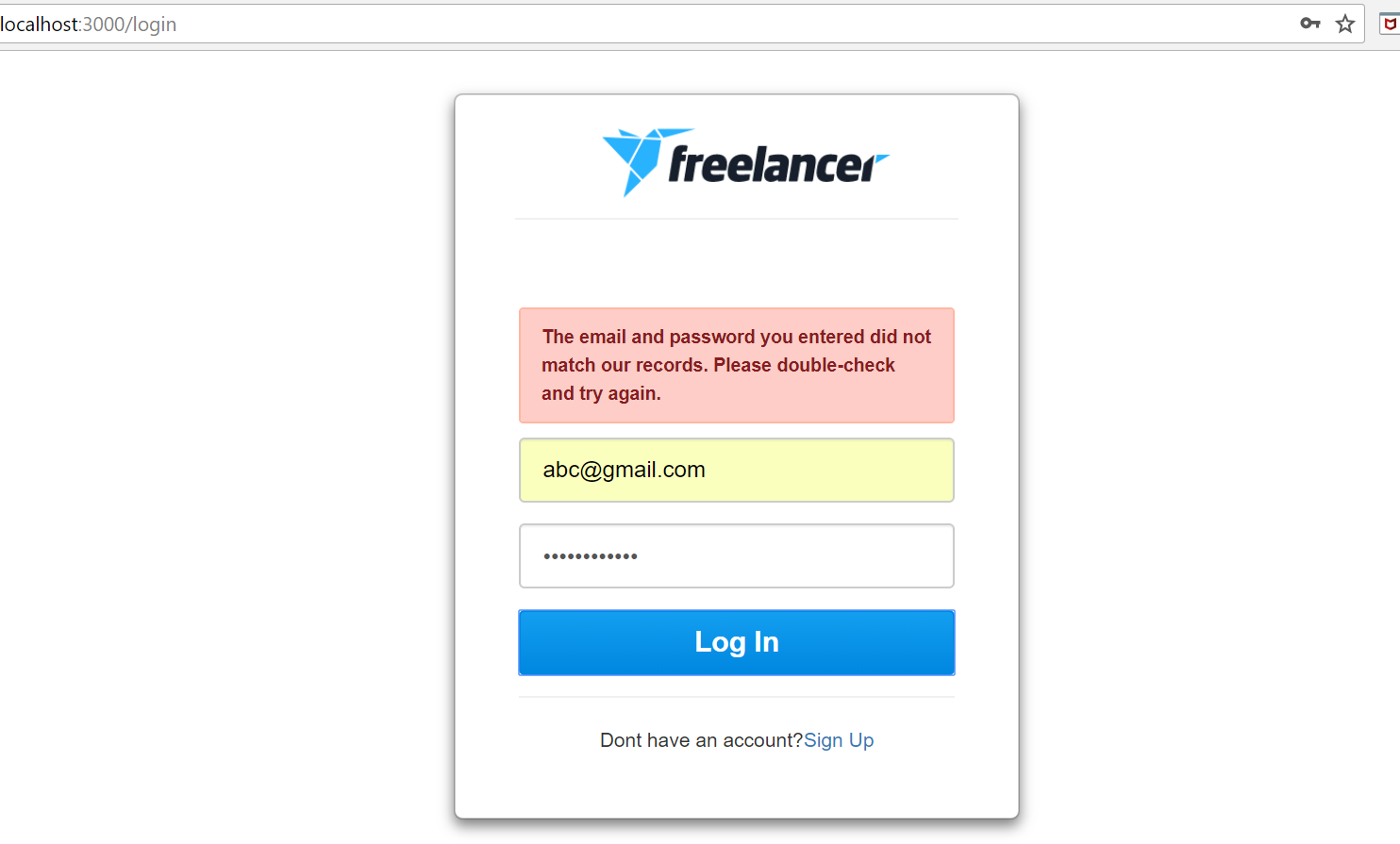
1. **Login page (login either with username or email )**



1. **Login Page validation (when user does not exist)**

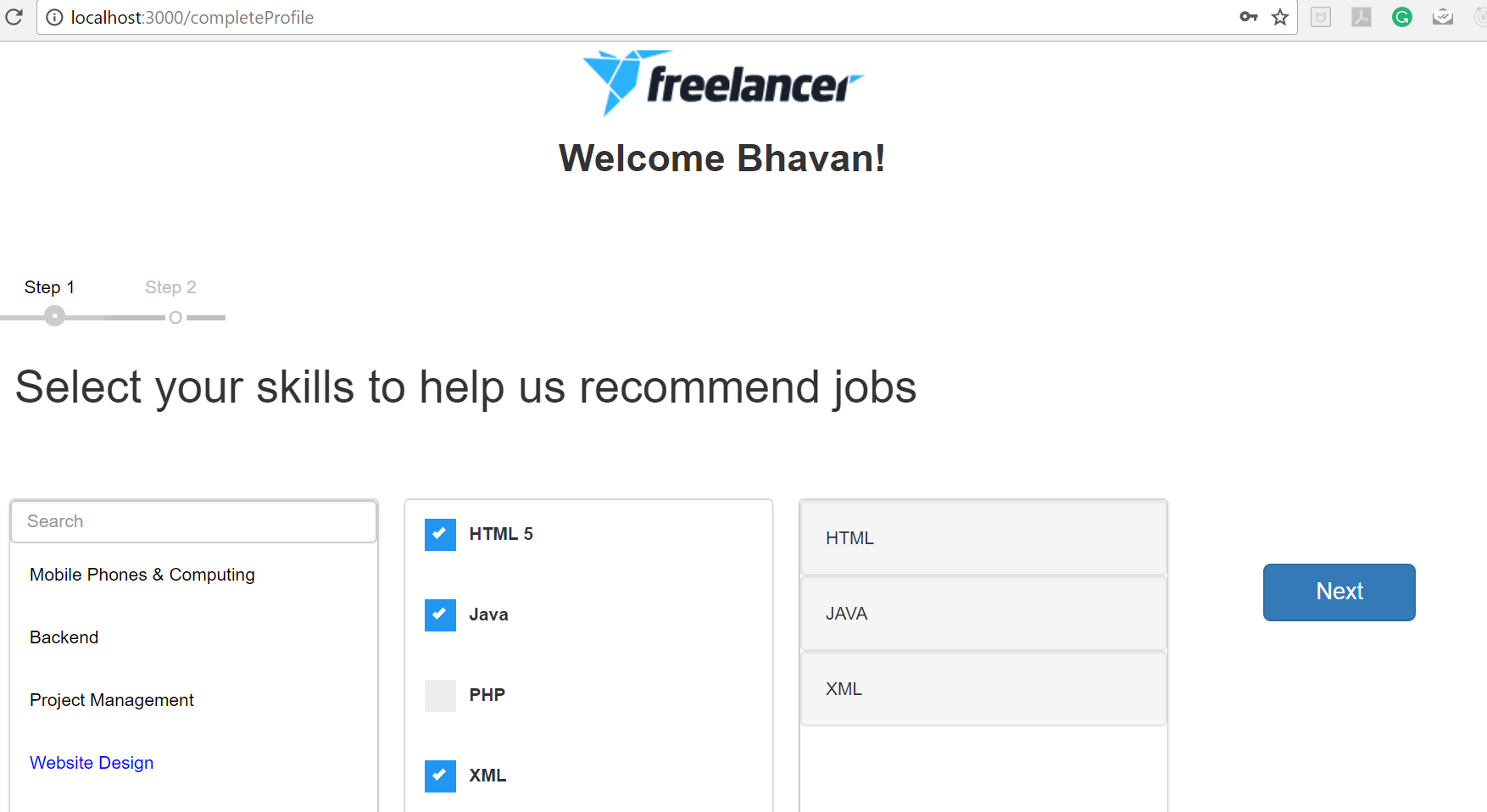


1. **Login page validation (When password entered is incorrect)**

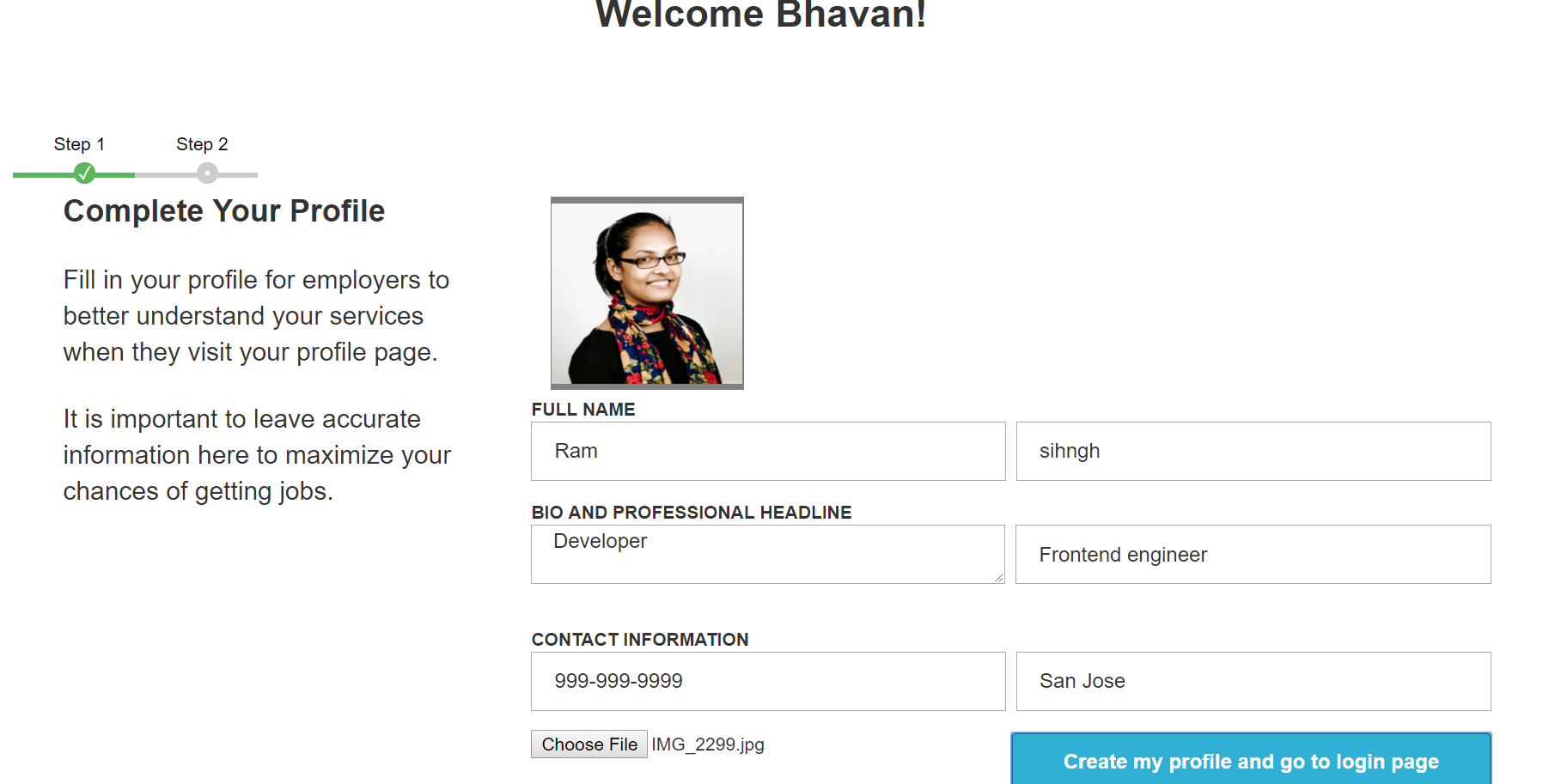


1. **After Sign up if clicked on “Work” radio button the sign up form navigates to complete form which asks to submit skills as per category:**

**There is a progress par to show the steps completed**

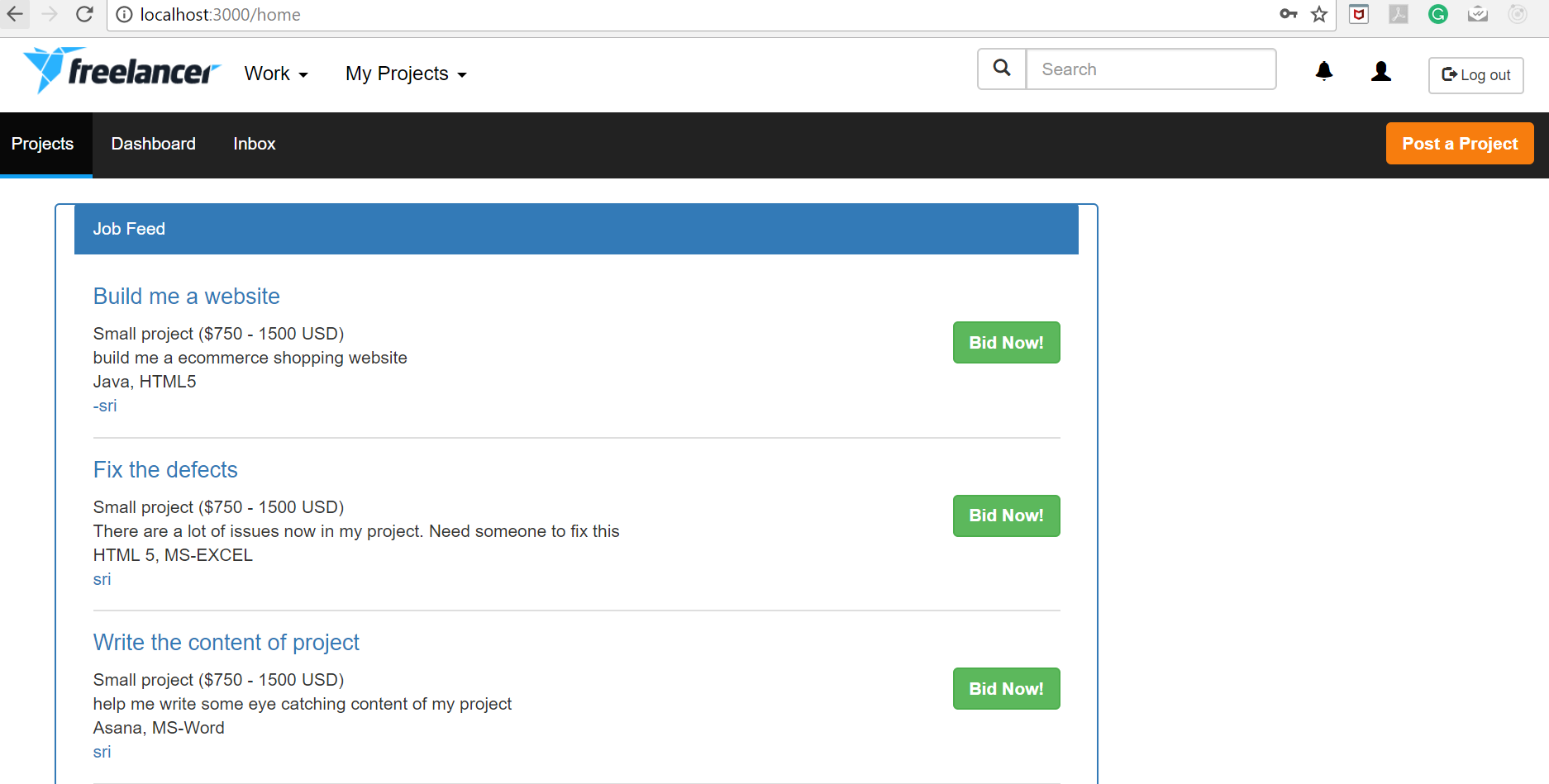


1. **Step 2 of complete profile**

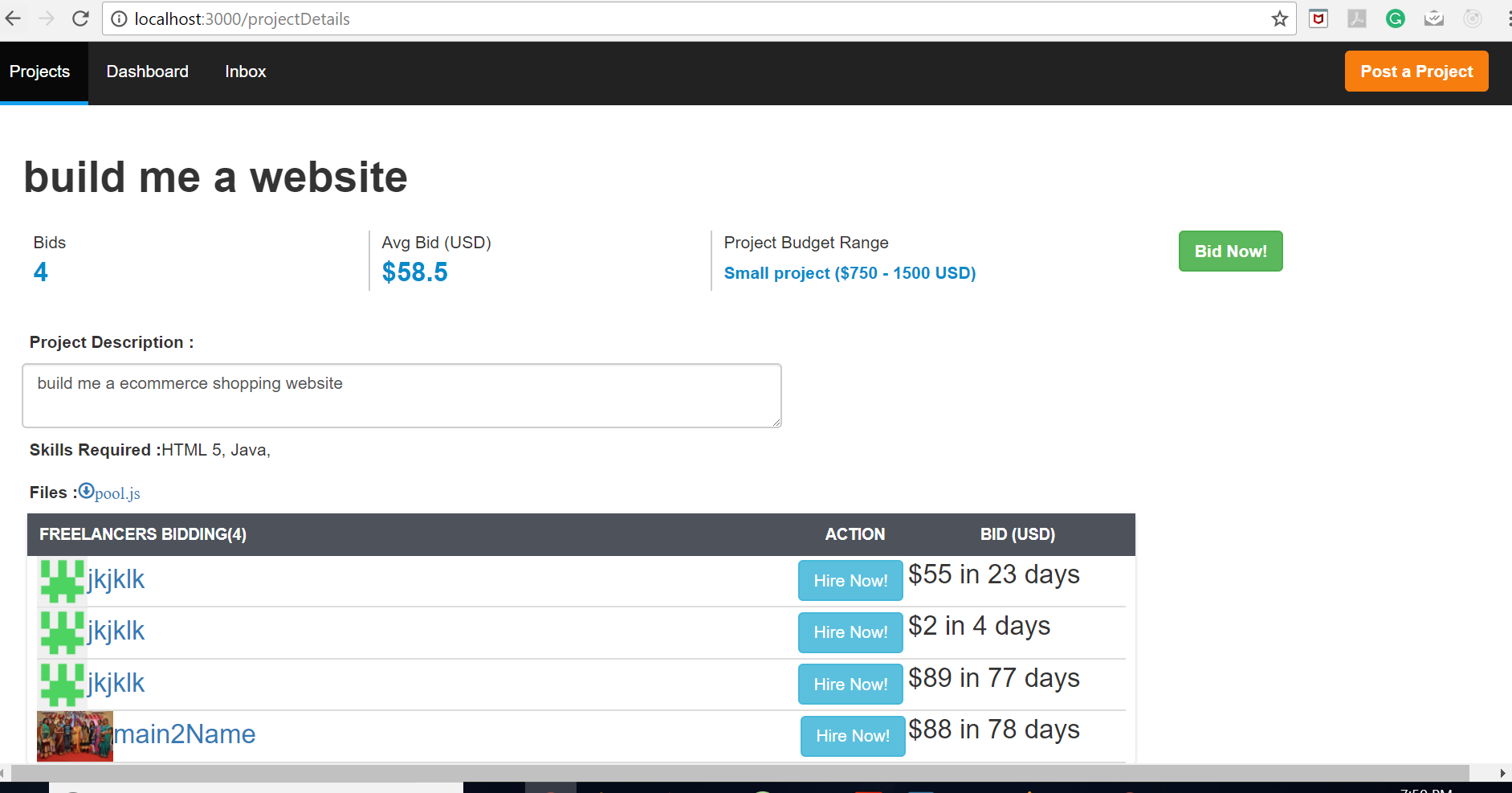


1. **After complete profile login and see the home page which has list of open projects**

**(username**



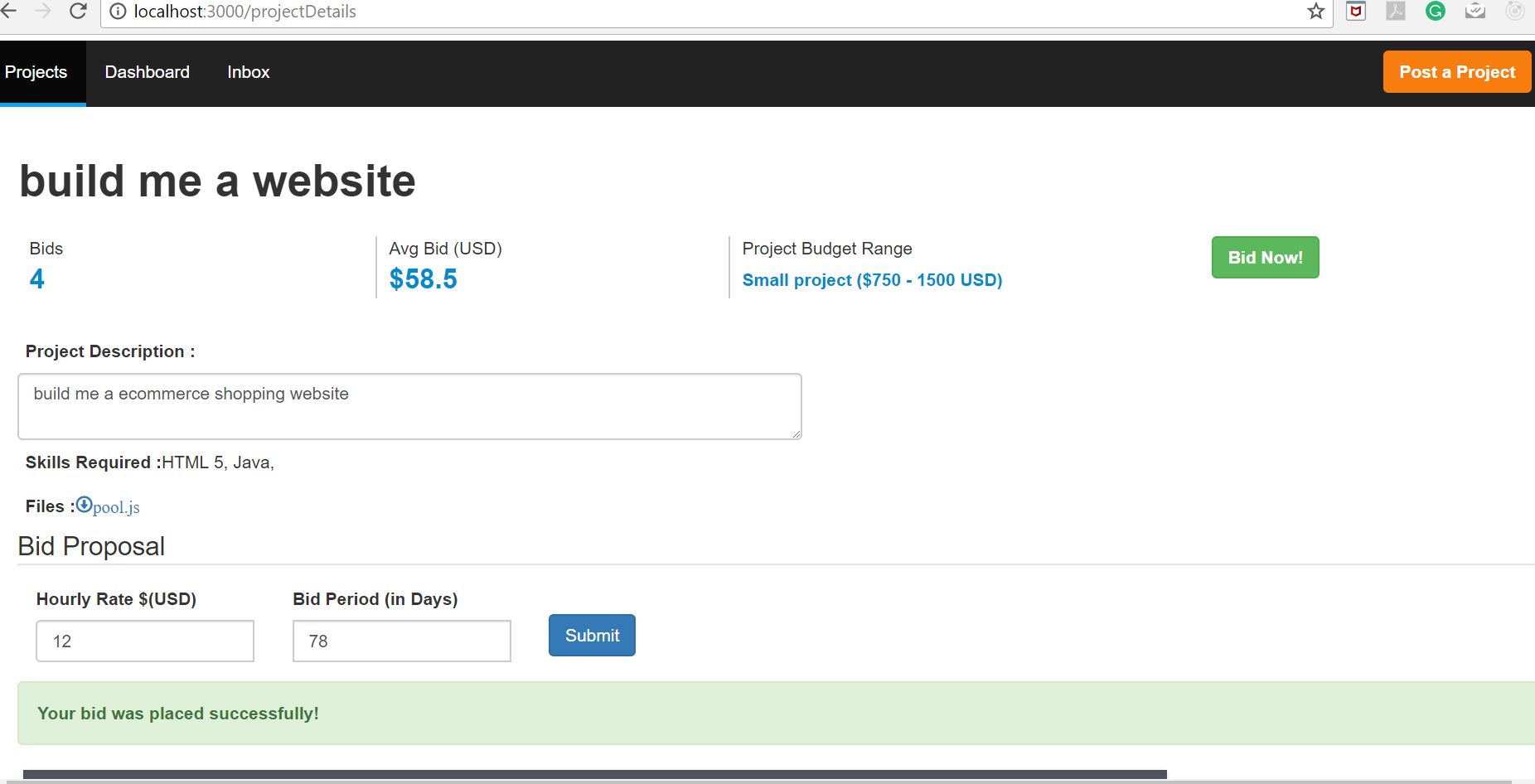
1. **Click on the project name link to view the Project details screen ( count of bids, average bids, description, file download, number of users bidding)**



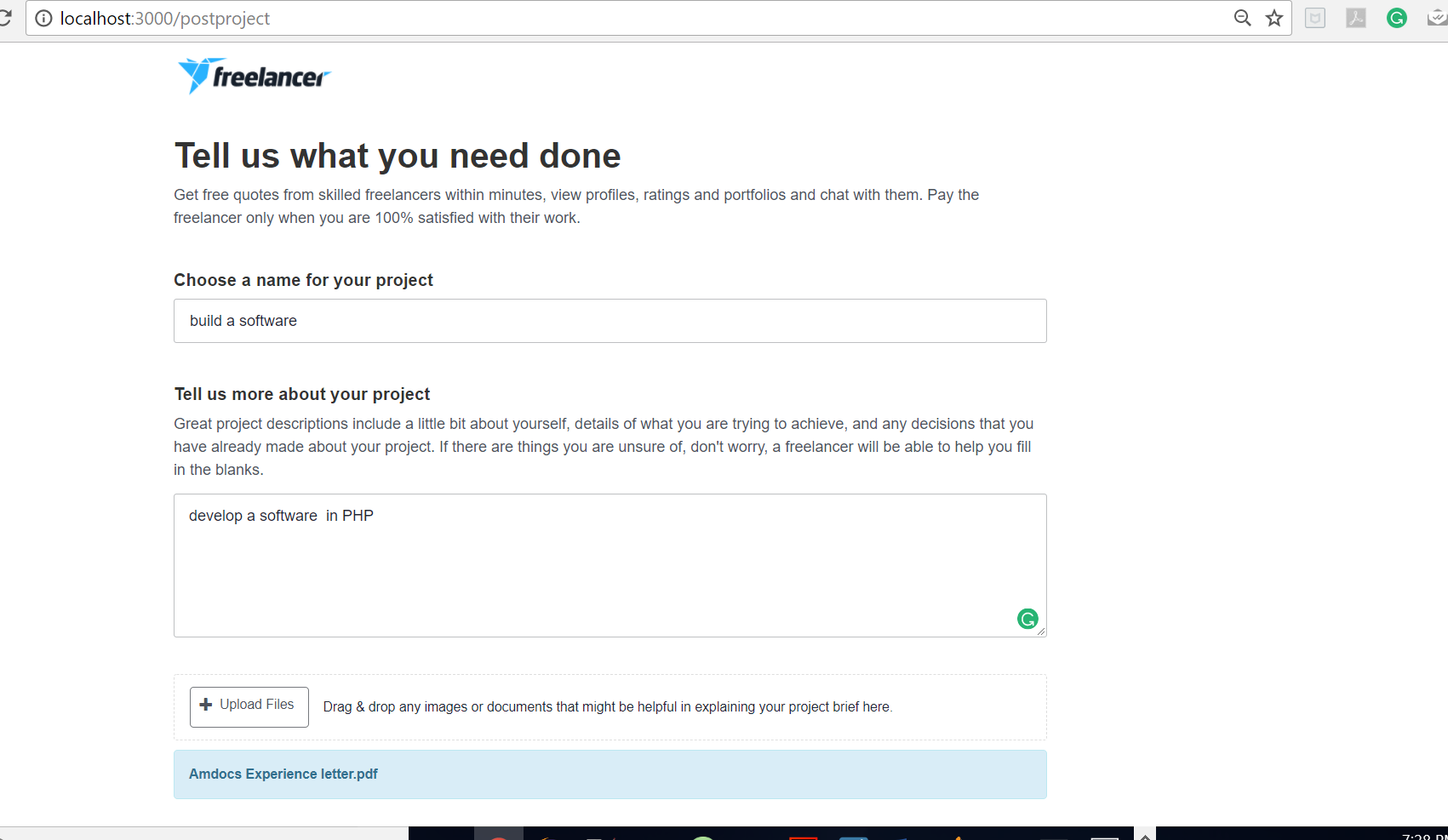
1. **Bid Proposal (toggle on Bid Now button )**

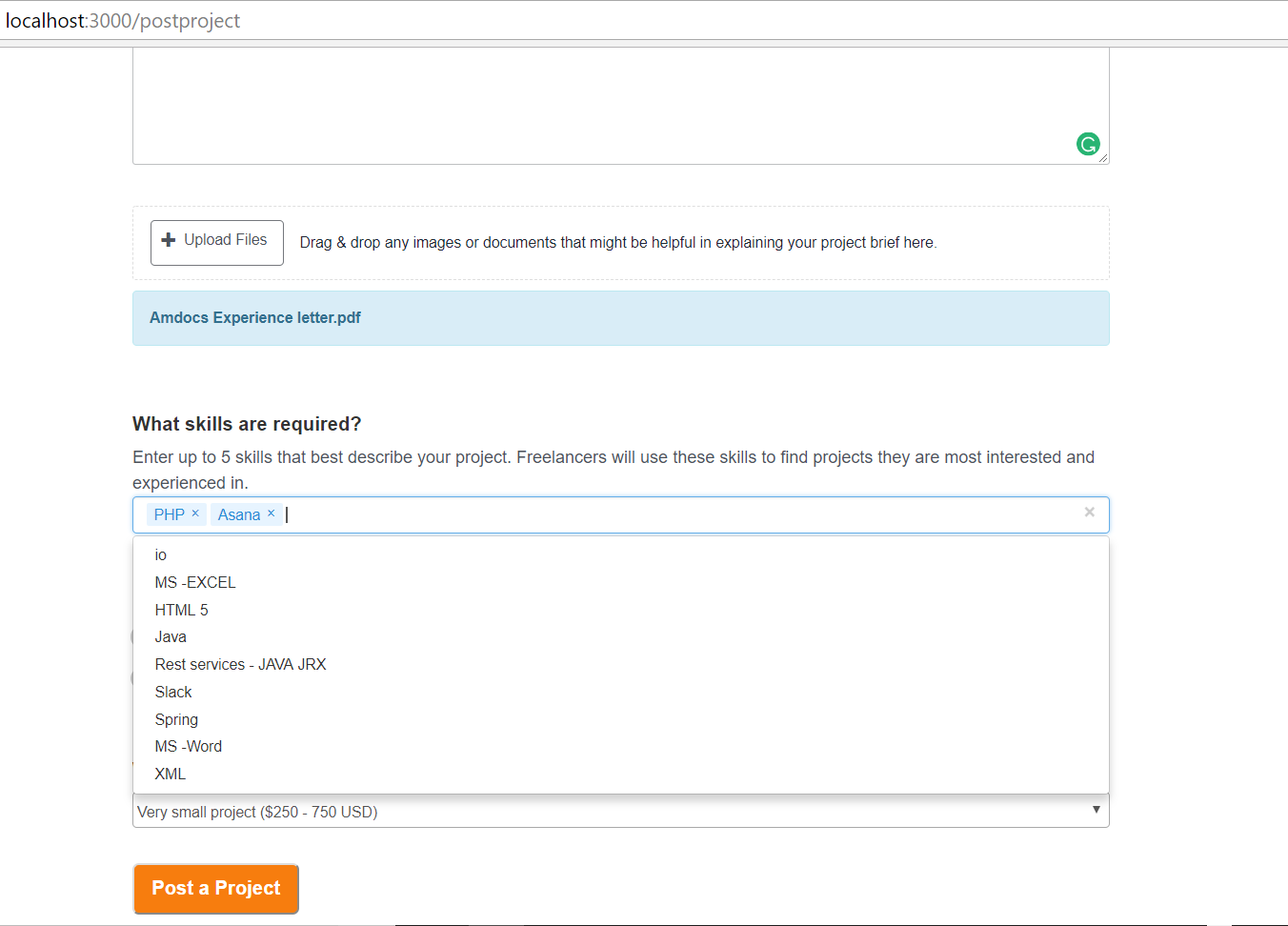


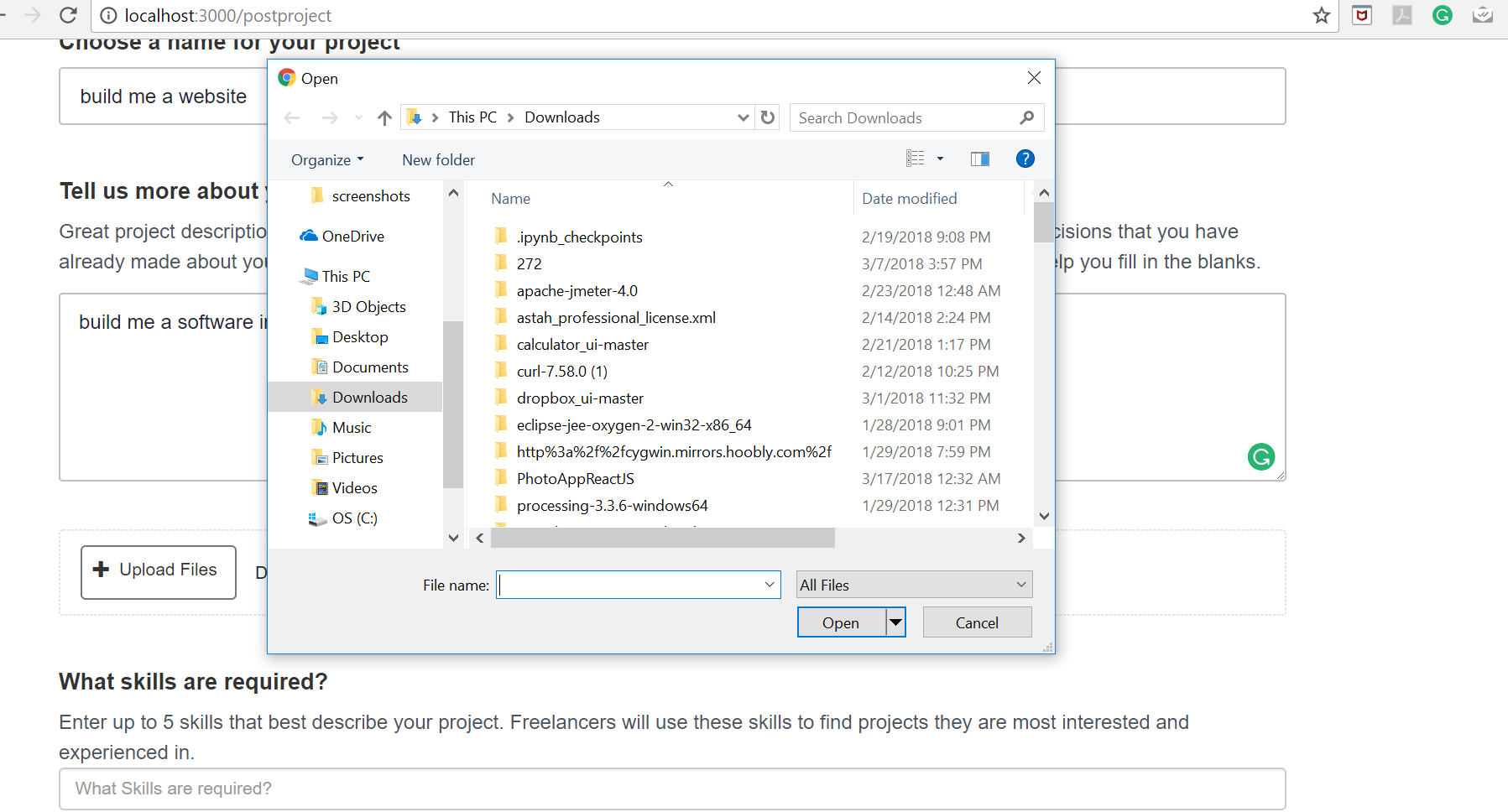
1. **Bid Proposal submitted successfully**

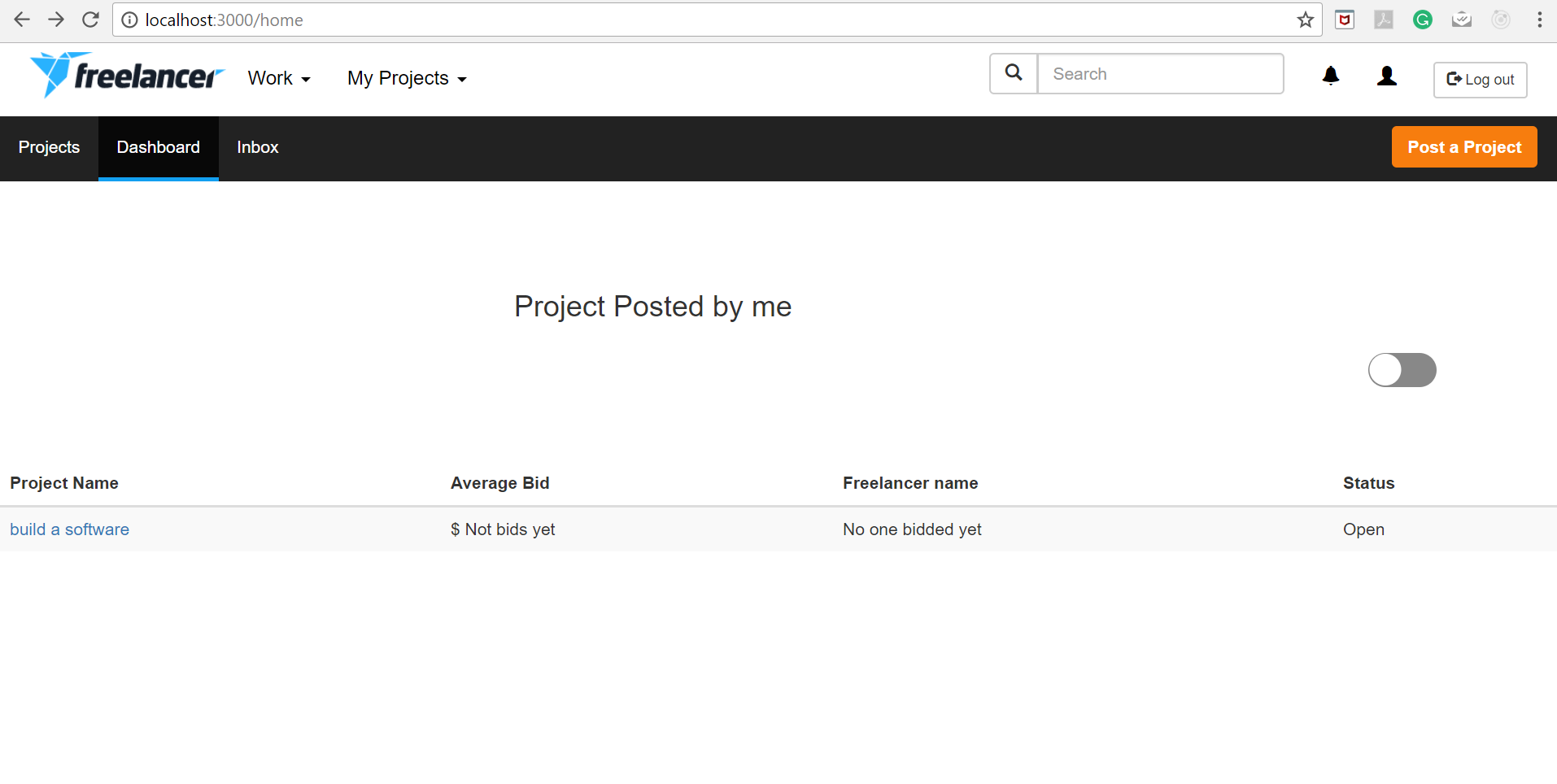


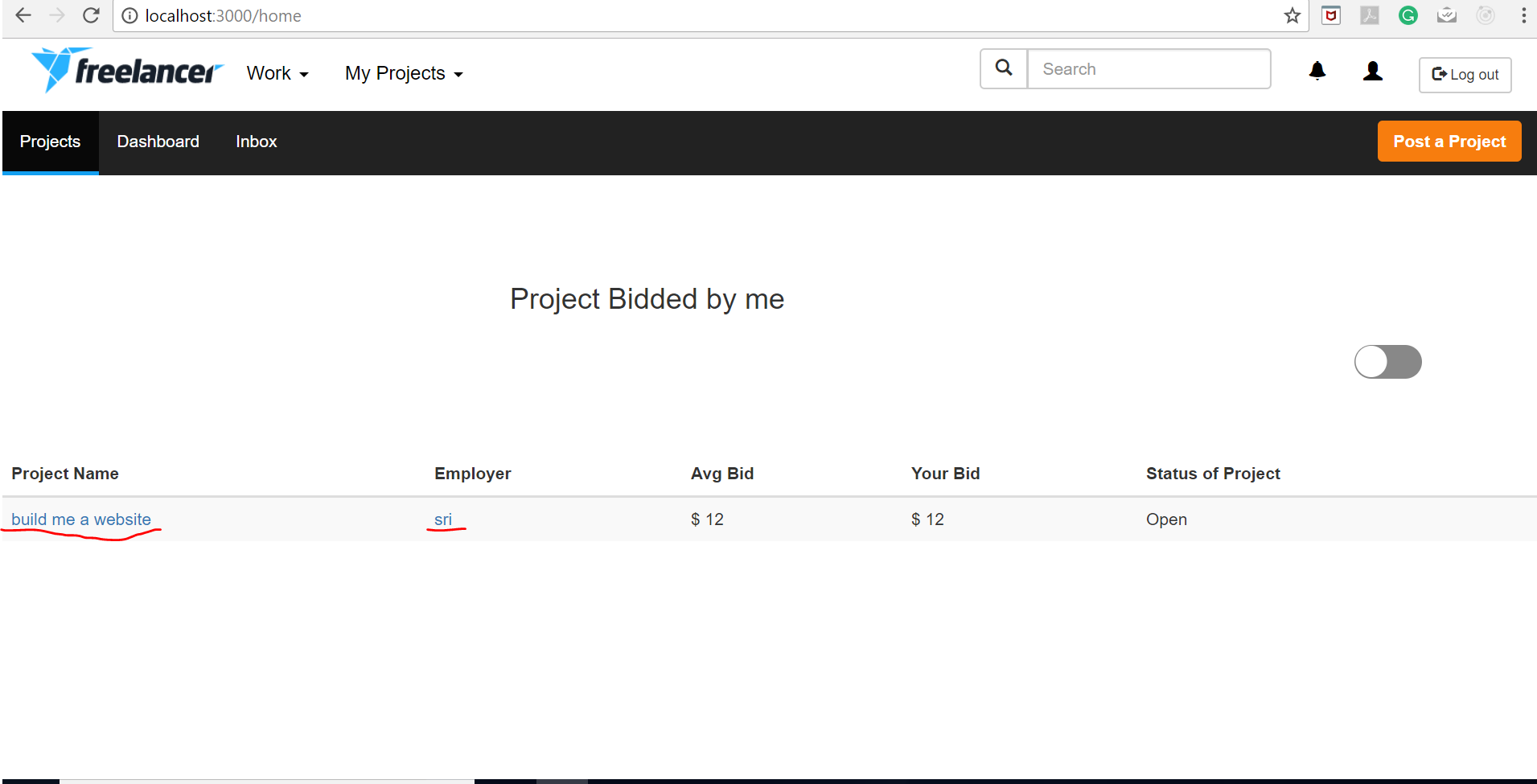
1. **Post Project Page ( including upload files )**



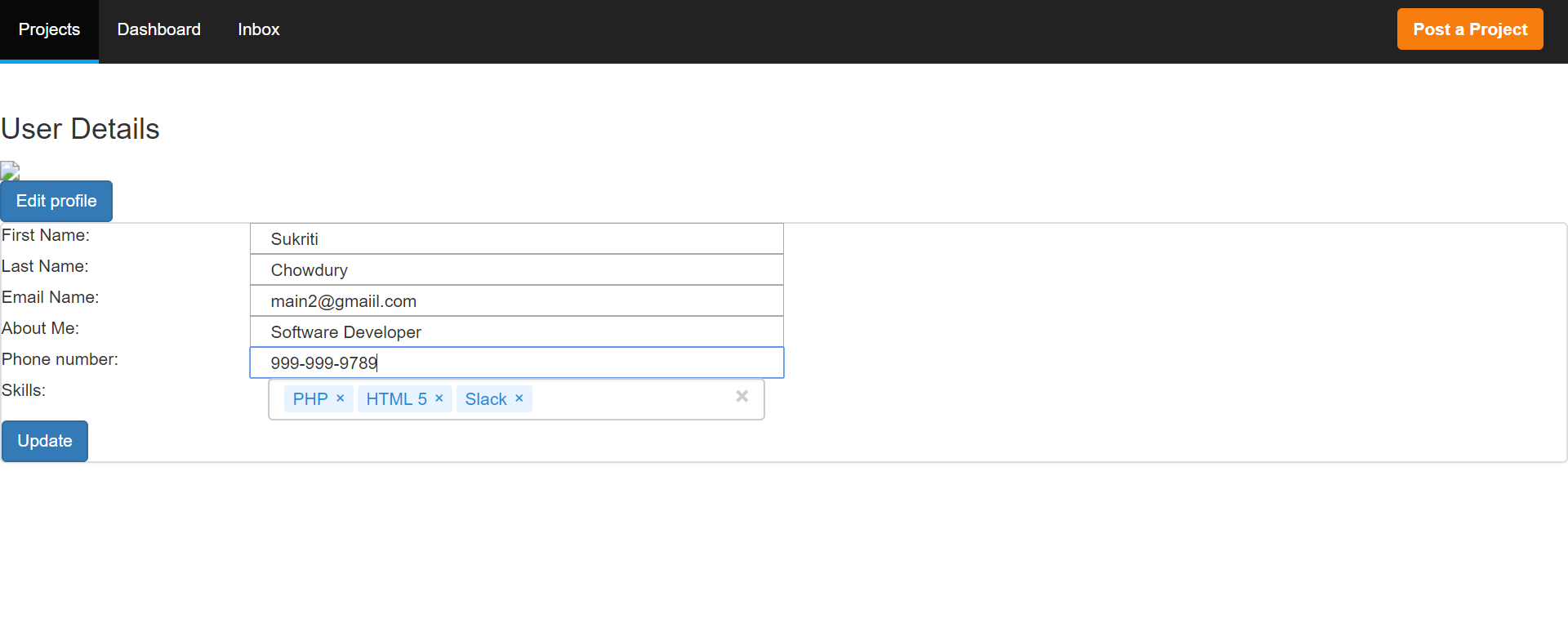


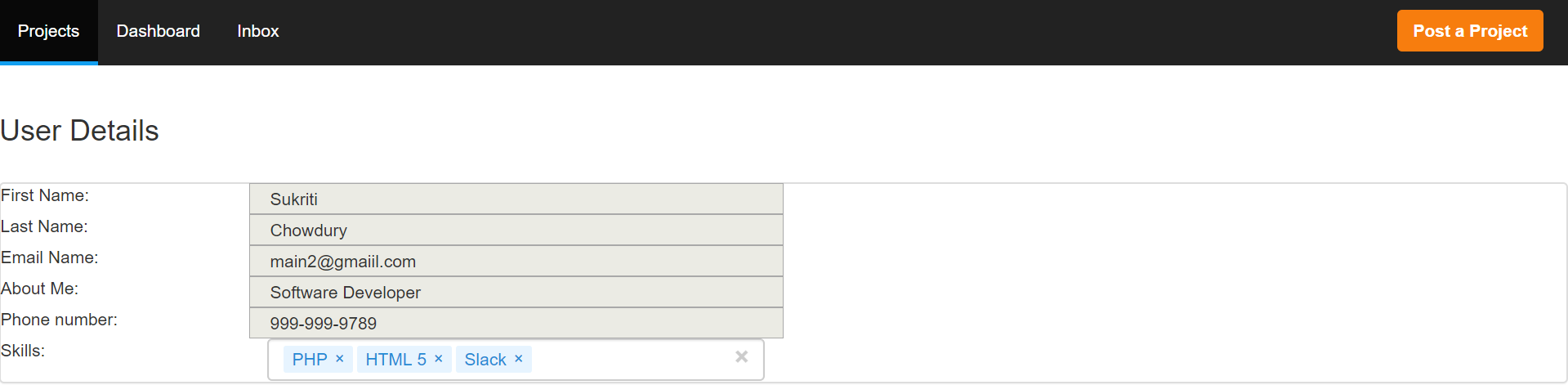


1. **Projects posted by me dashboard** **(both project name and username have hyperlink**
2. **Projects bidded by me dashboard (both project name and username have hyperlinks)**

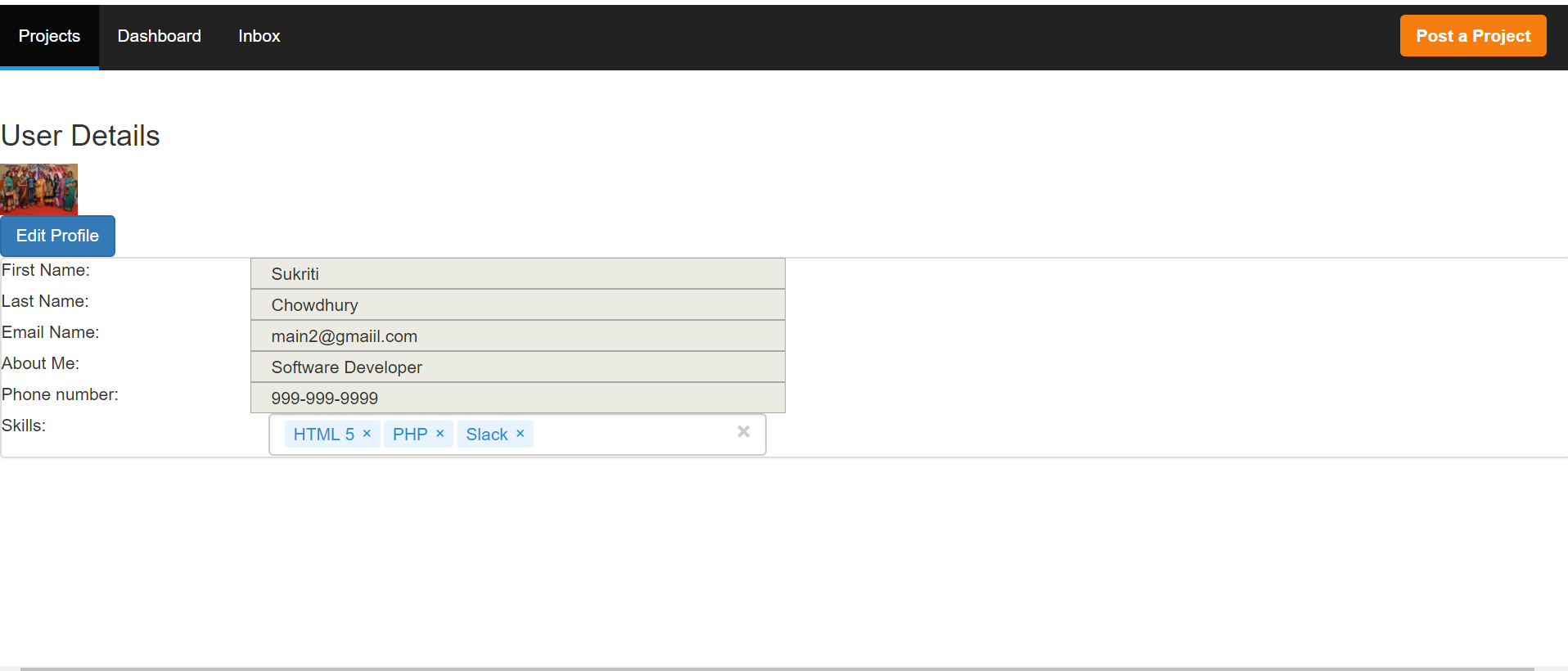


1. **Edit Profile**

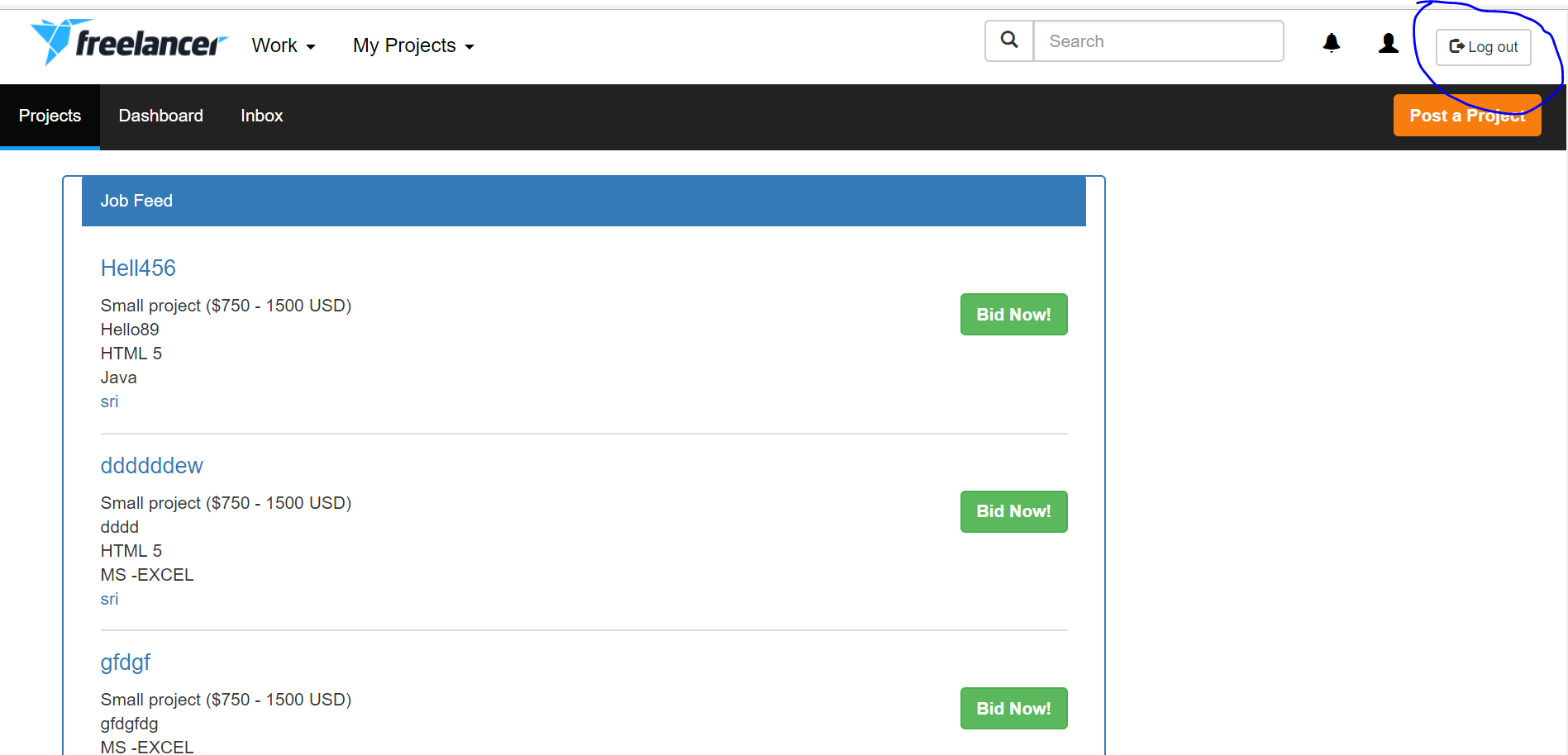


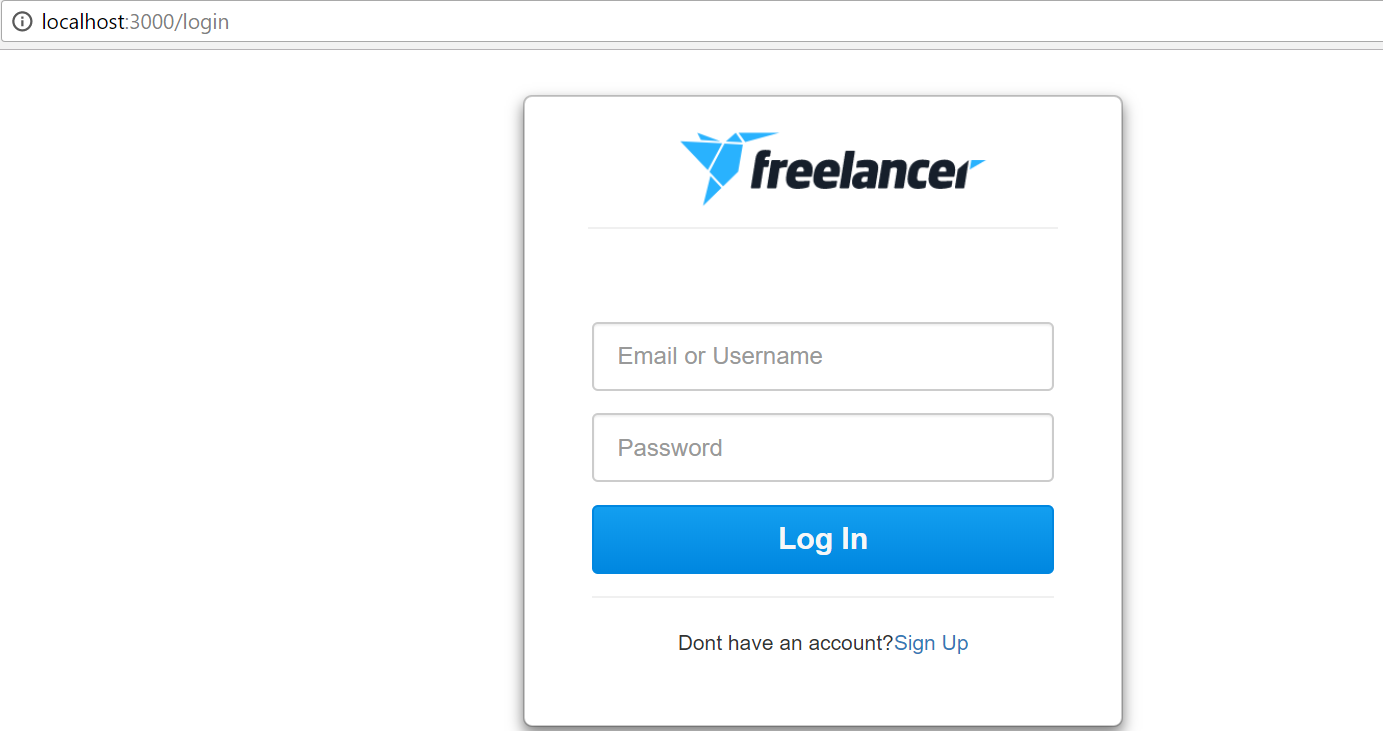


1. **User profile page**

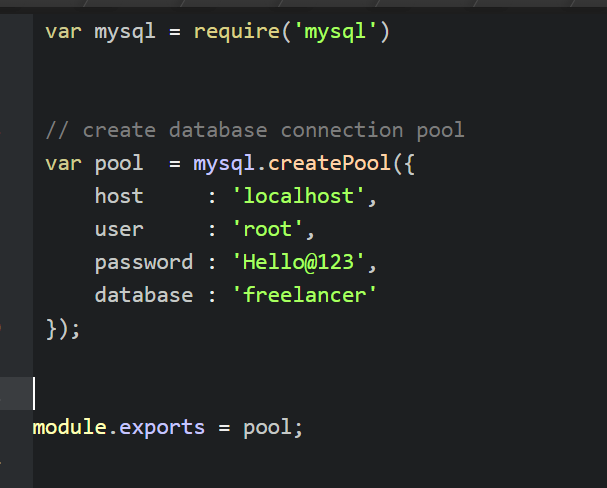


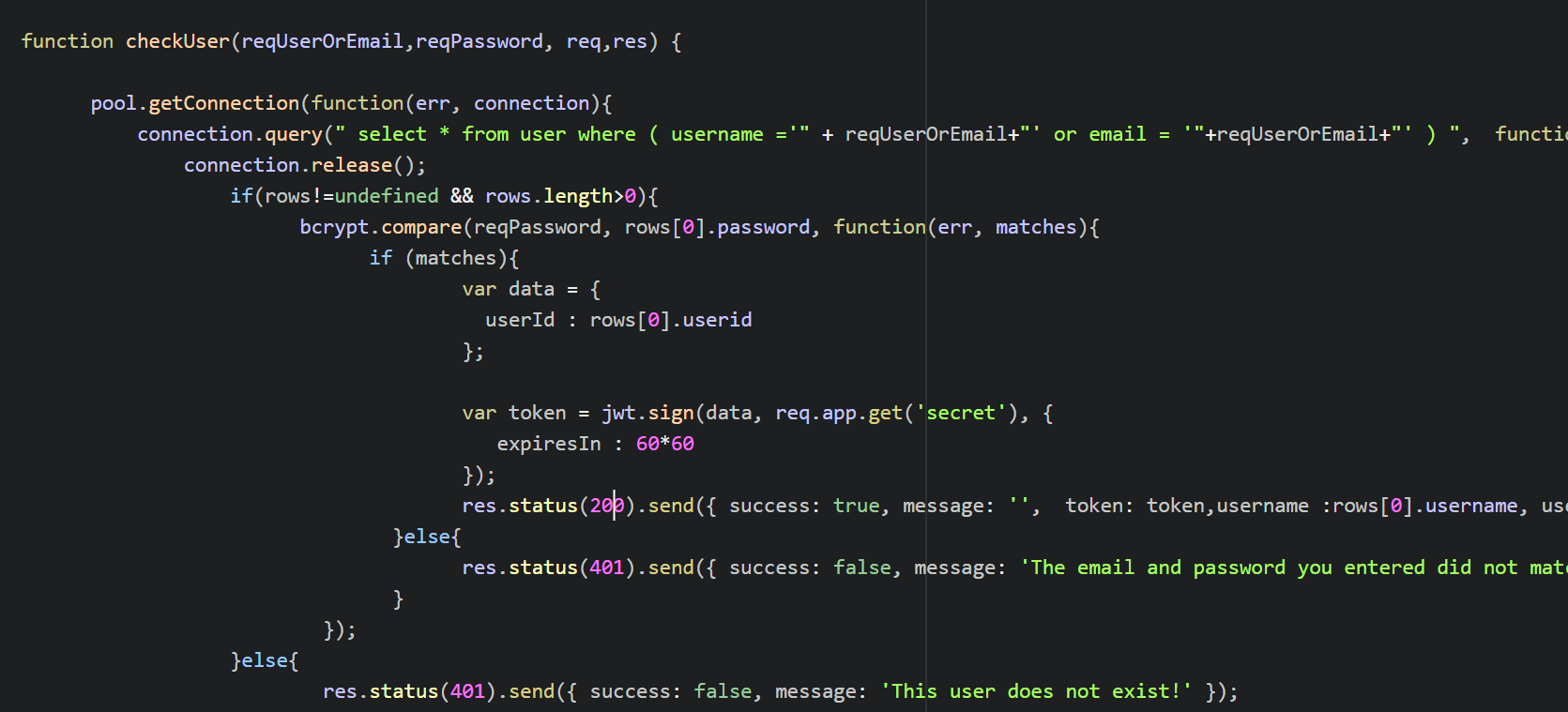
1. **Log out and return to login page**





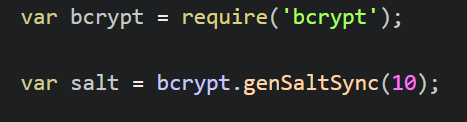
**Connection Pooling:**

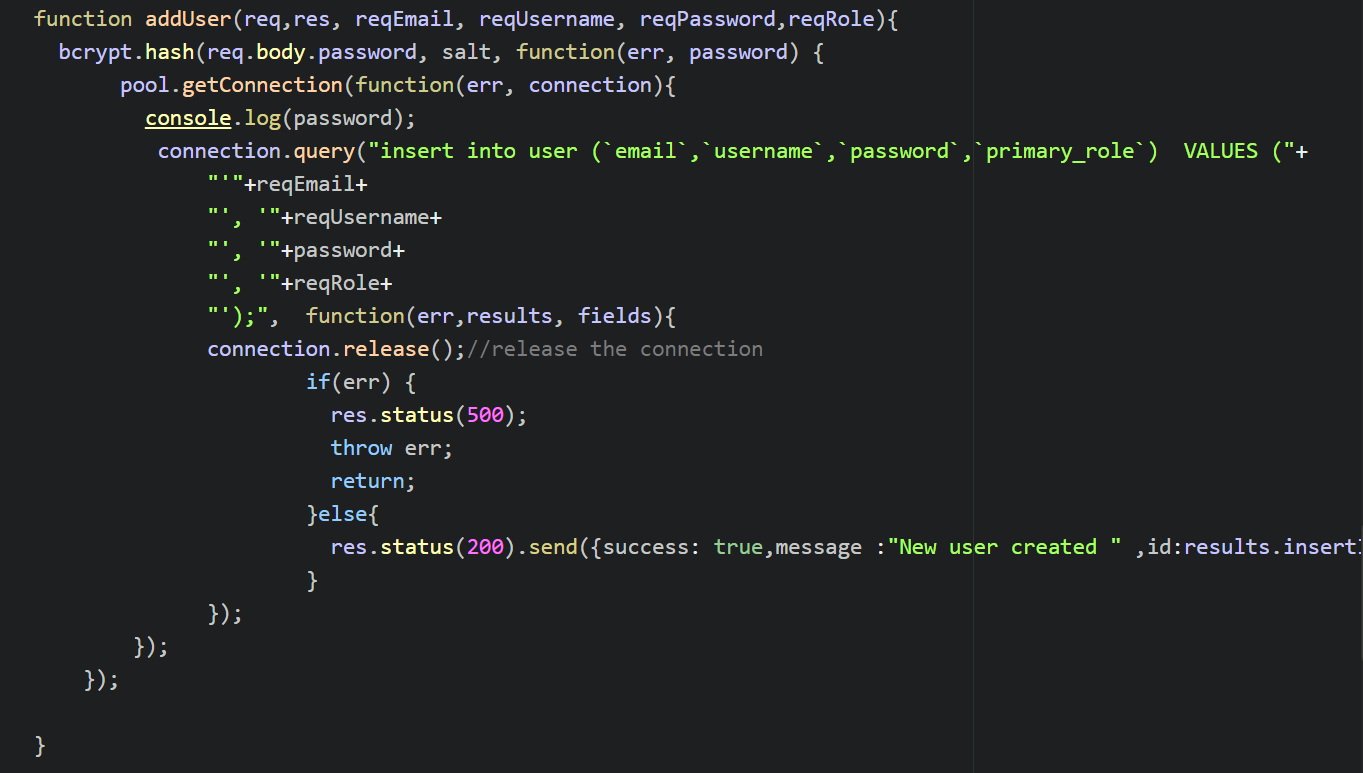


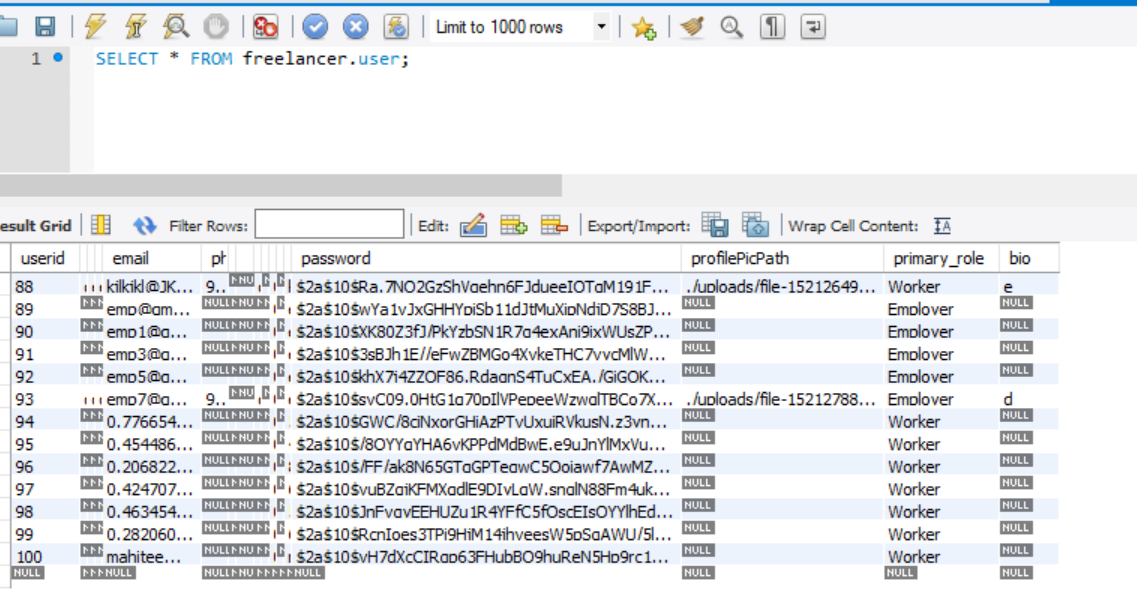


**Password Encryption :**

**I have used bcrypt for password encryption.**

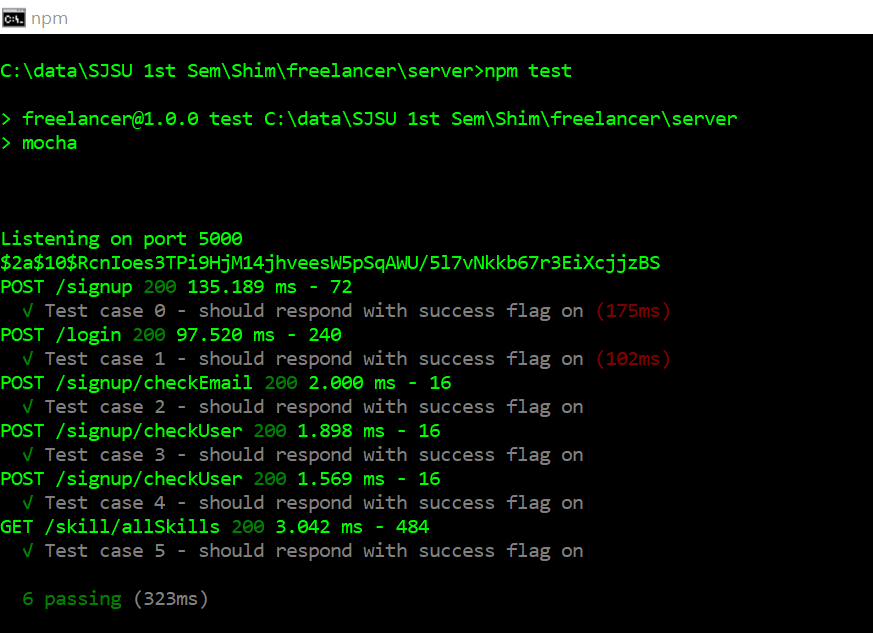






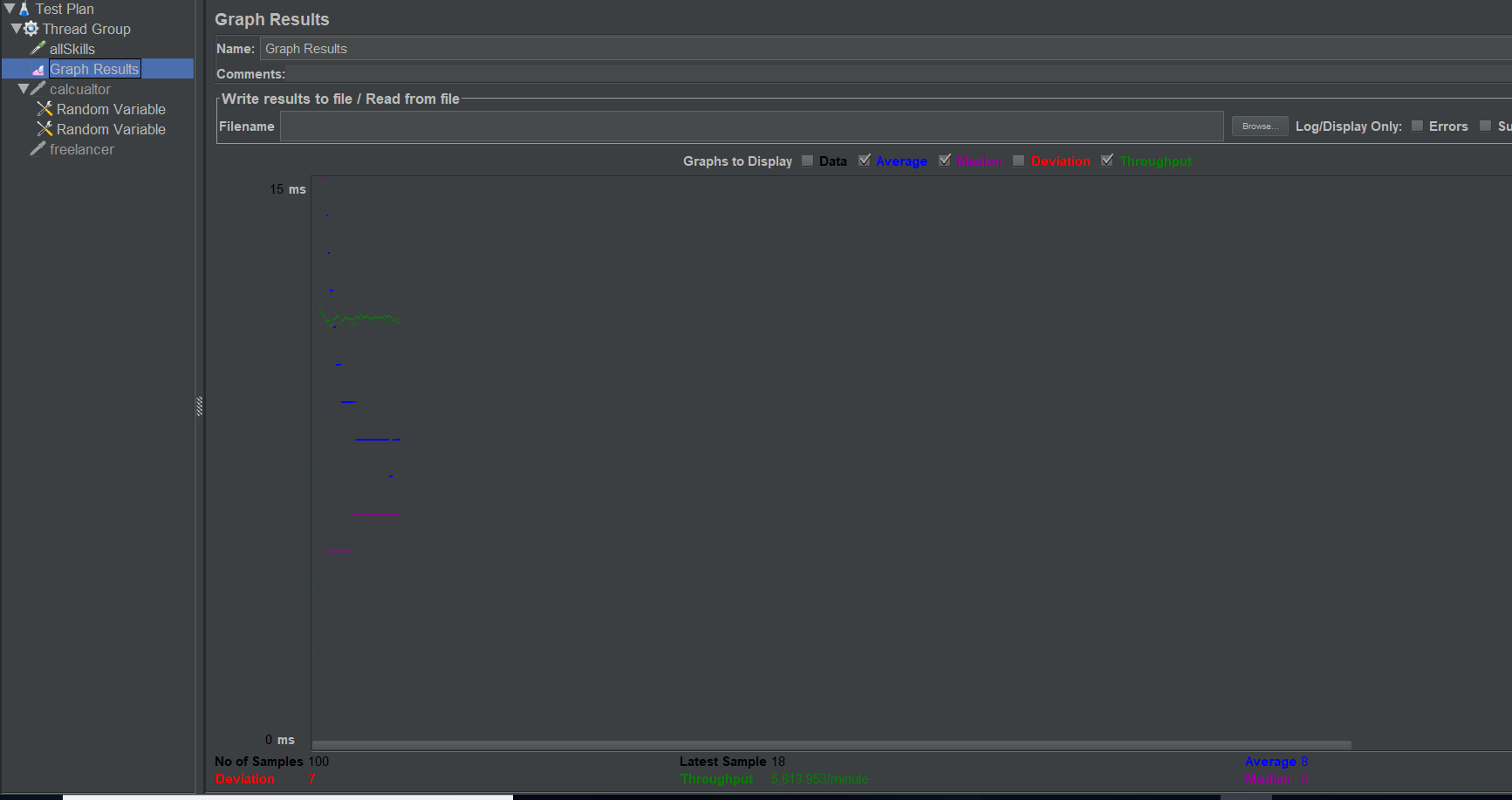
**Mocha Testing Report**

I have written 6 test cases for testing REST API

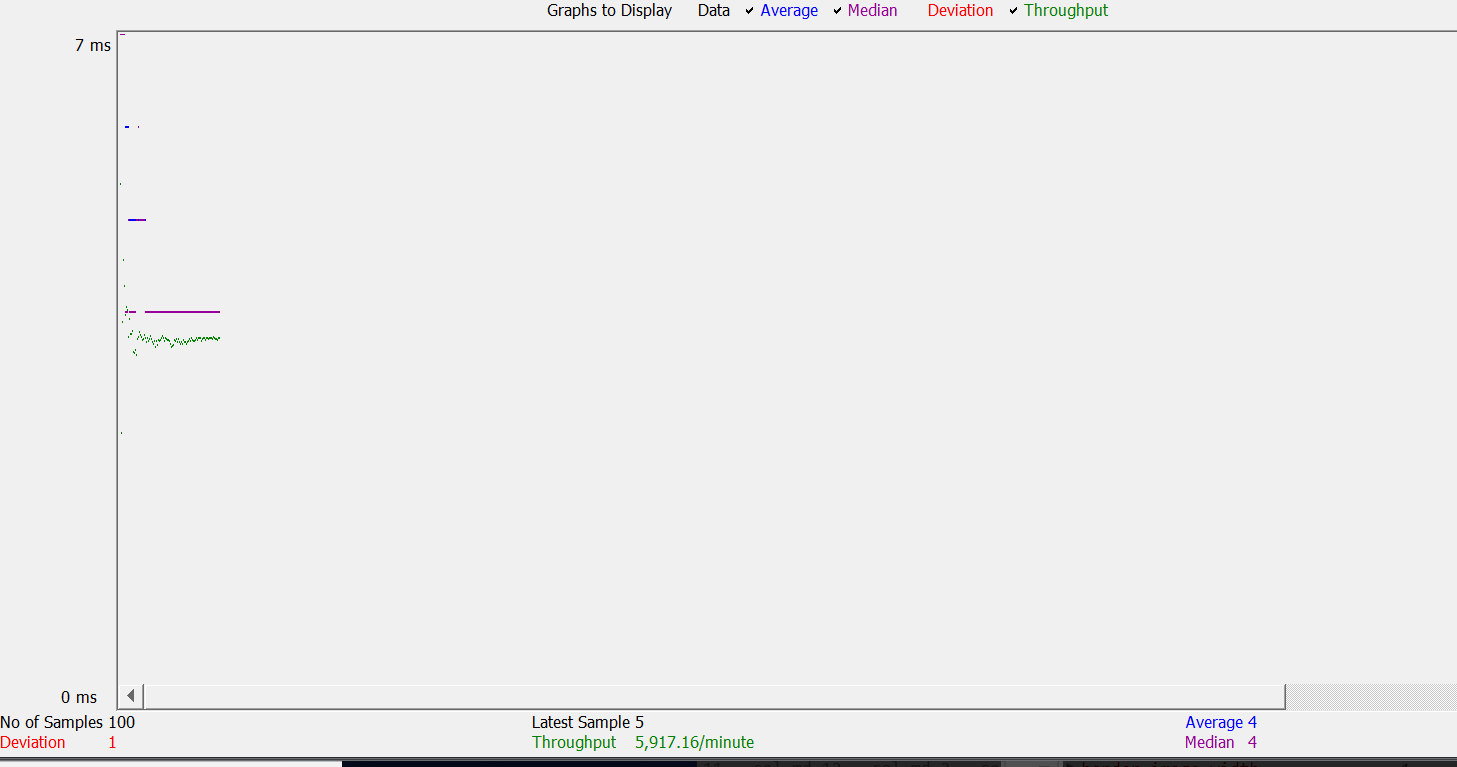


**JMeter Testing**

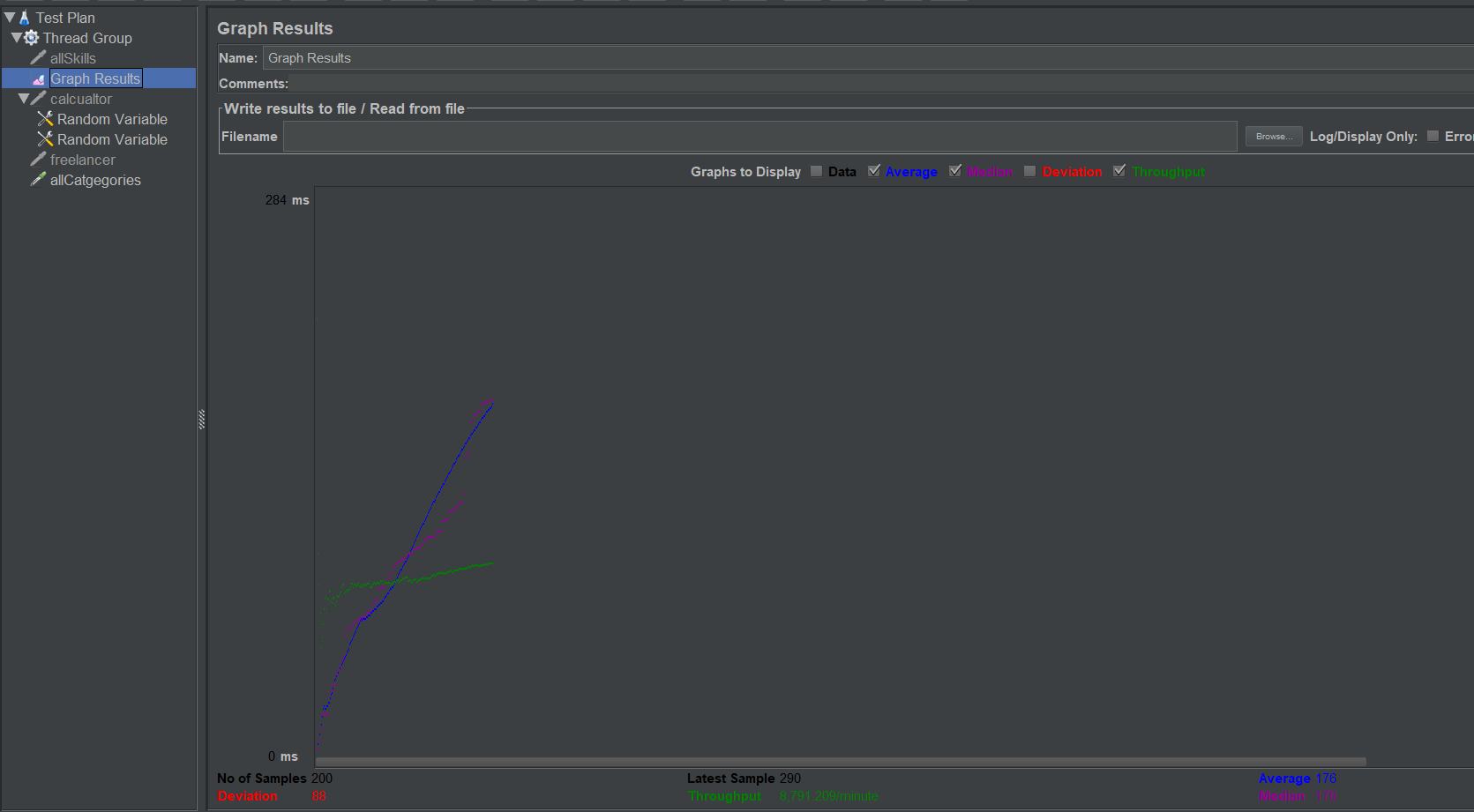
1. **100 concurrent users skill/allSkills - without connection pooling (Average 8)**



1. **100 concurrent users skill/allSkills – with connection pooling (Average 4)**



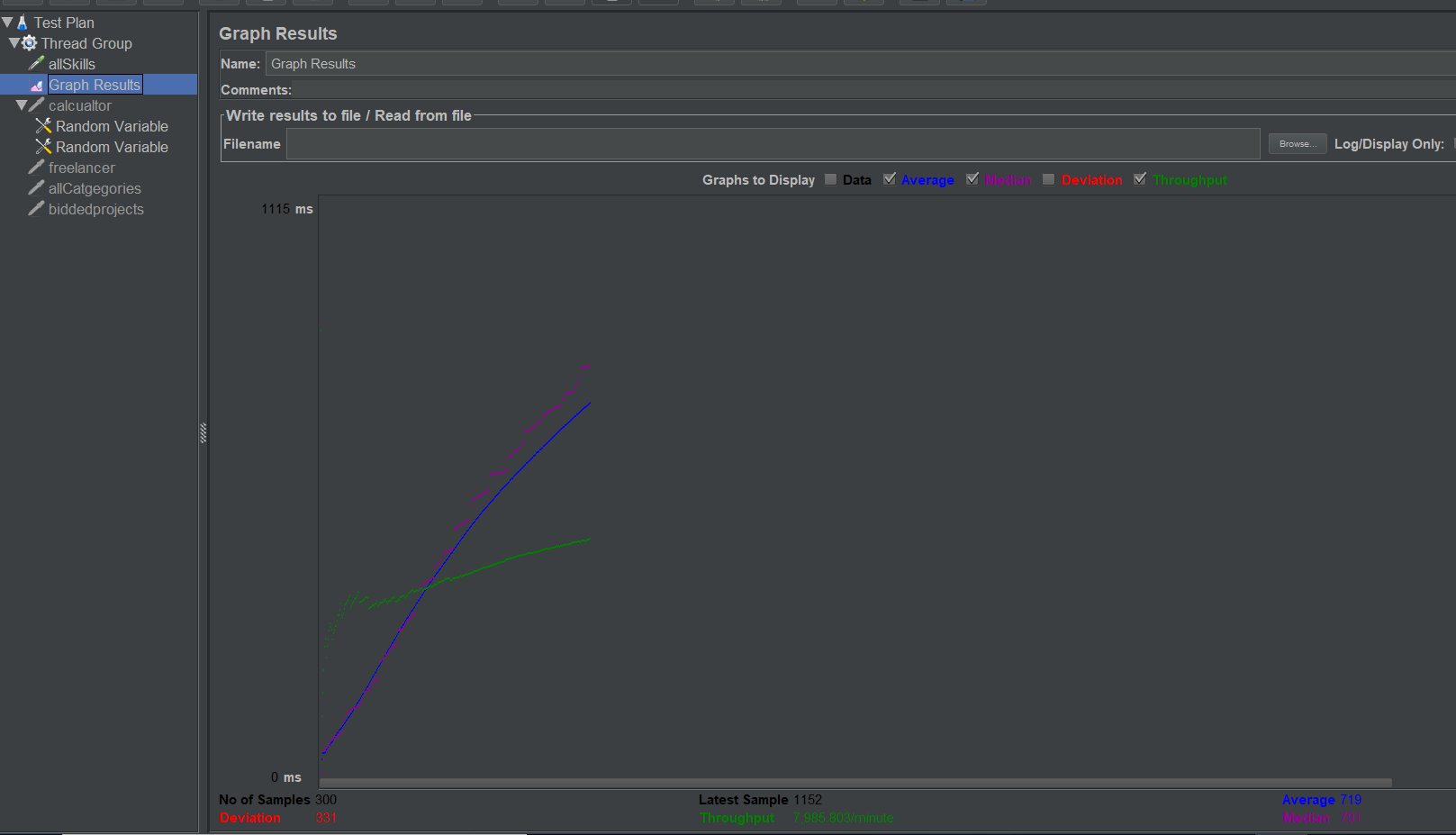
1. **200 concurrent users skill/allCategories – without connection pooling (Average 170)**



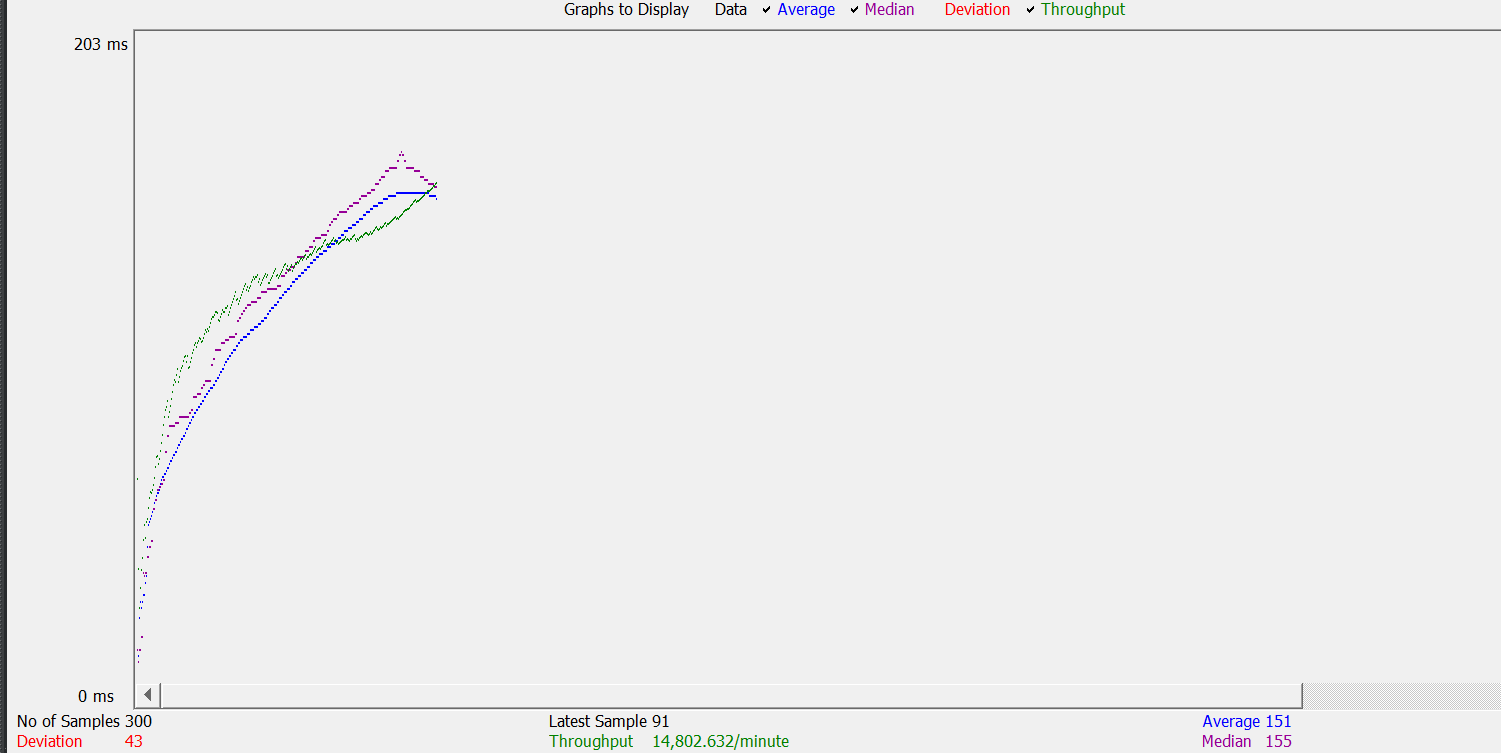
1. **200 concurrent users skill/allCategories – with connection pooling (Average 7)**



1. **300 concurrent users /biddedprojects/1– without connection pooling (Average 719)**

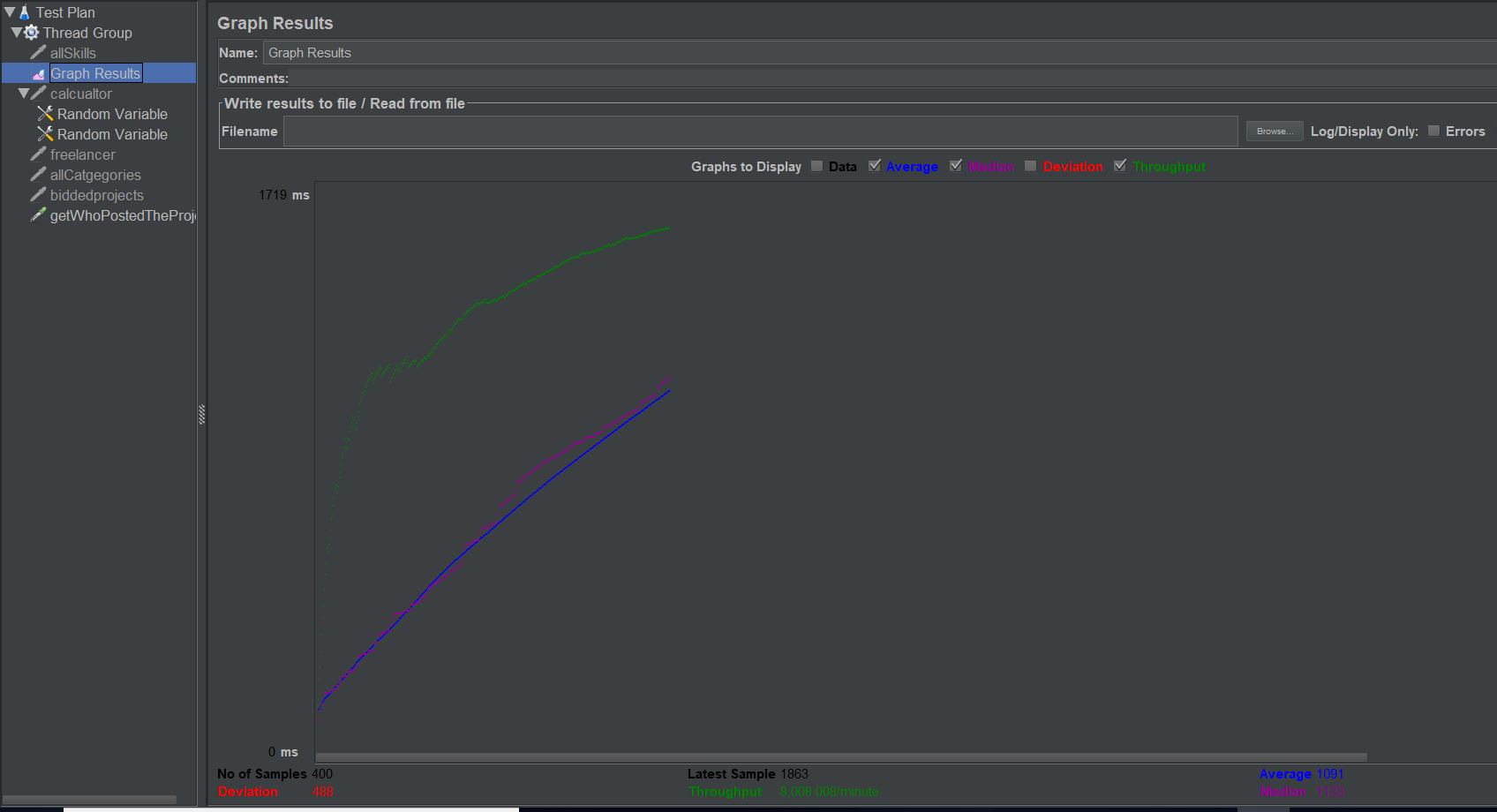


1. **300 concurrent users /biddedprojects/1 – with connection pooling (Average 151 )**



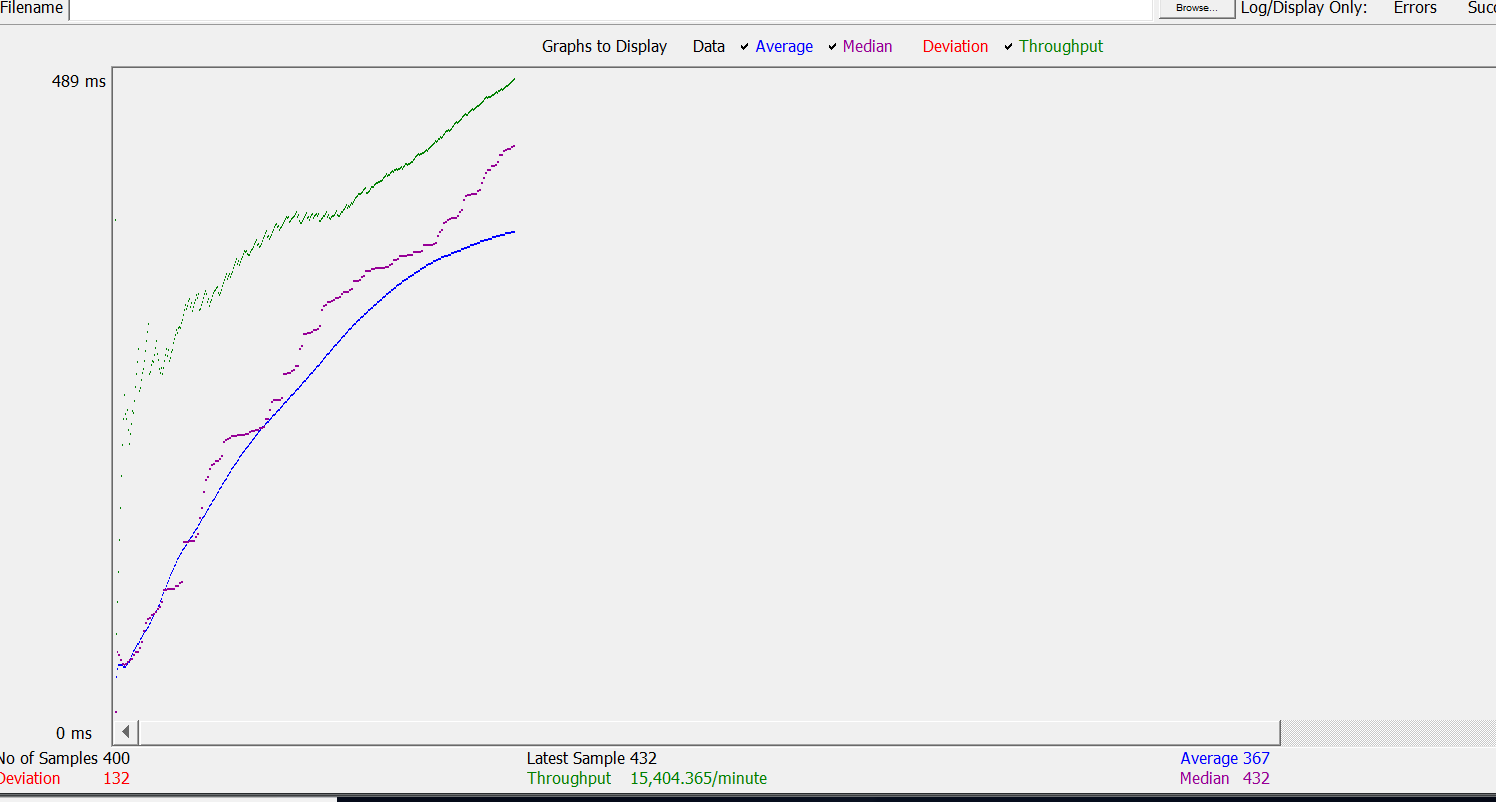
1. **400 concurrent users /project/getWhoPostedTheProject/1** – **without connection pooling**

**( Average 1091)**

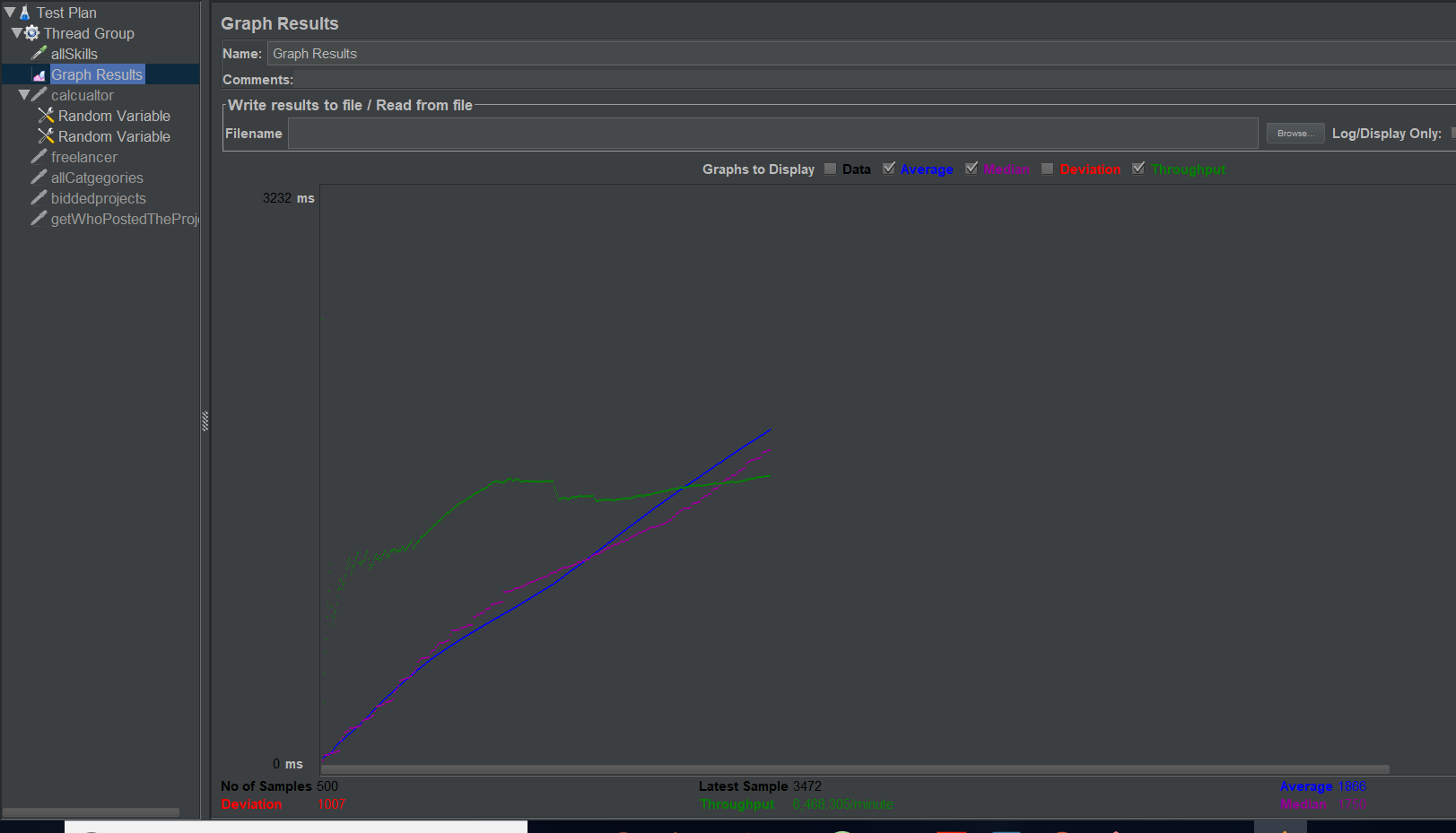


1. **400 concurrent users /project/getWhoPostedTheProject/1** –**with connection pooling**

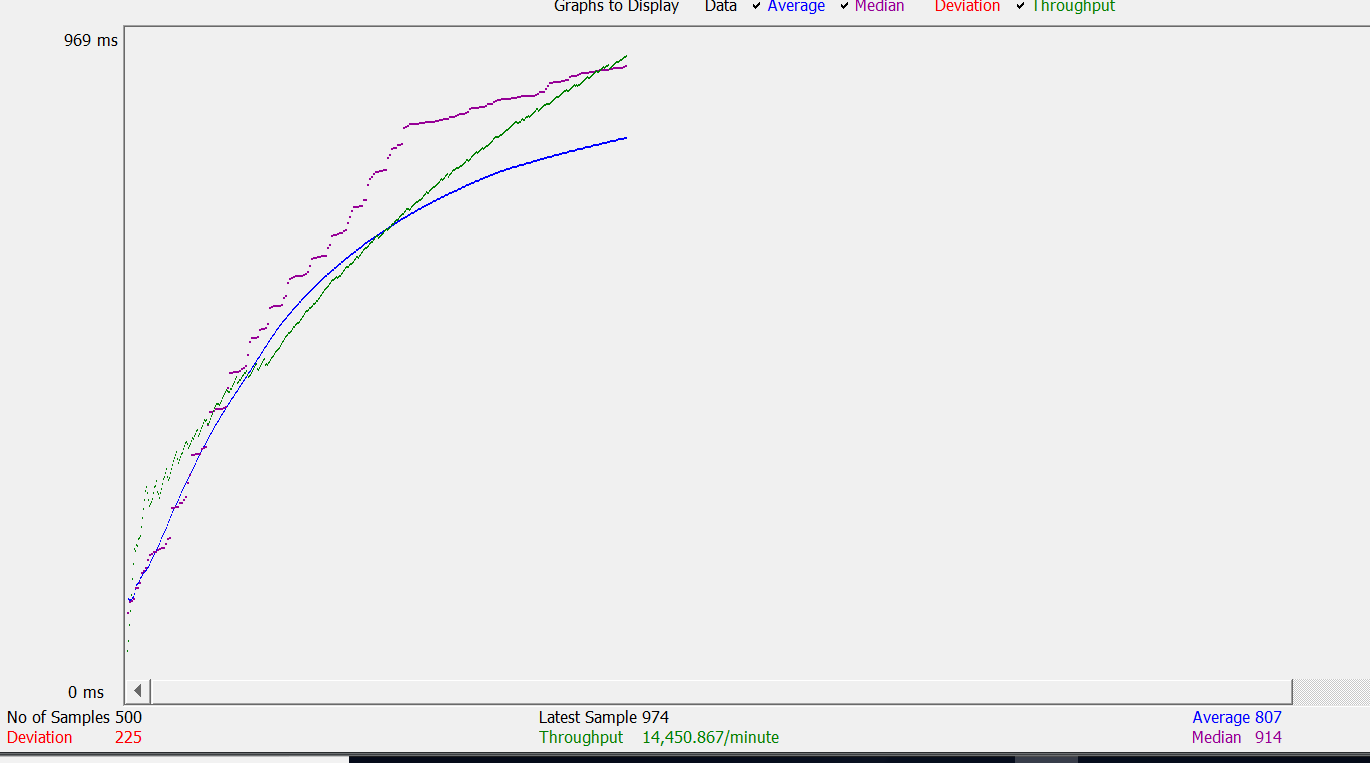
**( Average 367)**



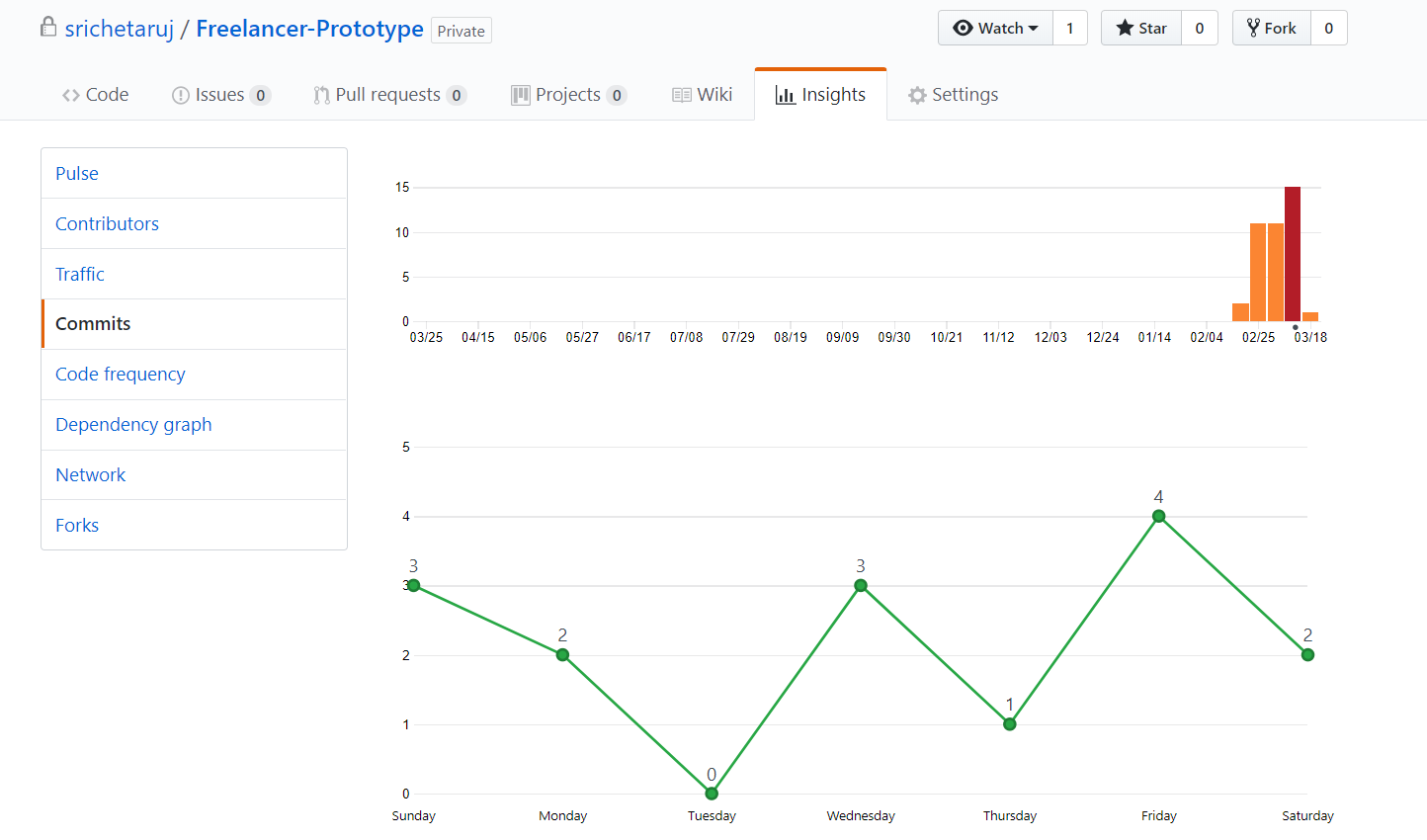
1. **500 concurrent users /project/getRecommendedProjects/1 request without connection pooling ( Average 1866 )**



1. **500 concurrent users /project/getRecommendedProjects/1 request with connection pooling ( Average 807 )**



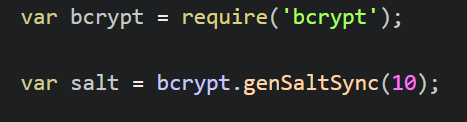
**Github commit history:**



**Questions and Answers**

1. **Explain the encryption algorithm used in your application. Mention different encryption algorithms available and the reason for your selection of the algorithm used.**

**Ans.** I have used bcrypt for encrypting password when a users sign’s up in the application. It hashes the password. Bcrypt is available in npm package manager. It is easy to hash and compare passwords. It uses the Eksblowfish algorithm to hash passwords. It is adaptive hash algorithm. Any state depends on both salt and key which is the password which I have used to generate the has. Hence, no state can be precomputed without the knowledge of both. Without the knowledge of the salt and password, one cant get the credentials. The salt is computed by passing the rounds which is the cost of processing data.



There are different hashing algorithms like SHA1, MD5, SHA512, RSA, AES, SHA256, SHA-3, etc. These are all general-purpose hashing techniques. They are perfect for calculating hash for plethora of data in minimum. Hence, they are good for ensuring the integrity of data and but not good for storing passwords. Bcrypt is an adaptive hash function that introduces a work which allows you to determine how expensive the hash function will be. This work factor value determines how slow the hash function will be, which makes it extremely resistant to brute force attacks. Any modern server have enough computing power that can crack hash of above mentioned algorithms easily. That’s why I have used bcrypt for hashing.

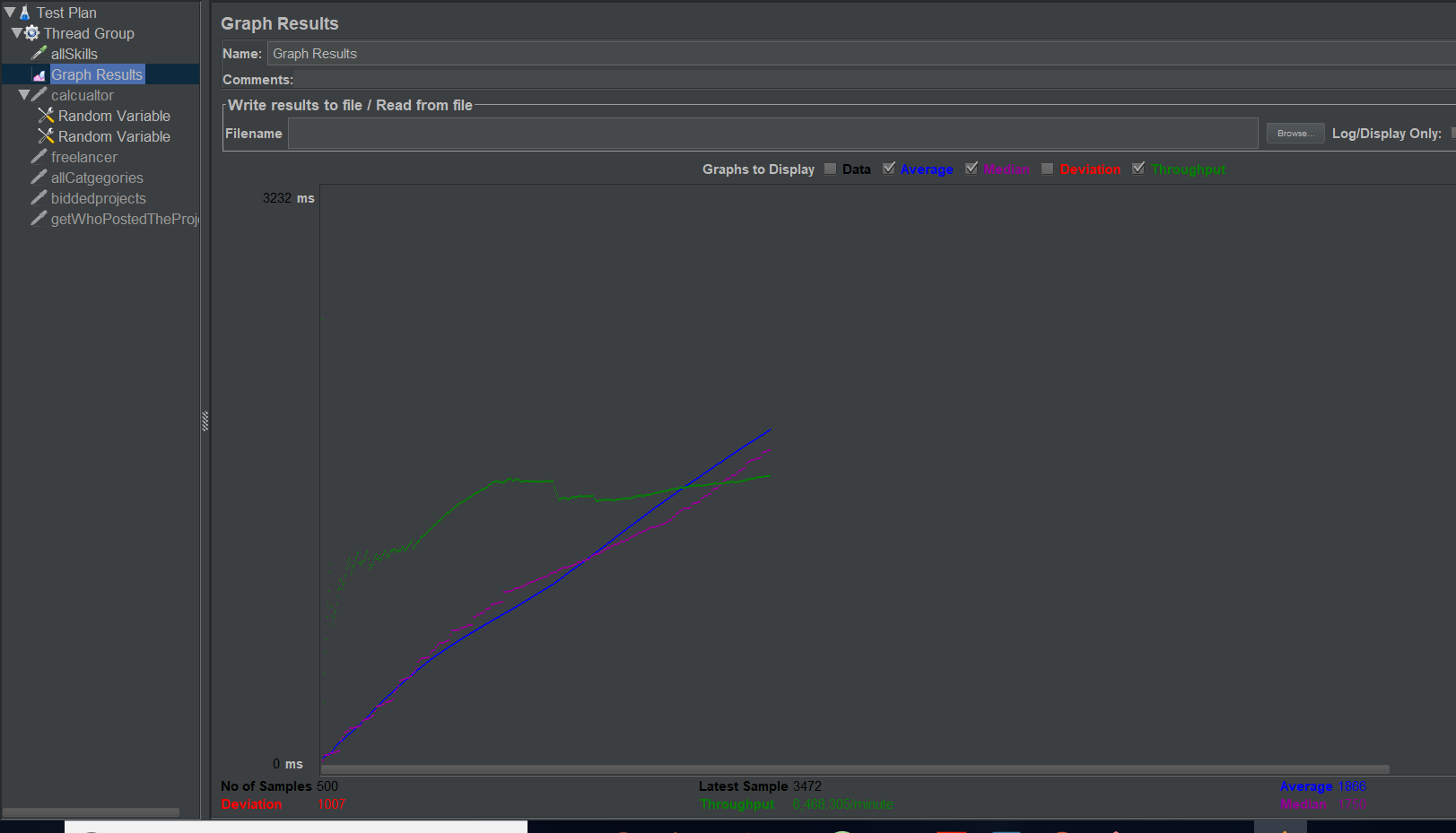
1. **Compare the results of graphs with and without connection pooling of database. Explain the result in detail and describe the connection pooling algorithm used in your code.**

**Ans :**

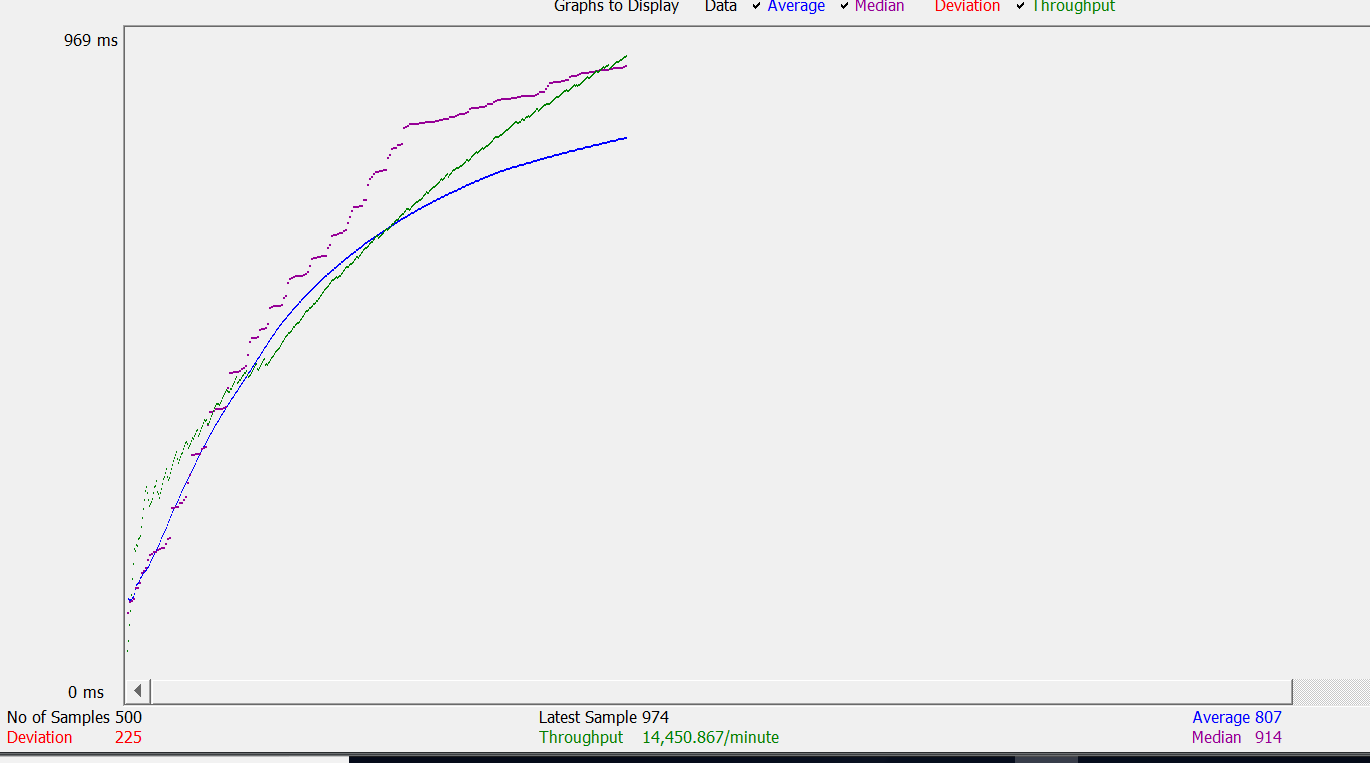
|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Average time | |
| Concurrent users | API | without connection pooling | with connection pooling |
|  |  |  |  |
| 100 | /skill/allSkills | 8 | 4 |
| 200 | skill/allCategories176 | 176 | 7 |
| 300 | /biddedprojects/1 | 719 | 151 |
| 400 | /project/getWhoPostedTheProject/1 | 1091 | 367 |
| 500 | /project/getRecomendedProjects/1 | 1866 | 807 |

**Summary of average tome of 5 api’s with and without connection pooling**

500 concurrent users /project/getRecommendedProjects/1 request without connection pooling ( **Average 1866 )**

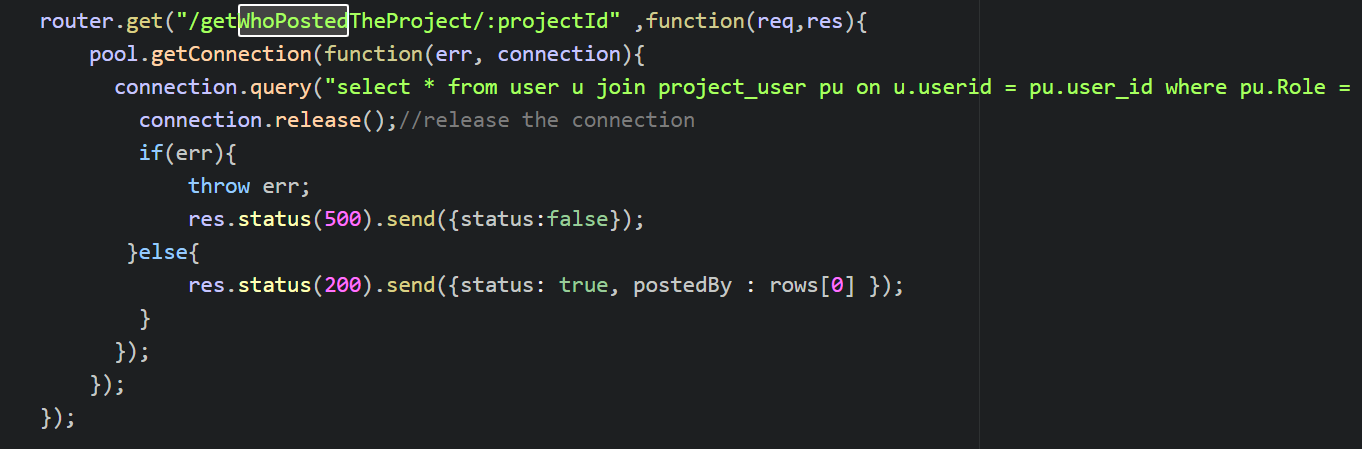


500 concurrent users /project/getRecommendedProjects/1 request with connection pooling ( **Average 807 )**



Here we see it clearly, without connection pooling running 500 concurrent users making request in get /project/getRecommendedProjects/1 api takes a longer than time taken using connection pooling.

MySql connection pooling allows us to create concurrent connections of a maximum number. When a query is fired, we go to the pool and get a connection from there. NodeJS mysql provides api for connection pooling. If there are any connection available in the pool then new connection is fetched and after query is finished the connection is released by writing connection.release(); after every api and returns to pool for further use. Example is given below.



1. **What is SQL caching? What all types of SQL caching is available and which suits your code the most. You don’t need to implement the caching, write pseudo code or explain in detail.**

Data is accessed using select query. With SQL caching data is stored in-memory. Querying a database is much slower than accessing data in memory. We need to identify which data is queried most by the users and save it in-memory for faster access time and high performance. In my freelancer application, the application revolves around skills and categories which are static. So, its is better to cache them in memory rather than querying it every time.

There are 4 types of caching mechanisms:

1. Adhoc query caching
2. Auto parameterization
3. Prepared queries, using either sp\_executesql or the prepare and execute method invoked through your API
4. Stored procedures or other compiled objects (triggers, TVFs, etc.)

 To determine which plan is most suitable, we need to look at the value the cacheobjtype and objtypecolumns in the sys.dm\_exec\_cached\_plans view. The cacheobjtype column can have one of five possible values:

Extended Procs, Inactive Cursors, Algebrizer Tree, Execution Plans, Compiled Plans.

Pseudo code for caching:

//check if required skill present in memory cache or not

If(cacheMemory.contains(quert.skill)){

//if yes then take data form in-memory cache

return cacheMemory[query];

} else {

//if no then take data form database

return FreelancerDatabase.query(query);

}

1. **Is your session strategy horizontally scalable? If YES, explain your session handling strategy. If NO, then explain how can you achieve it.**

Yes. It is horizontally scalable. In horizontal scenario, when you start to replicate servers, you must deal with the separate central session storage system. Use of JWT in this scenario is seamless, there is no need to store session since token-based authentication is stateless. Application can scale easily because tokens are used to access resources from different servers without worrying if the user was logged in on a server. JWT prevents tempering on client side as well as they also can be encrypted. This reason makes application very secure. It is very lightweight and smaller in size, can be sent with URL, POST parameter or inside HTTP header. It is also good way to transmit data successful between any parties

Advantages of JWT:

1. Easier to scale
2. Easier to use
3. More flexible
4. Built in expiration functionality
5. Works for users that blocks cookie