CMPE 273 Lab 2

Hongyuan Li (011838151)

Freelancer

http://ec2-18-218-144-136.us-east-2.compute.amazonaws.com:3000/

Introduction

The freelancer application is built to allow users post and bid for projects. Users can sign up, login and logout the website. Users can update their profiles with skills and images. Users can post projects with budget, skill set information. Users can bid on projects with price and number of days. Project owners can hire bidders.

Kafka Backend server is built with Node.js, Express.js, Kafka-node, and mongoose.

REST Server is built to Node.js, Express.js and kafka-node. REST Server connects to Kafka Backend through kafka.

Frontent client is built with React, redux and Saga. In addition:

'react-router-dom' is used for routing

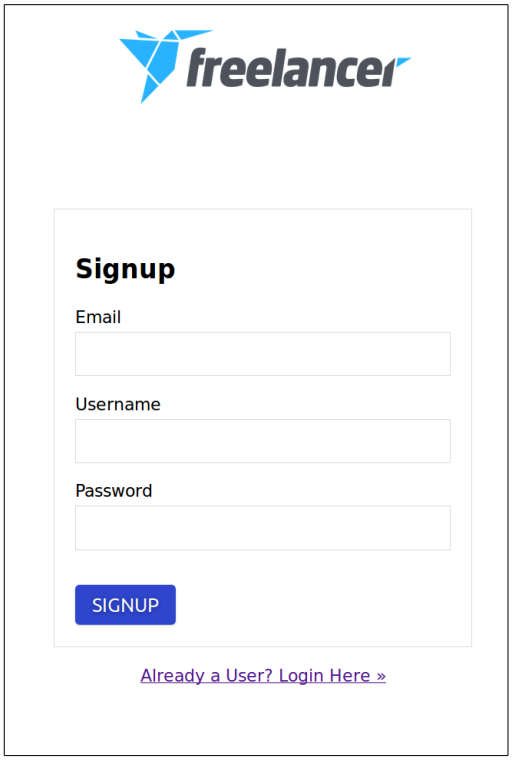
'admin-on-rest' and 'material-ui' are used to build the layout and widgets.

Results

User functionalities:

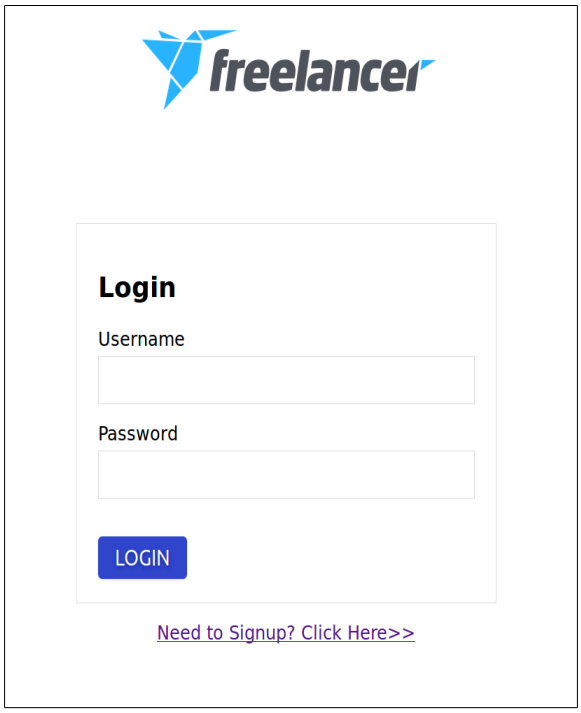
1. Sign up

Users need to provide, email, username, and password to signup. Once signup is successful, page will be redirected to login. Bcrypt is used to encrypt passwords



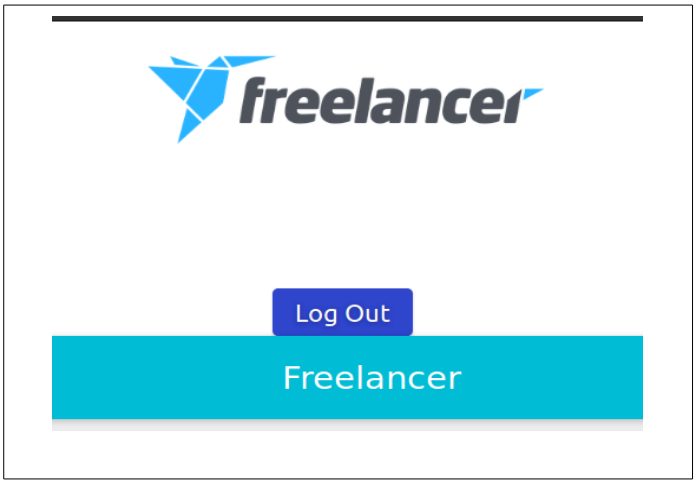
2. Sign in

Users need to use username and password to sign in. When login is successful, page will be redirected to home



3. Sign out

On top of the home page is the sign out button. When click, user is signed out and the page is redirected to login.



4. Menu:
On the left side of the home page, there's a responsive menu. Here's the direction.

Projects: list all the projects. Users can edit, create, and bid projects on this page, depends on ownership and availability.

Bids: list all the bids. Users can delete their own bids, if it's not chosen. Employers can click on "hire" to hire bidders.

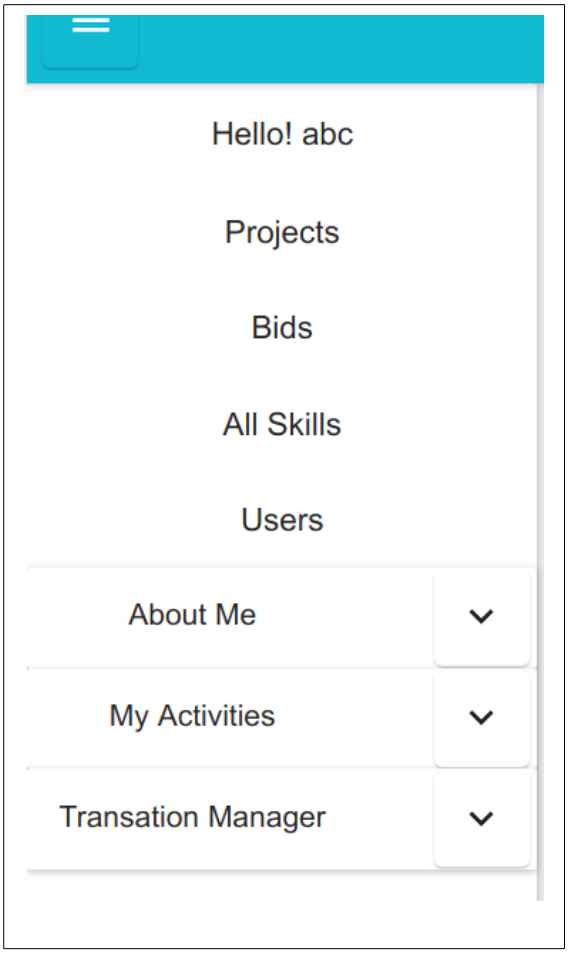
All skills: list all skills available to be added to users and projects

Users: list all users.

About Me: contains options about the user

My Activities: contains options about bids and projects related to the user

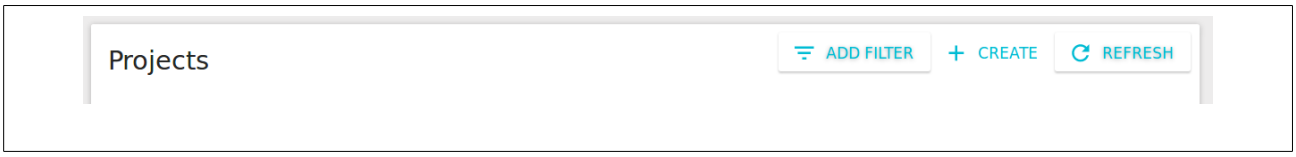
Transaction Manager: transaction informations



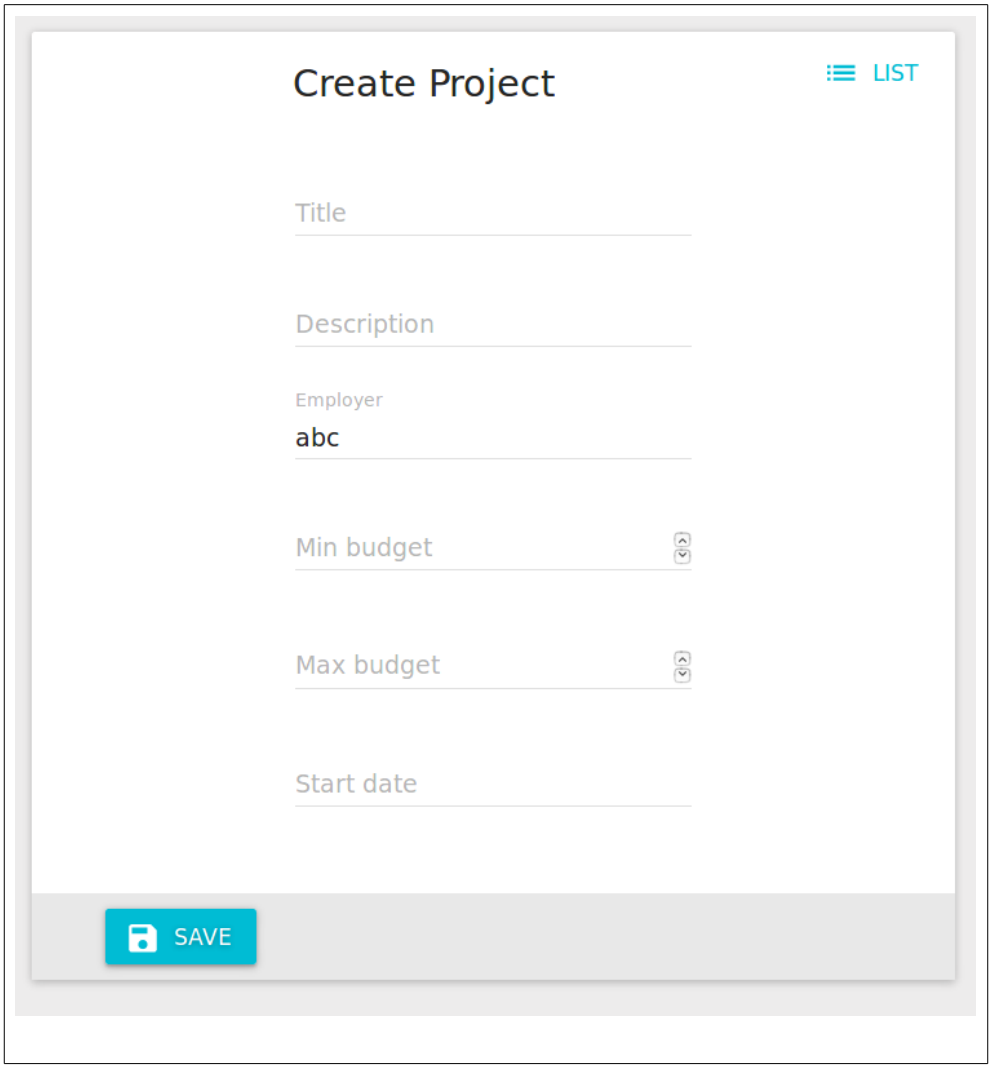
General Information:

1. create a project

On the project page, users can click at the "create" button to create a project.

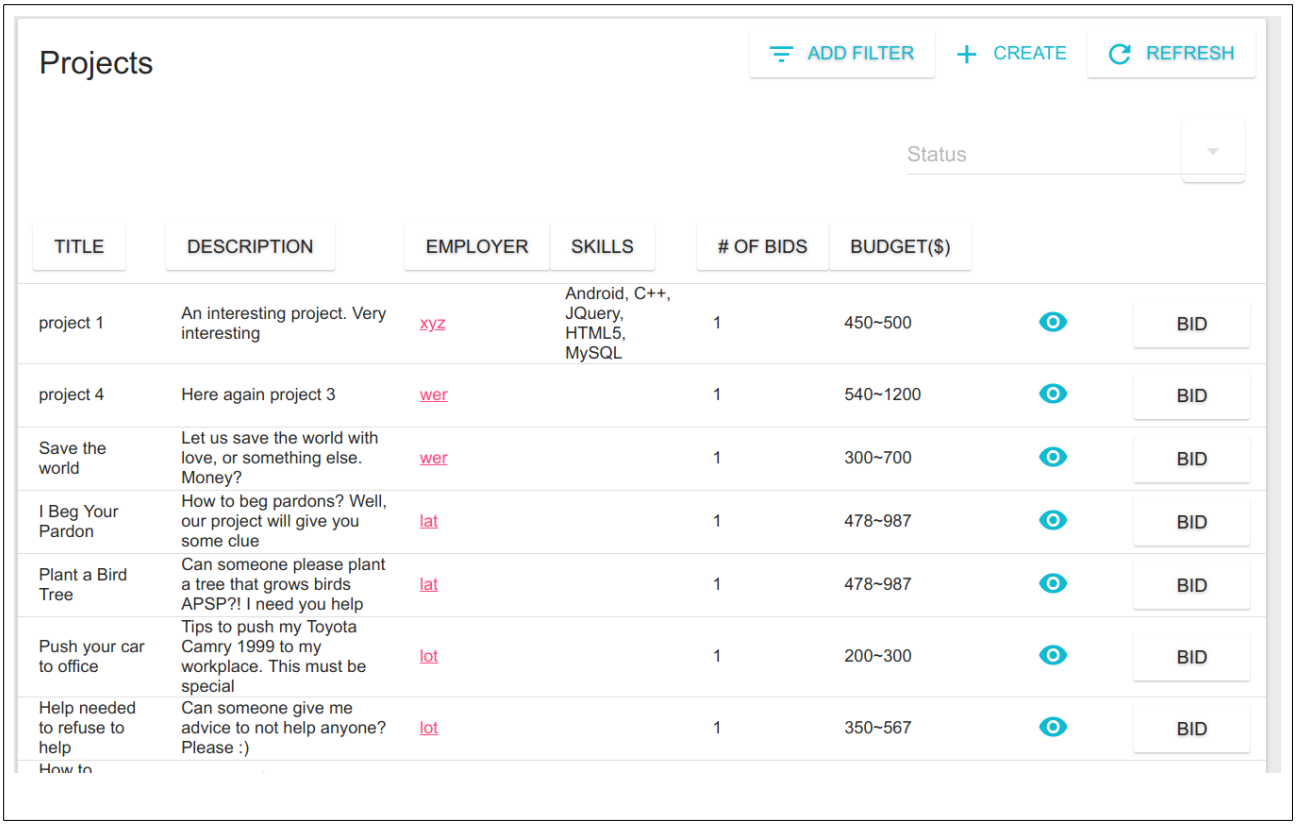


Users can fill out the create project form, to create a new project, with basic information. Once click "save", users are redirected to the projects page again.

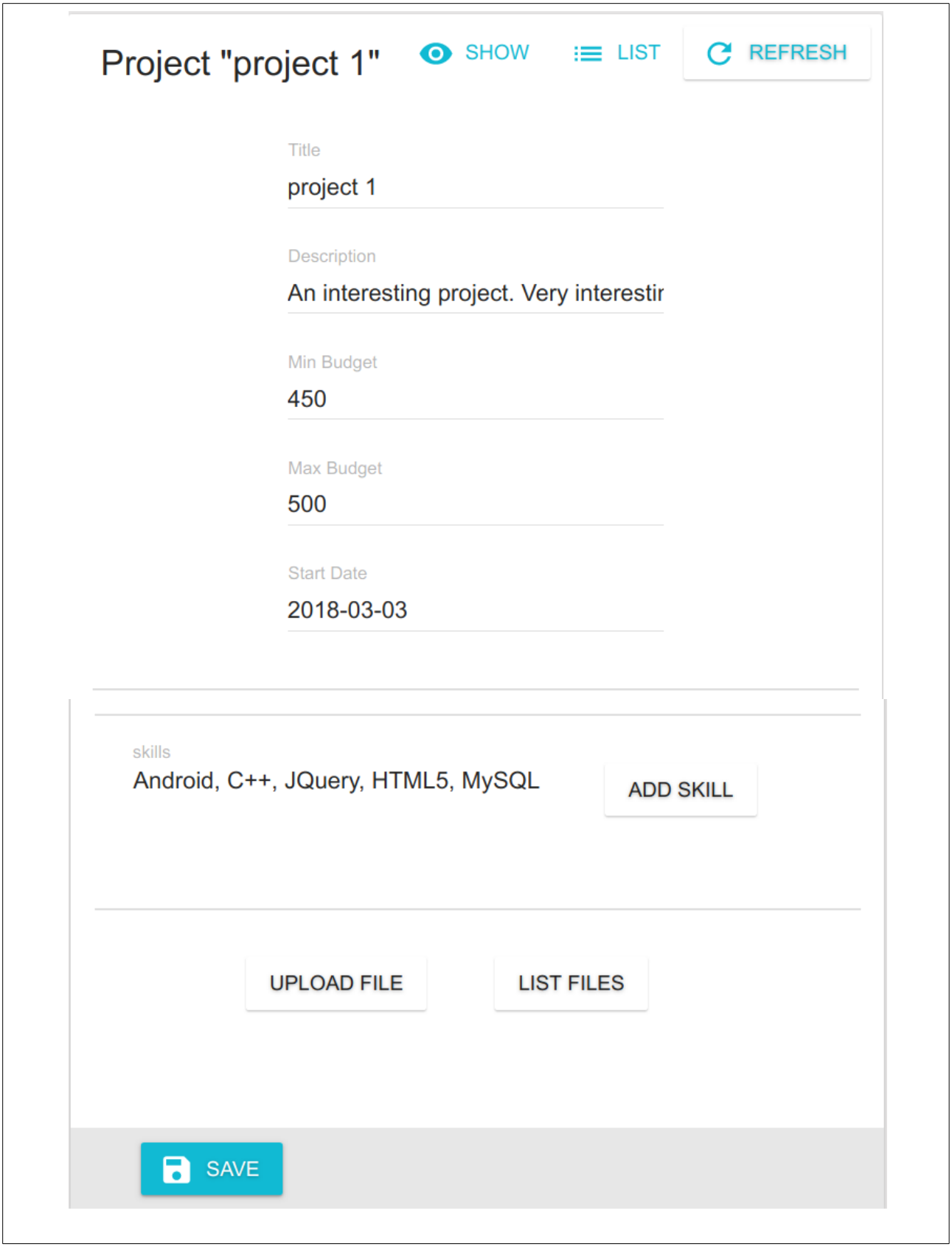


2. Edit Project

Once a project is created, we can click at the "Edit" button to add skills and upload files. Only project owners can edit their own projects.

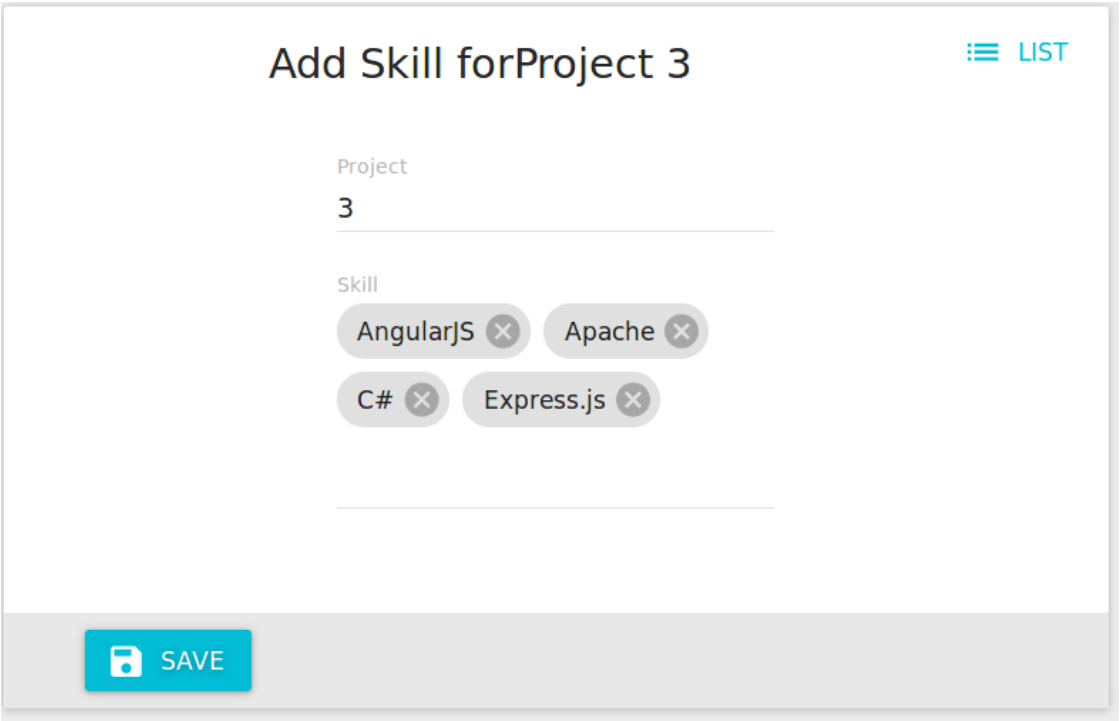


On the edit page, we can click at "Add Skills" and "Upload", for adding skills and uploading files respectively.

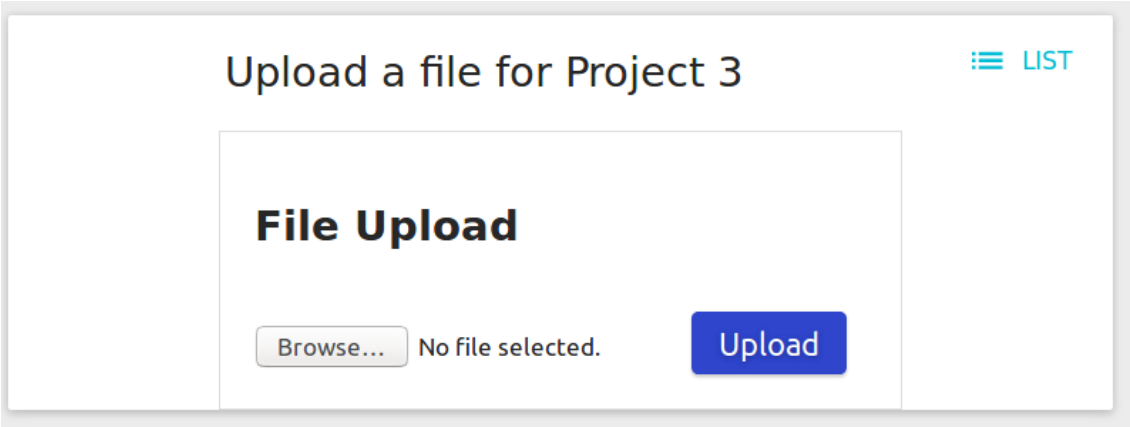

Project "project 1"
SHOW
LIST
REFRESH
Title
project 1
Description
An interesting project. Very interestir
Min Budget
450
Max Budget
500
Start Date
2018-03-03
skills
Android, C++, JQuery, HTML5, MySQL
ADD SKILL
UPLOAD FILE
LIST FILES
SAVE

3. Add Skills

The add skill page allows the project owners to add skills required for this project.

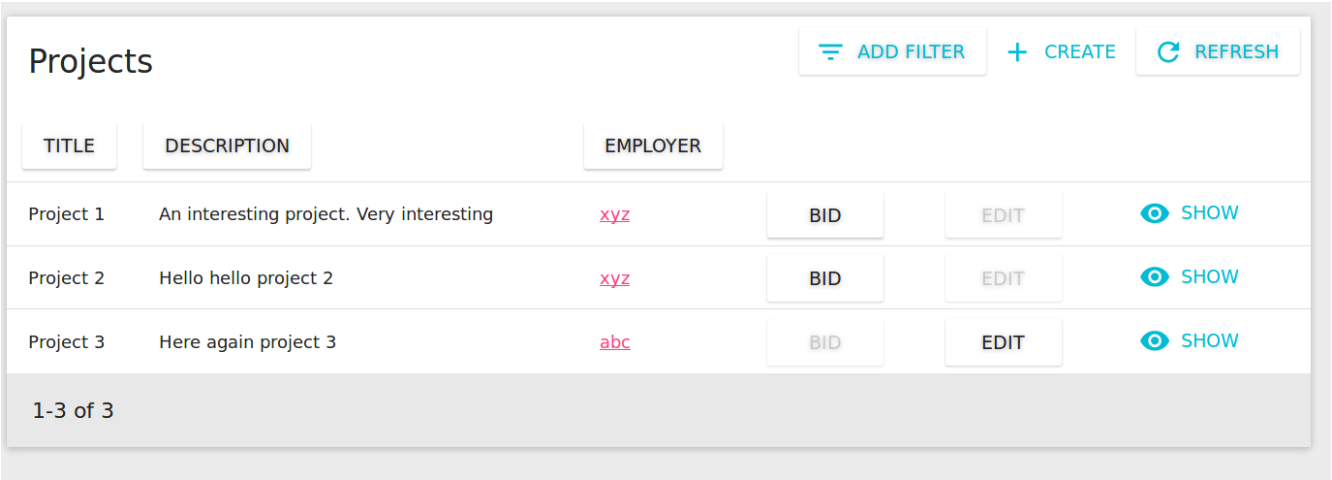


4. Upload files
We can also upload files for the project. Once the file is uploaded, page is back to the edit project page.

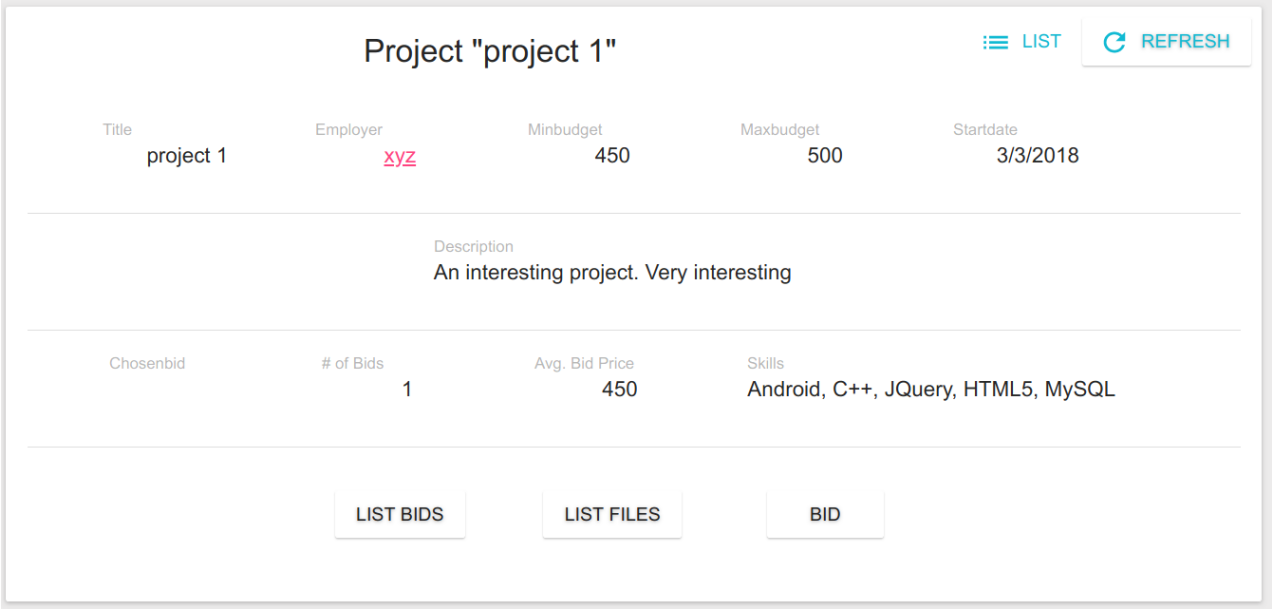


5. Projects

The projects page shows all the available projects, users can click at "BID" to bid on these projects. Employers are listed links to the user information page. The "Show" button, will open a detail view page for the project.



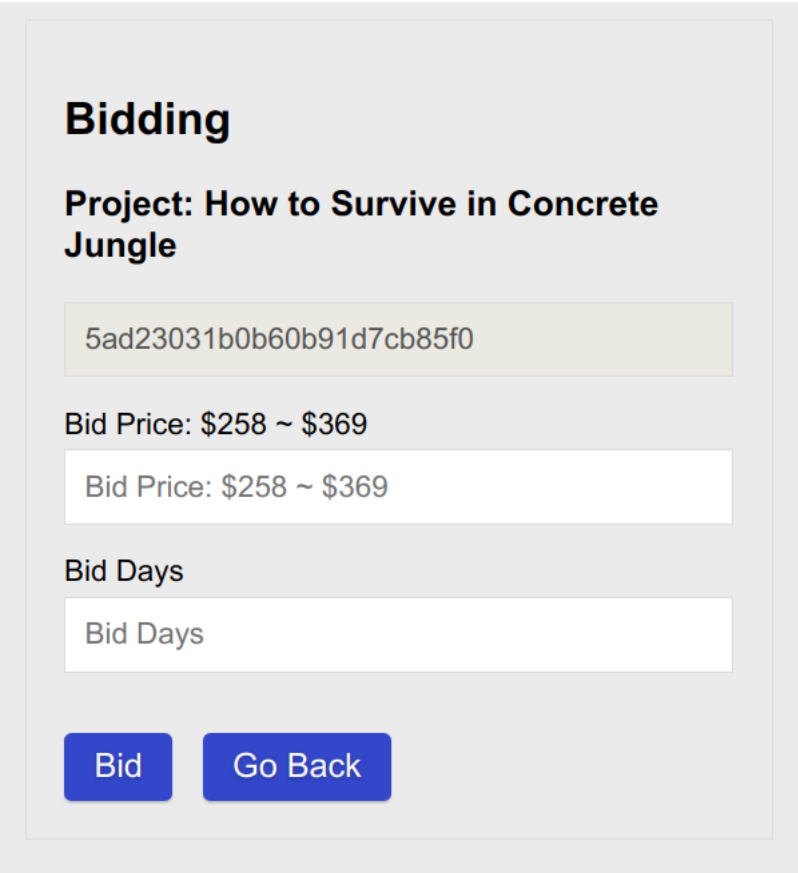
6. Show Project



On the show project page, we can see the regular projects details, as well as, number of bids, average bid prices, and whether a bid is chosen. It also shows the skills required for the project. If you click at the "LIST FILES" button, the page will be redirected to the file list page for this particular project.

7. Bid

By clicking on the "BID" button on a project, the bidding page is open. Project ID, User ID and employer ID are prefilled on disabled text filled. To bid the project, users only need to fil out the bid price and days. When clicking on Bid, the page is redirected to the bids list.



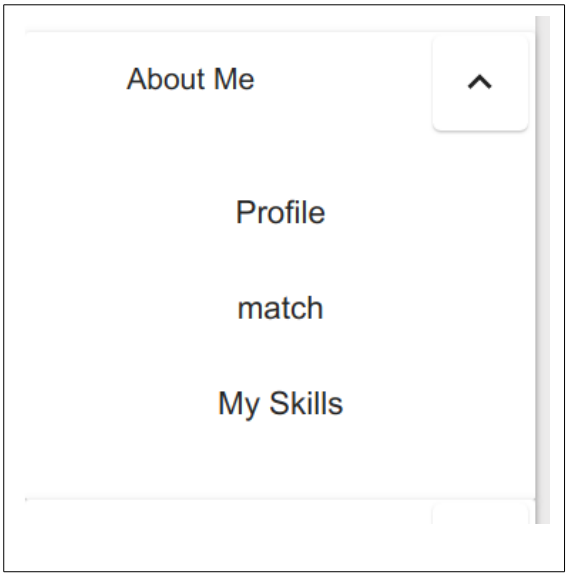
8. List of bids:
The list of bids page shows all the bids placed. Project and bidders on the list are links to the project and user show pages. "HIRE" button is only available when the current user is the owner (employer)of the project and the project is active. "DELETE" button is only available when the current user is he bidder of the project and he project is active.

Bids

ADD FILTER REFRESH

ID	PROJECT	BIDDER	ISACTIVE	BIDPRICE	BIDDAYS	
5ad23076b0b60b91d7cb85fb	project 3	xyz	✗	600	80	HIRE
5ad23076b0b60b91d7cb85fc	project 3	wer	✗	700	100	HIRE
5ad23076b0b60b91d7cb85fd	project 2	abc	✗	350	60	DELETE
5ad23076b0b60b91d7cb85fe	project 2	wer	✗	430	65	
5ad23076b0b60b91d7cb85ff	project 2	lot	✗	550	50	
5ad23076b0b60b91d7cb8600	project 2	lat	✗	660	60	
5ad23076b0b60b91d7cb8601	project 2	Yet	✗	700	50	
5ad2a348843cb372dbea8a42	Exciting Old Project	xyz	✗	600	23	HIRE
5ad38a2f73b94c0bd5de772a	project 1	abc	✓	450	23	DELETE

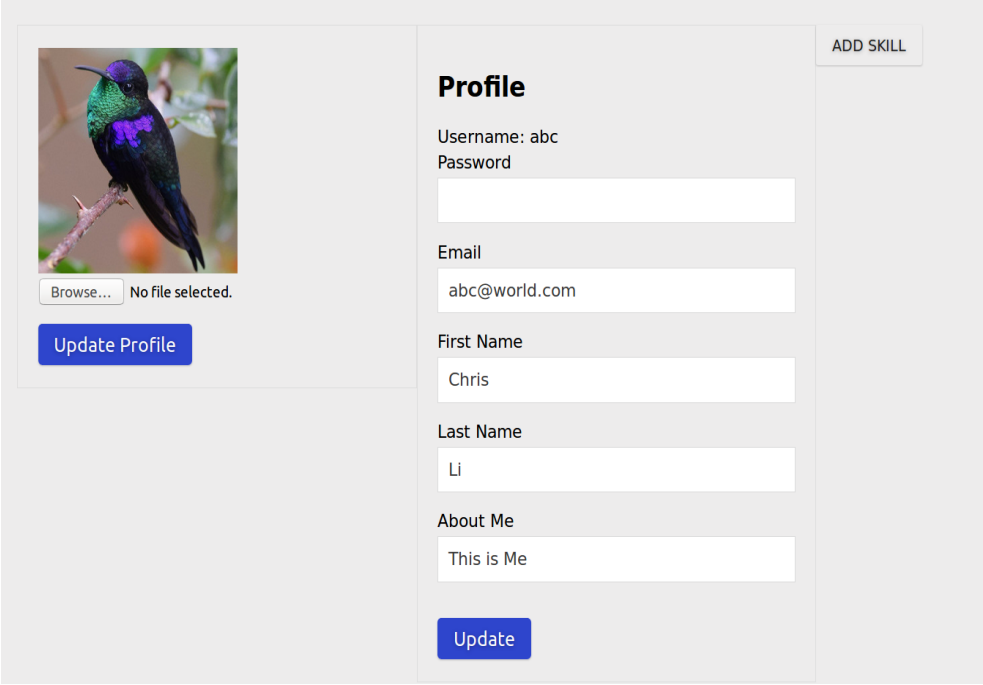
About Me:



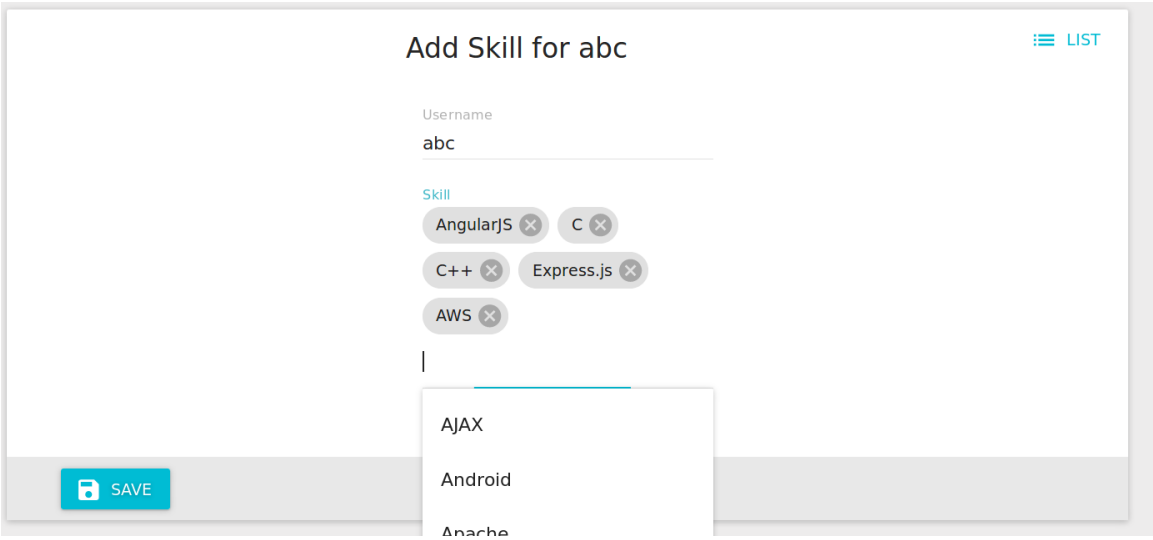
On the profile page, the user can first change his/her profile photo, on the left. Simply click at "browse", select a photo, and click at "Update Profile".

Users can also update other information on this page. Simply fill out the form and click at "Update".

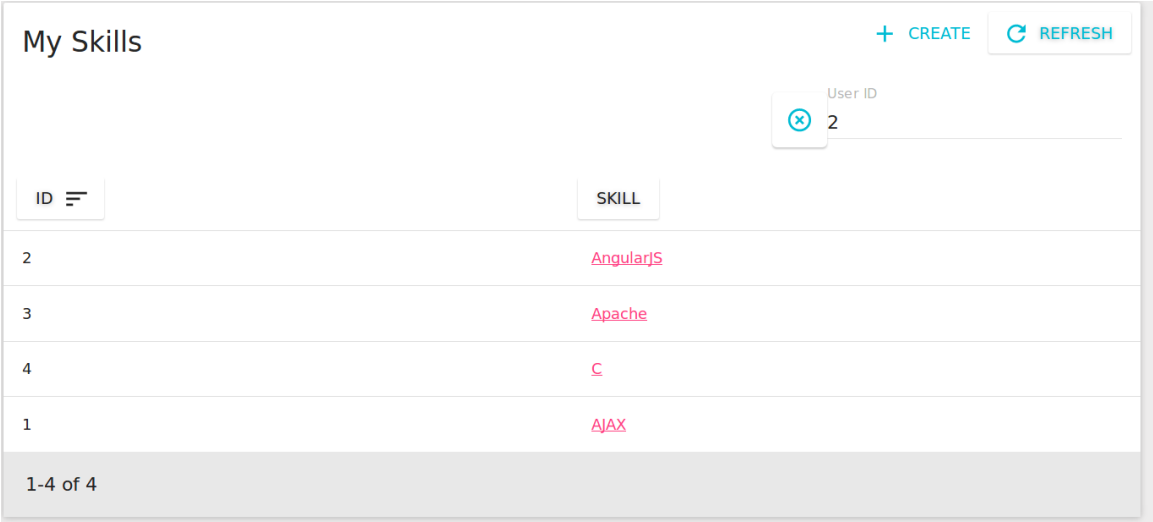
Users can add skills, by clicking at the "Add Skills" on the right.

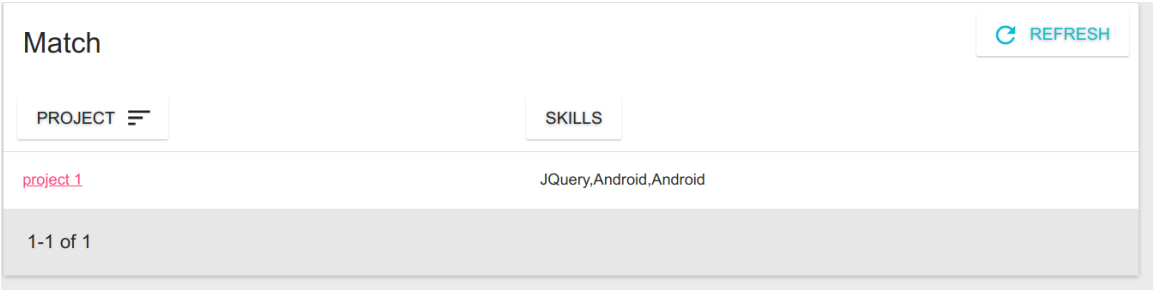


When clicked at the "Add Skill" button, users are redirected to add skill page. Username is pre-filled. Users can choose skills from a drop down list. Once click "save". The skills are saved and page is redirected to user skill list (or my skills):

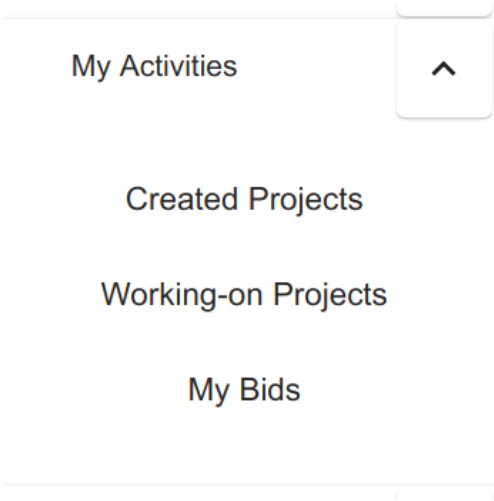


This is a my skills page, that listed the skills a user has.



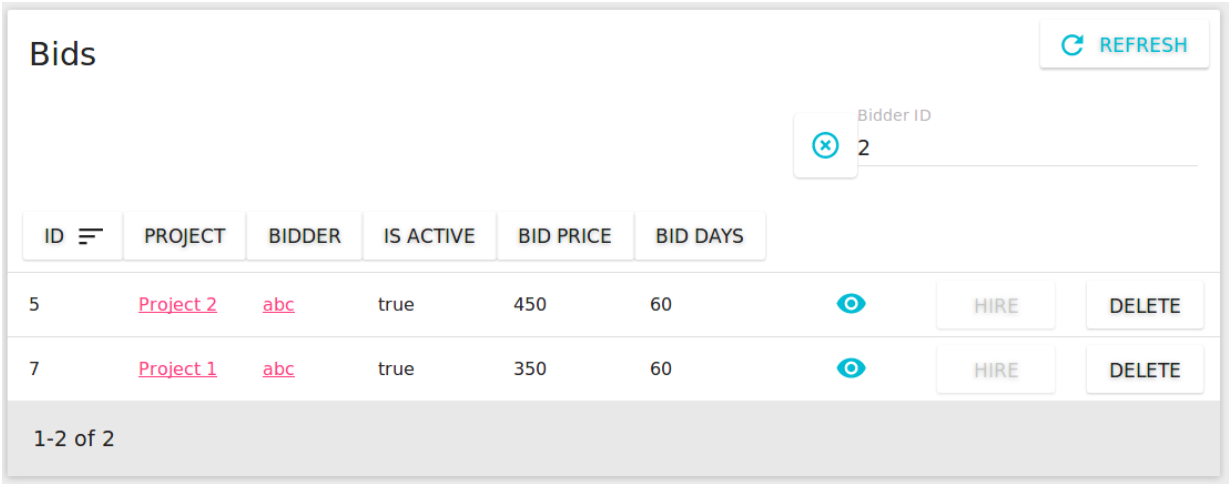


My Activities:



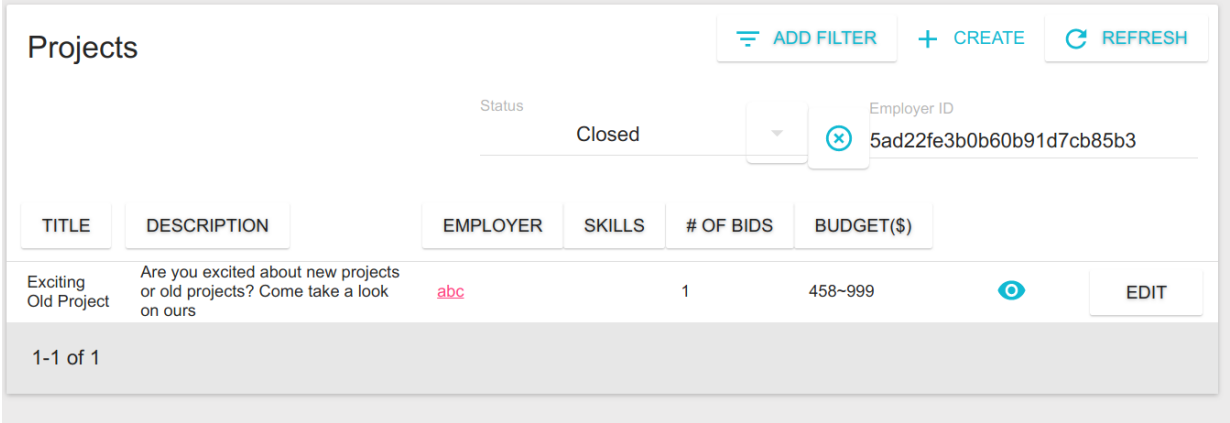
1. My Bids:

Clicking on the "My Bids" item on menu (left), brings up the list page of bids that are placed by the current user.

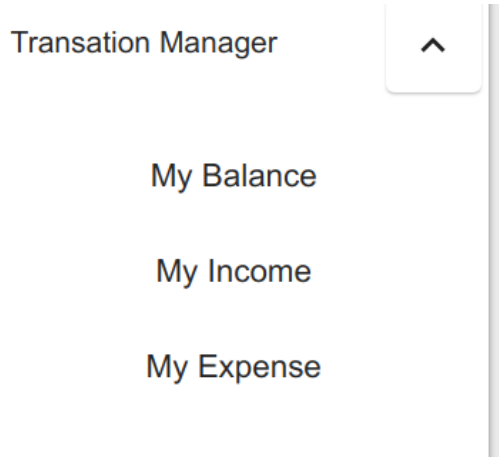


2. My Created Projects:

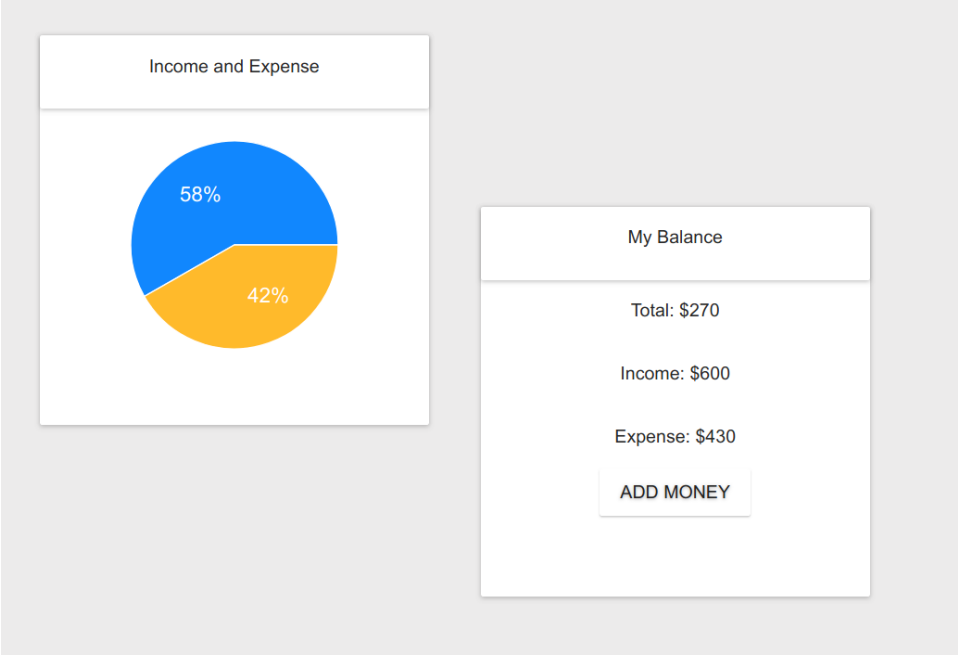
Clicking on the "Created Projects" item on memu(left), brings up the list page of projects that are created by the current user.



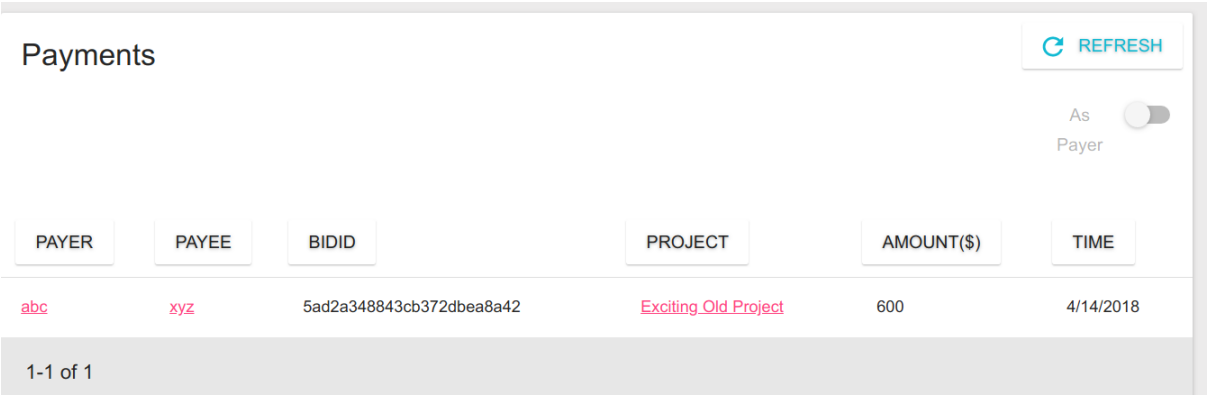
Transaction Manager:



1. My Balance:

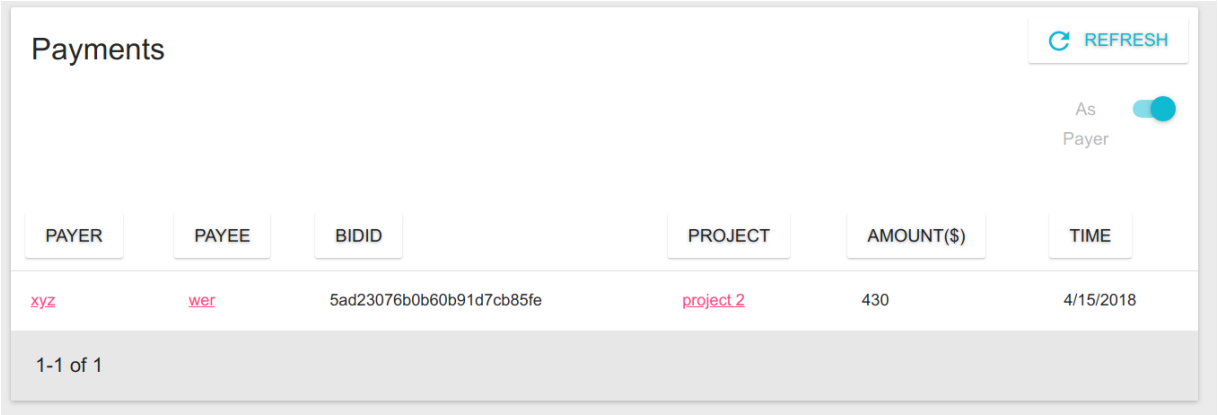


2. My Payments (Income):



3. My Payments (Expense):

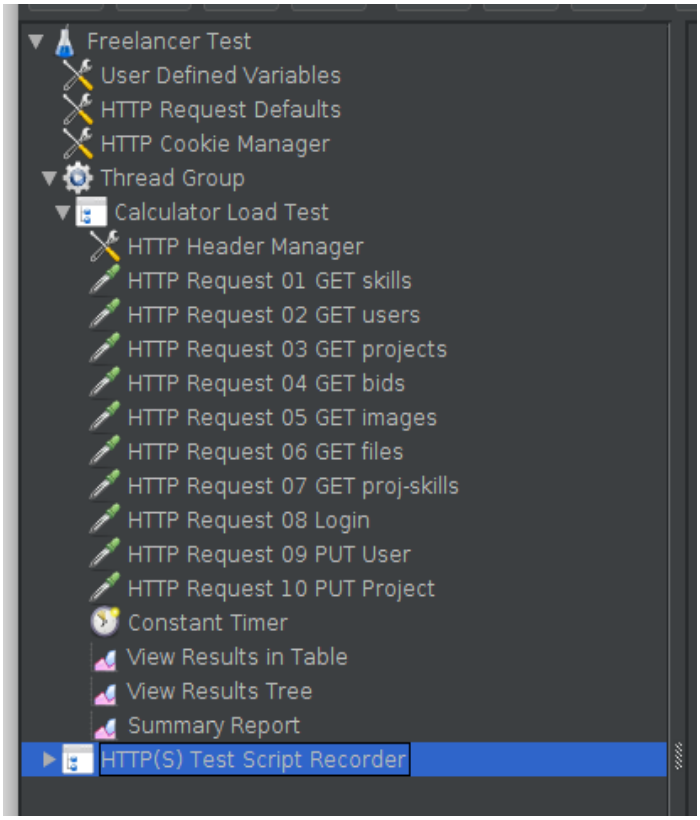
Toggle the As Payer option on top right corner

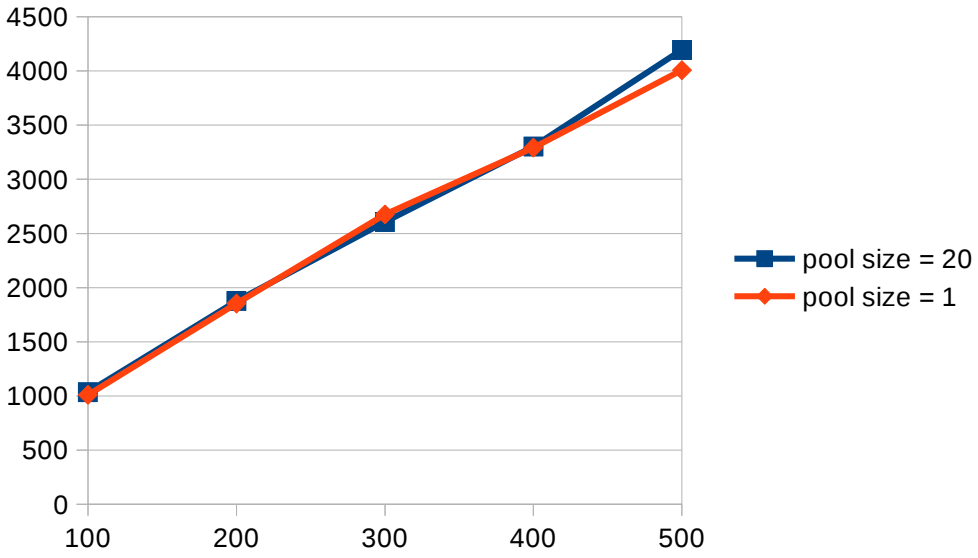


Performance:

The JMeter testing results is shown below for 100, 200, 300, 400 and 500 concurrent users with (blue, pool size = 20) and without(red, pool size = 1) connection pooling. Each user sends out 50 random requests, chosen from the 10 options, as shown below. As we can see, with connection pooling, the response time increases with a higher rate than when using connection pooling.

The better performance of the connection pooling is attributed to the design of reusing connection. Web applications usually have a large number of small db requests, that the overhead for creating and destroying a connection for each request is just not worthwhile.





Interestingly, there's no obvious difference in the performance of pool size 20 and pool size 1. We tested this system with Kafka as the messaging service. In contrast, using connection pool for mysql significantly improved the performance. One possible explanation is the fast

database operations on Mongo DB. In relational database, there are many constraints enforced that could potentially slow down a single query, which makes connection pooling critical to handle the long running query problems. As to Mongo DB, queries are usually simple without many constraints.

Mocha Testing:

```
File Edit View Search Terminal Help
> rest-server@0.0.0 test /home/chris/GitHub/cmpe273/lab2/rest-ser
ver
> NODE_ENV=test mocha --timeout 10000 --exit

REST Producer ready


  REST Server tests
    Users
Making Request:  FLC_TPC_USER GET_ALL
REST Consumer ready! FLC_TPC_USER_RS
GET /api/users 200 809.654 ms - 1669
      ✓ it should GET all users (846ms)
{ username: 'abc', password: '123' }
Making Request:  FLC_TPC_USER GET_ONE
Making Request:  FLC_TPC_SESSION POST
REST Consumer ready! FLC_TPC_SESSION_RS
POST /api/users/login 200 1107.958 ms - 402
      ✓ it should Login (1119ms)
{ email: 'my@email.com', aboutMe: 'Hello hello' }
Making Request:  FLC_TPC_USER PUT
PUT /api/users/abc 200 174.904 ms - 207
      ✓ it should update user (184ms)
    Projects
Making Request:  FLC_TPC_PROJECT GET_ALL
REST Consumer ready! FLC_TPC_PROJECT_RS
GET /api/projects 200 774.224 ms - 3391
      ✓ it should GET all projects (786ms)
Making Request:  FLC_TPC_PROJECT_FILE GET_ALL
REST Consumer ready! FLC_TPC_PROJECT_FILE_RS
GET /api/proj-files 200 737.329 ms - 207
      ✓ it should GET all files (746ms)


  5 passing (4s)
```

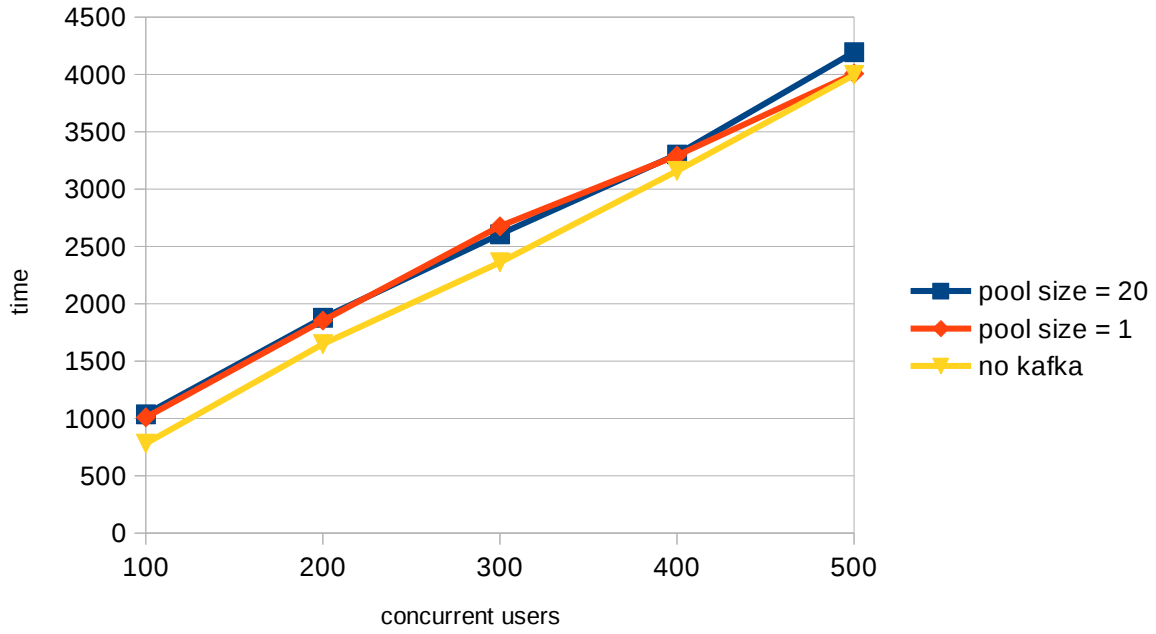
Q & A:

1. Compare passport authentication process with the authentication process used in Lab1.

In lab 1, I did not use passport in the authentication process. In stead, username and encrypted password are returned and saved in local storage. Each time when there's a need there's a need for authorization in calling the RESTful API controller, the HTTP request includes the username and encrypted password to the backend for verification. This approach is certainly secure. This information can be intercepted and used by other easily, as long as it's been saved in the local storage in a browser with the correct format.

In lab2, I used passport authentication and jwt strategy. When user login happens, our backend generates a JSON web token with a secret key. This token is then returned to the cllient. On the client side, only a JSON web token is stored. This jwt is passed through HTTP header for backend to validate. The session information, including the jwt is stored in mongo DB. When the user signs out, the session information, as well as, jwt token is deleted from the database. Hackers who stole this token, will not be able to use it, since the information is deleted from the database.

2. Compare performance with and without Kafka. Explain in detail the reason for difference in performance.



From the chart above, we can compare performance with an without Kafka. The yellow line represents data collected from non-kafka system. REST server connects directly to MongoDB, with pool size 20. We can see that the non-kafka system performs a little better than the kafka systems. One possible reason is the network latency. Our mongo db resides

on on Mlab US-Central, our kafka server is located in US-East, and our Jmeter clients are here at the we coast. A round trip in a non-kafka system is shorter. To improve the performance of kafka system, we can put kafka and mongo db close enough to reduce the latency. Another possible reason to explain this result is that the number of concurrent users is not large enough to show the advantage of Kafka. We can also argue that some tuning can be done to improve Kafka throughput.

3. If given an option to implement MySQL and MongoDB both in your application, specify which data of the applications will you store in MongoDB and MySQL respectively.

MongoDB, a NoSQL database, is quite convenient and fast for developing distributed web application using JavaScript. It has high availability, but low consistency. Thus it's not easy to perform transactions that requires ACID properties, offered by relational databases like MySQL. If separate my implementation into MySQL and MongoDB, I will make the following design changes.

MySQL: project details, project skills, bids, users, user skills, payments.

There could be different types of joins involved. For example,
1. Match project skills and user skills
2. List all projects and names of bidders
3. List all the projects that a user made payments on.

MongoDB: project files, user images, session, and authentication.

Operations on MongoDB are mostly save and retrieve. There are usually no join involved. Yet, because its high availability and quickness, it's suitable for large files and frequent interaction.