

CS 130: Software Engineering

Professor: Miryung Kim

TA: Twinkle Gupta

Propose

Part C Report

Project URL: <https://github.com/micwa/propose>

Youtube URL:

<https://www.youtube.com/playlist?list=PLgXpOPe6ghCvkoo6ccx3szaF2fViGwFow>

Team Name: NL

Team Members:

Michael Wang, 204458659

Jason Jiang, 604409327

James Kang, 904469956

Nathan Chou, 104453049

Algan Rustinya, 604670111

Frederick Kennedy, 404667930

Motivation

As the software world expands, there will always be people who need software to be written, and people willing to write that software in return for money. Our web application, “Propose”, aims to connect these types of people in a marketplace where clients can propose projects and engage freelancers to work on them.

Right now, there are no software-oriented freelancer sites that allow users to freely post project work. Our site allows anyone to sign up and either specify projects they want to be completed, or choose to work on any number of publicly posted projects. There are no restrictions on who is allowed to post projects or apply to them. Clients and freelancers can communicate requirements and progress through a special dashboard for each project.

User Benefits

Propose provides users with a convenient and simple place to either pick up a job as a freelancer or to have your idea developed as a client. Its main benefits would be in connecting freelancers that are actively looking for work with clients that are looking for people, and by connecting the freelancers and clients they could create software projects that would not have been created otherwise.

The freelancer will be assisted in the following ways:

1. Will be able to easily find work to do
2. Will be able to communicate and design the project with the client
3. Will be able to find projects that match the freelancer’s skillset
4. Can vet potential clients based on reviews by other freelancers
5. Will be able to create a record of projects done and build a reputation

Similarly, the client will be assisted in the following ways:

1. Can create a software project without the required skills
2. Will be able to connect with freelancers that are willing to work for you
3. Will be able to choose the most skilled freelancer among applicants
4. Can communicate with the freelancer to ensure that the project develops as intended
5. Can vet potential freelancers based on reviews by other clients

Feature Description and Requirements

Features

There are two main types of users on this site: freelancers and clients. Freelancers are users who are looking for work. Clients are users who post projects and pay freelancers to complete them.

Personalized Project Dashboard:

This dashboard shows the list of projects associated with a user. The user can be associated to a project either as a client who host the project or as a freelancer who work on the project. Every user has two dashboards: working dashboard and completed dashboard. A working dashboard list all of the active projects associated with the user, while a completed dashboard list all of the completed projects. A project is activated when a freelancer accept the work offer given by the client. Once the freelancer has completed every task in the project, then the project is completed.

The user can interact with these dashboards by clicking a project in the list and view the details of the project such as the description, tasks, compensation, and skills tag. If the user is the client to that project, then the user can check the progress of the project and see which tasks have been completed and which tasks are still in development. If the user is the freelancer to that project, then the user is allowed to marked tasks as completed when they are done with the task.

Project Search:

In this case, the user will act as a freelancer and search for an available project that the user can apply. A project is available for a freelancer to apply if the client of the project have not hired any freelancer to work on the project. Propose provides an advance search that allows the user to filter projects based on a specific search term and skill requirements for the project. A freelancer can easily apply a project by click the apply button and write the application message. The client of the project will then get notified about the new application and can decide whether to decline the application or to extend an offer to the freelancer. An offer has an expiry time where the freelancer can only accept the offer before the offer is expired.

Freelancer Search:

In this case, the user will act as a client who are looking for a freelancer to work on the project that the client host. The freelancer search shows a list of freelancers with its rating and bio. The client can then click a particular freelancer to see the capabilities of the freelancer in more details such as the resume, skills, and the past projects of the freelancer. Propose also provides an advance search that allows the user to filter the freelancer list based on their skills or by their name. When the client has found a suitable prospective freelancer, the client can then send an invitation to the freelancer requesting the freelancer to work on a project that the client host. The freelancer will get notified about the new work invitation and can decide to decline the invitation or to apply to the project instead.

Requirement description

Non-functional requirements are as follows:

1. Modifiability

Our project is divided into a modular backend and frontend. The backend uses Django and implements a REST API using the Django REST Framework. The API is divided into five modules: account, application, dashboard, project, tag. Each module encapsulates the models and logic related to its operations. Because communication with the frontend is done via HTTP requests, it is relatively simple to add new functionality by creating a new model and a corresponding APIView that handles GET and POST requests for it. Serialization and request parsing is conveniently handled by the REST Framework. The frontend uses React.js components, which are modular and can be easily reused.

2. Security

User accounts and authentication are required to our site. All users may browse projects, apply to projects, and view users. However, there are limitations to who may accept applications, withdraw applications/offers, and view full project details (and more). For example, once a user creates an application, only that user may withdraw it. Offers can only be made by the client and accepted by the freelancer it belongs to. For the full project view that includes tasks and comments by the client and freelancer, only they are able to view and add tasks and comments. Only the client may update their own project's details.

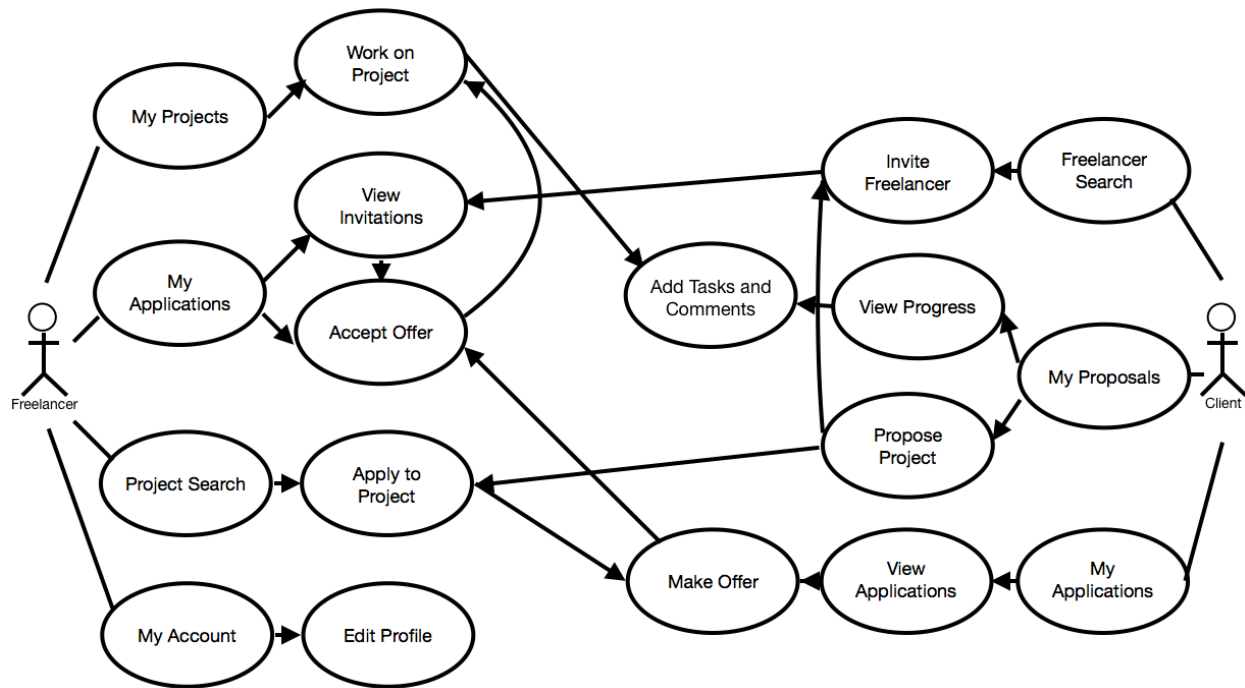
3. Testability

As mentioned before, the backend is decoupled from the frontend. The two communicate using a REST API which can be tested using standard HTTP requests. The backend is separated into modules which contain all related functionality. Classes are represented using Django models and request handling logic is done using APIViews from the Django REST Framework.

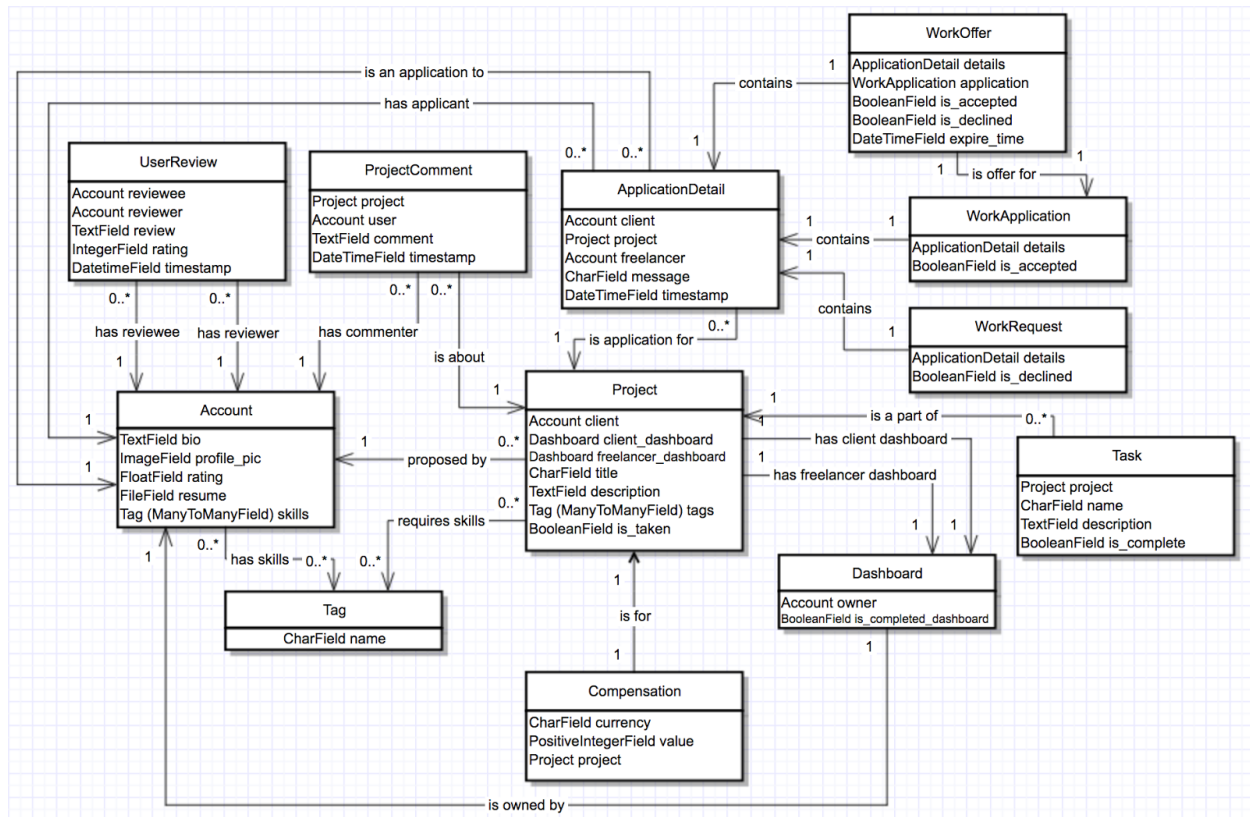
4. Usability

The site is designed to be user-friendly and easy to navigate. To use the site, users first have to create an account. From there, the user lands on the homepage with a navigation bar containing six tabs: project search, freelancer search, my projects, my proposals, my applications, and my account. The project search and freelancer search tabs are self-explanatory. One can also filter project/freelancer by keyword and tags/skills. The user can check the status of applications, offers, and requests in the "my applications" tab. "My projects" contains projects that one is working on, as either a client or freelancer; and "my proposals" contains projects that one has posted.

Design



Use Case Diagram



UML Class Design

We are using the Django REST Framework to implement our APIs. With RESTful API design, all our backend logic is now encapsulated by well-defined endpoints. Our Web Application on the backend is represented by 5 Django apps otherwise known as modules (Account, Application, Dashboard, Project, and Tag). All requests and responses are in JSON format. For example, POST requests that change an instance of a model will make a request to the API with the input parameter being a JSON representation of the model. To edit user information, one can pass in a user JSON object as the input parameter to the API call. Using the RESTful API design corresponds to the information hiding principle because changes to models will not affect the API. Clients only will need to modify how they process the API requests and responses. Our endpoints are designed to be intuitive and correspond to particular models (classes). Anytime a model is returned, we will expect a JSON object representation of a model to be returned.

API Description

Our project uses a RESTful design for our API. Endpoints are specified by URIs, which the client communicates with via HTTP GET, POST, etc. requests. The parameters for GET requests are part of the query string. The parameters for POST requests are objects serialized from Django models (see UML). Below is a detailed breakdown of the endpoints by module.

Account module:

HTTP Request	URL	Parameters	Return value	Description
GET	/api/users	tags, search_terms	Account[]	Given an optional array of tags and search terms, return a list of Account objects that match the tag and search term criteria
POST	/api/users	user=Account	HTTP Status Code	Creates a user
GET	/api/users/<id>	None	Account	Given the id of a user return the user page of that user
GET	/api/users/<id>/review	None	Review[]	Return the reviews for the given user

POST	/api/users/<id>/review	review=Review	HTTP Status Code	Creates a review for the given user
GET	/api/profile	None	Account	Returns the account of the logged in user
POST	/api/profile	account=Account	HTTP Status Code	Update the account object of a user

Project module:

HTTP Request	URL	Parameters	Return values	Description
GET	/api/projects	tags, search_terms	Project[]	Given a list of tags and and search terms, return a list of projects the satisfy the search criteria
POST	/api/projects	project=Project	HTTP Status Code	Create a new project
GET	/api/projects/<id>	None	Project	Return the project with the given id
POST	/api/projects/<id>	project=Project	HTTP Status Code	Update the project with new information
DELETE	/api/projects/<id>	None	HTTP Status Code	Delete the project with the given id
GET	/api/project/<id>/comments	None	Comment[]	Returns a list of comments for a project with id <id>
POST	/api/project/<id>/comments	comment=Project Comment	HTTP Status Code	Adds a comment to a project with id <id>
GET	/api/project/<id>/tasks	None	Task[]	Returns a list of tasks for a project with id <id>

POST	/api/project/<id>/tasks	task=Task	HTTP Status Code	Adds a task to a project with id <id>
GET	/api/project/<id>/tasks/<id>	None	Task	Returns a specific task
POST	/api/project/<id>/tasks/<id>	task=Task	HTTP Status Code	Edit a specific task

Application module:

HTTP Request	URL	Parameters	Return values	Description
GET	/api/applications	None	Application[]	Returns a list of the user's applications
POST	/api/applications	application=Work Application	HTTP Status Code	Creates a new application
GET	/api/applications/<id>	None	Application	Returns the application with id <id>
DELETE	/api/applications/<id>	None	HTTP Status Code	Deletes an application with id <id>
POST	/api/applications/<id>/decline	None	HTTP Status Code	Decline an application with id <id>
GET	/api/applications/<id>/offer	None	Offer	Return the offer for an application with id <id>
POST	/api/applications/<id>/offer	offer=WorkOffer	HTTP Status Code	Create an offer for an application with id <id>
DELETE	/api/applications/<id>/offer	None	HTTP Status Code	Delete an offer with id <id>

POST	/api/applications/<id>/offer/<id>/accept	None	HTTP Status Code	Accept the offer for the given application, add the project to corresponding client's and freelancer's dashboards
POST	/api/applications/<id>/offer/<id>/decline	None	HTTP Status Code	Decline the offer for the given application
GET	/api/requests	None	Request[]	Returns a list of requests for a user
POST	/api/requests	workRequest=WorkRequest	HTTP Status Code	Creates a request
GET	/api/requests/<id>	None	Request	Returns a request with id <id>
DELETE	/api/requests/<id>	None	HTTP Status Code	Delete a request with id <id>
POST	/api/requests/<id>/decline	None	HTTP Status Code	Decline a request with id <id>

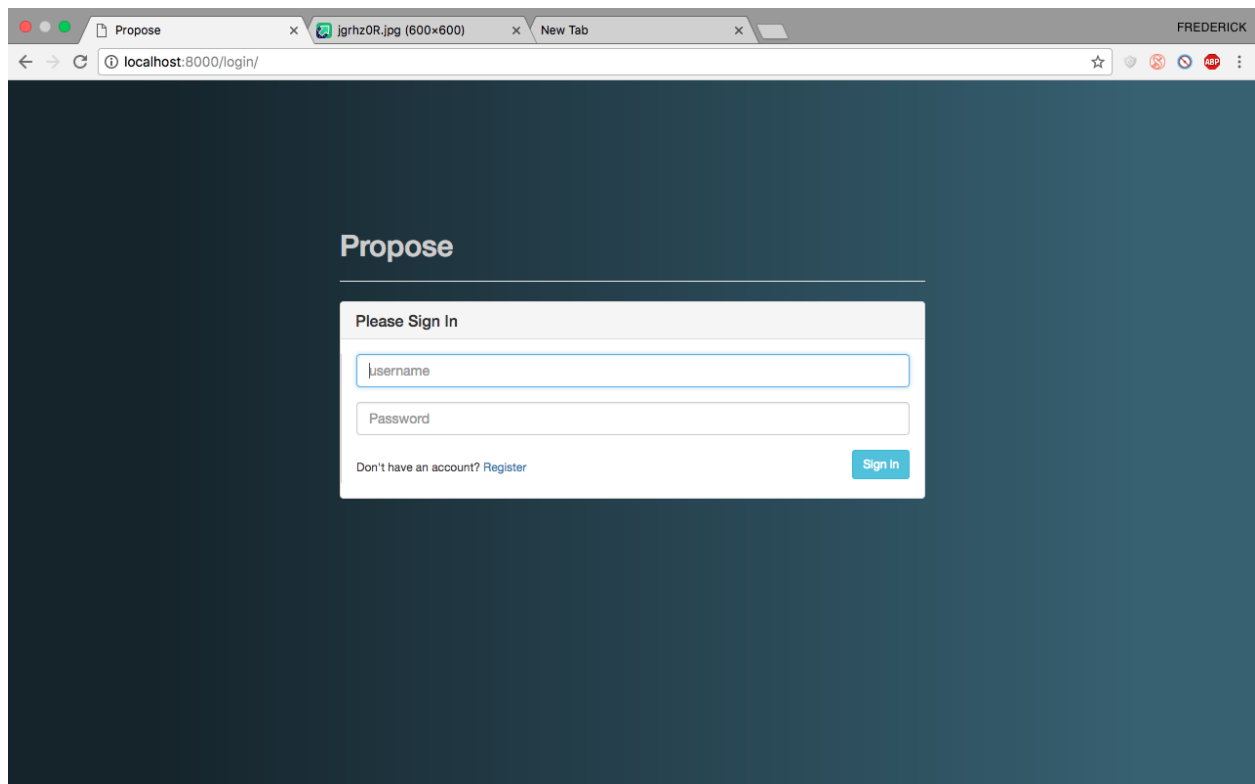
Dashboard module:

HTTP Request	URL	Parameters	Return values	Description
GET	/api/dashboard/working	None	Project[]	Returns a list of uncompleted projects
GET	/api/dashboard/completed	None	Project[]	Returns a list of completed projects
GET	/api/dashboard/user/<id>/working	None	Project[]	Returns a list of uncompleted projects of a specific user
GET	/api/dashboard/user/<id>/completed	None	Project[]	Returns a list of completed projects of a specific user

Tag module:

HTTP Request	URL	Parameters	Return values	Description
GET	/api/tags	None	Tag[]	Return all tags
POST	/api/tags	tag=Tag	HTTP Status Code	Create a new tag

User Interface Snapshots



Login page

Propose

localhost:8000/register/

Propose

Register

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email address

Password*

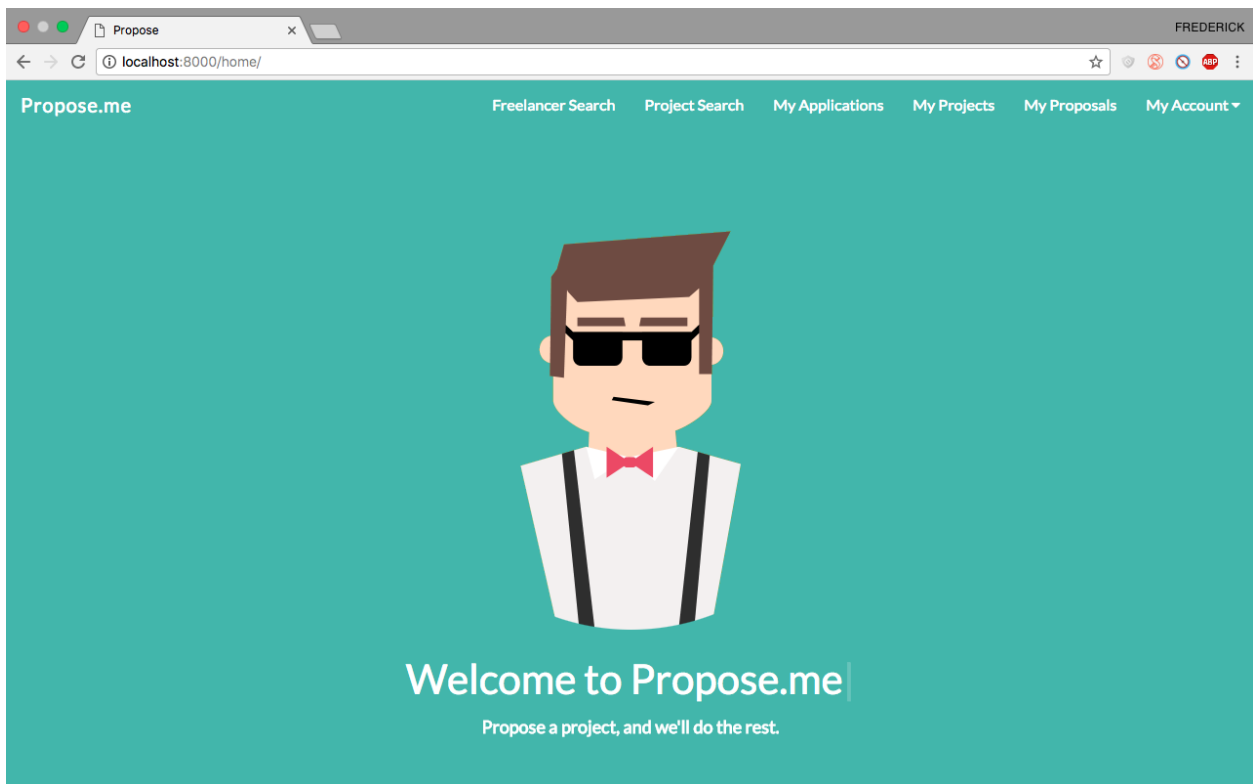
- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation*

Enter the same password as before, for verification.

Already have an account? [Sign In](#)


User Register page



Splash page

Propose.me

Freelancer Search Project Search My Applications My Projects My Proposals My Account




Potato Peter

Profile Summary

I'm a front-end developer!ssss

Rating

4/5 

About Work Info Reviews

Phone 9256819639

Email fkennedy@ucla.edu

Facebook facebook.com

Twitter twitter.com

Website website.com

Completed Projects

Skills


Node.js

Django

Profile page

Propose.me

Freelancer Search Project Search My Applications My Projects My Proposals My Account




Algan Rustinya

Profile Summary







I'm a back-end developer!

Rating

3.3/5 

Leave a Review

About Work Info Reviews

	Potato Peter Hello!	Rating 
	Potato Peter He's a great person!	Rating 
	Potato Peter You're a potato	Rating 

Completed Projects

Skills

Node.js

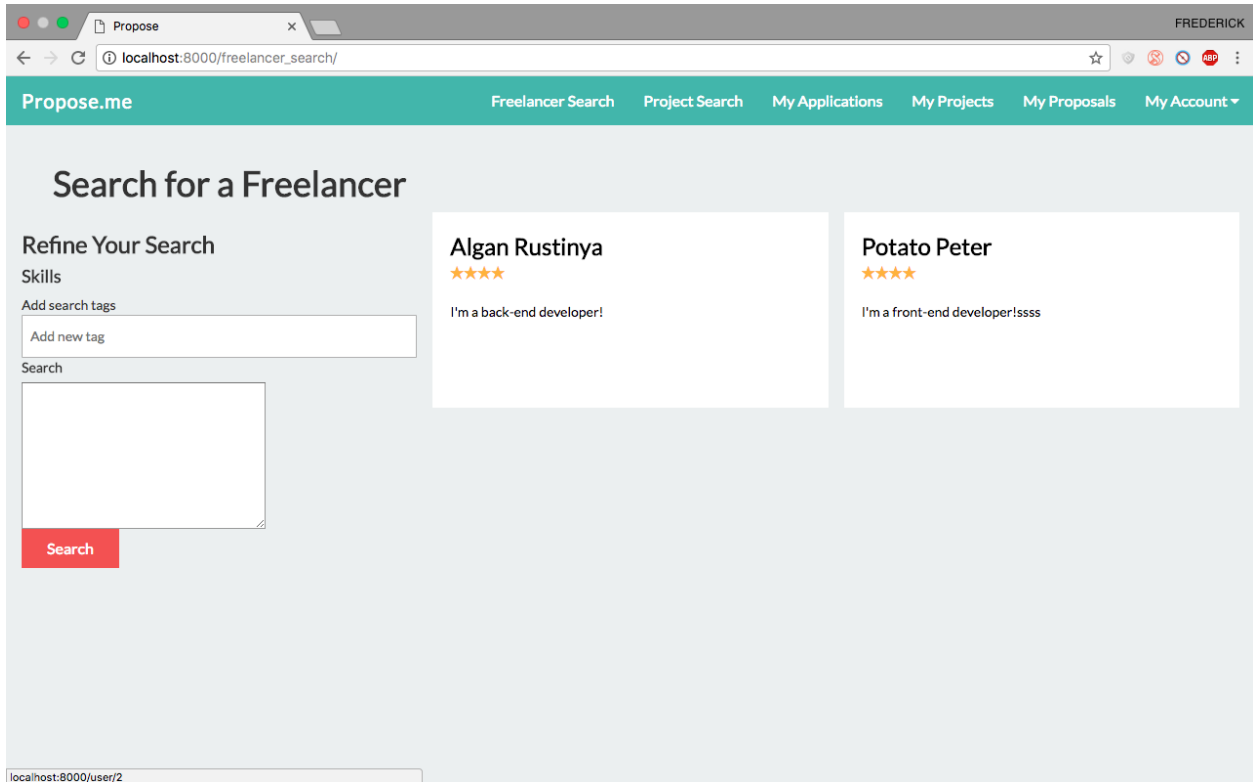
Django

Python

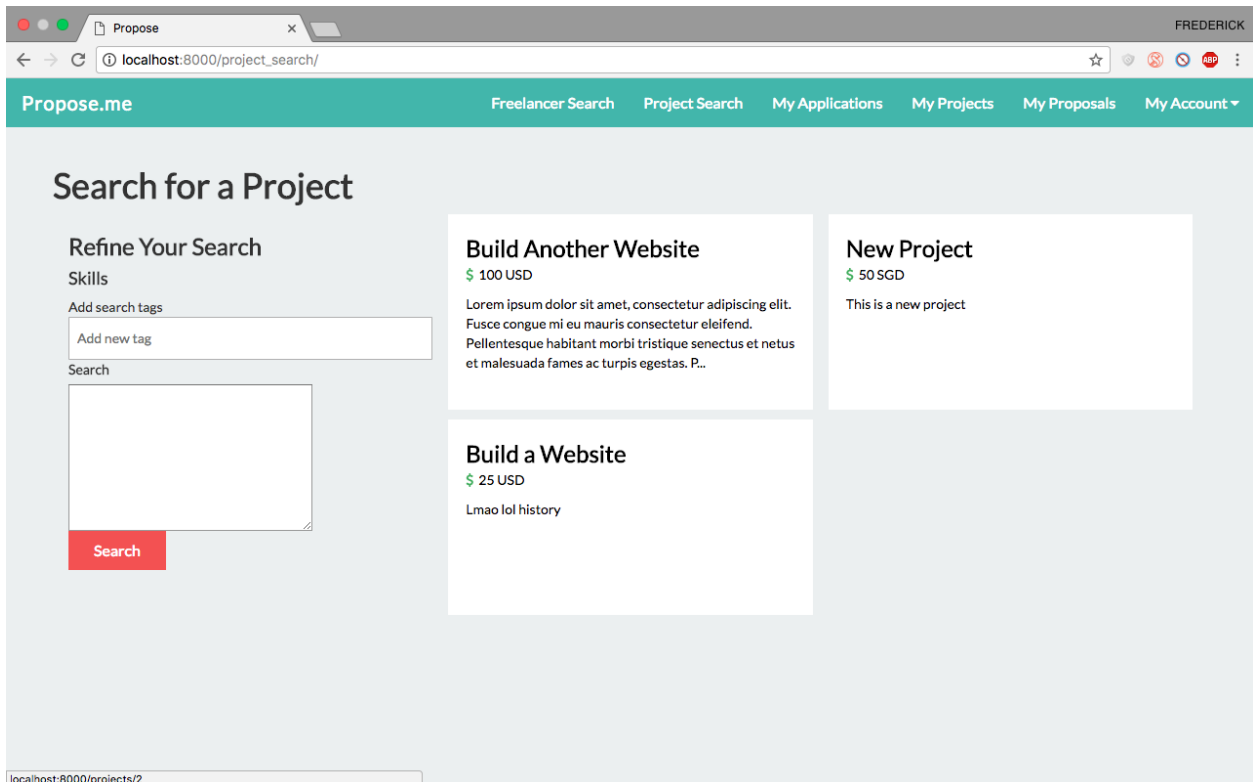
C++

Java

Profile page viewed as other user (with reviews)



Search for a Freelancer



Search for a Project

Propose.me

Freelancer Search Project Search My Applications My Projects My Proposals My Account

Edit Profile

First Name: Potato

Last Name: Peter

Email: fkennedy@ucla.edu

Biography: I'm a front-end developer!ssss

Skills: Node.js Django Add new tag

Edit

Edit Profile page

Propose.me

Freelancer Search Project Search My Applications My Projects My Proposals My Account

Edit Project

Project Title: Build a Website

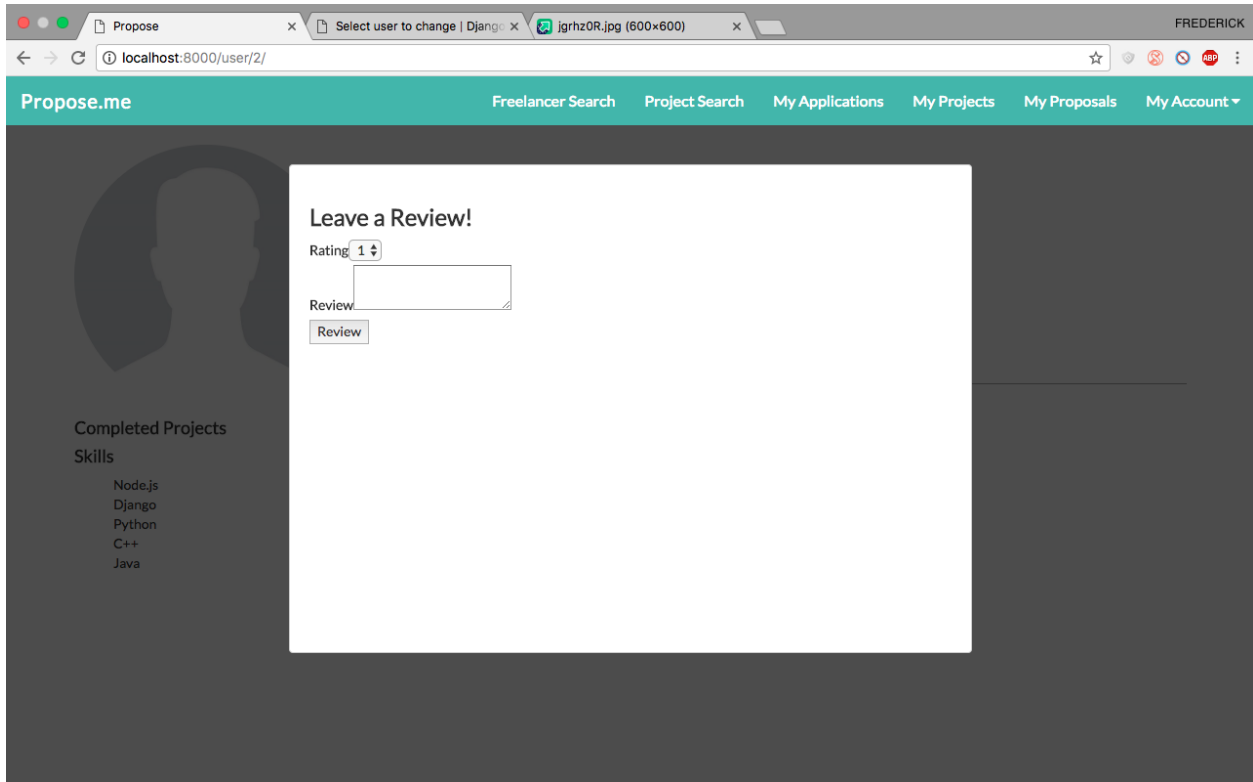
Description: Lmao lol history

Compensation: USD

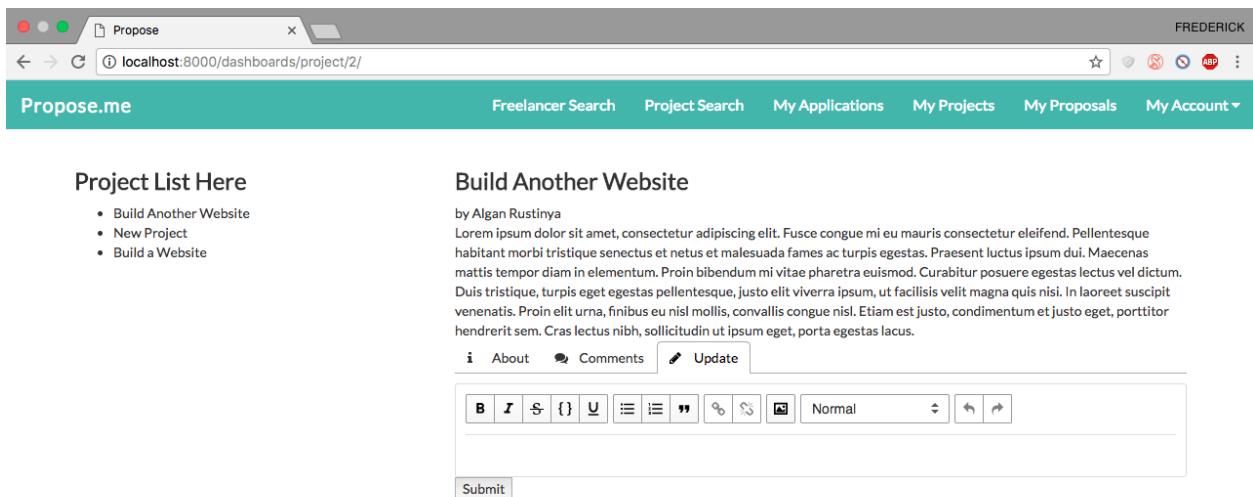
Tags: 25

Edit

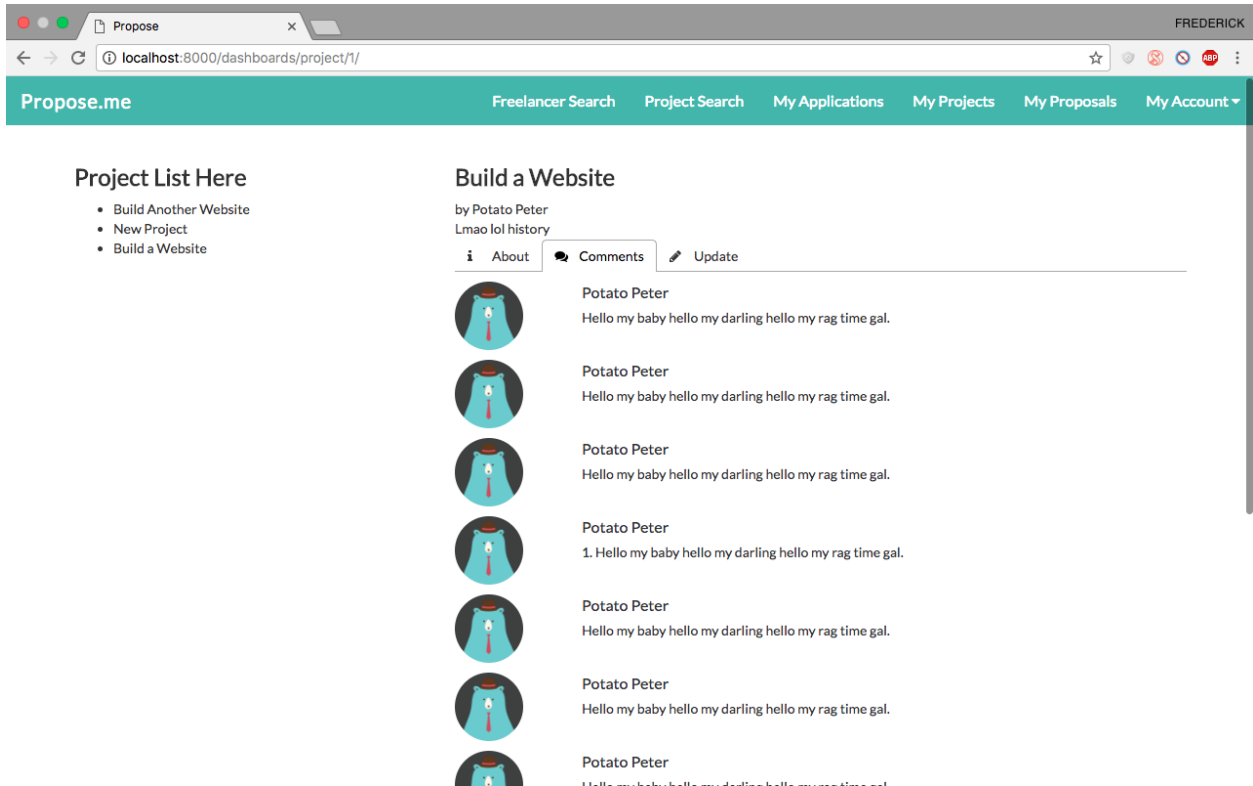
Edit project page



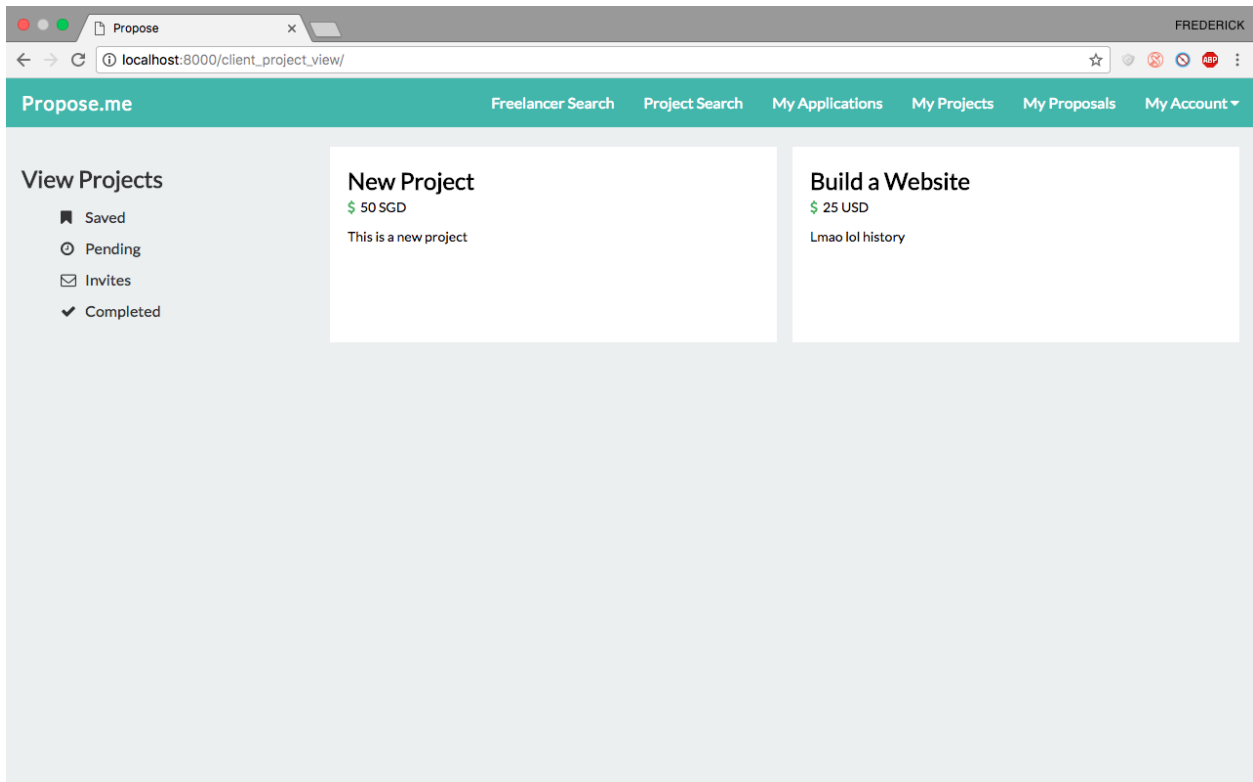
Modal to leave a review for a user



Project Dashboard to update comments



Project dashboard comments



My Projects page

Test Scenarios & Test Cases

The server API calls were tested using the built-in testing framework provided by Python, Django, and Django Rest Framework. Specifically, Python provides general unit testing capability (for example, assert functions), and Django and Django Rest Framework adapt this capability to test web applications and RESTful API's.

Since our application's classes are highly interrelated, generating sample test objects was done through `factory_boy`. This provides helper classes and methods to create *factories*, which generate test objects for not only a particular class but also its dependent classes. For example, a `ProjectComment` belongs to a particular `Project` and `Account` (the commenter). `factory_boy` allows us to simply ask `ProjectCommentFactory` for a sample `ProjectComment` without having to worry about the dependent `Project` and `Account` objects.

Each API call has a corresponding unit test. In general, each API call's test:

- Ensures that the user must be logged in/authenticated to make the API call. Otherwise, a 403 Forbidden should be returned.
- Ensures that the API call fails nicely when invalid input is provided. For example,
 - a 404 Not Found should be returned if an id not corresponding to any database row is passed;
 - a 403 Forbidden should be returned if the user tries to modify data (projects, applications, etc.) that they do not own;
 - a 400 Bad Request should be returned if the user sends malformed or incomplete data.
- Creates necessary sample test data using factories mentioned earlier.
- Makes the API call.
- Compares the API call with what was expected based on existing test data.

Moreover, each module's tests are preceded by a setup function that creates an sample user account that represents the logged-in user performing each test action.

The unit tests are found in the `tests.py` file inside each module. For example, the project tests are located at <https://github.com/micwa/propose/blob/master/propose/project/tests.py> .

Contributions

Nathan Chou fixed some implementation bugs in the API that arose in the last part of the project. He mainly focused on learning to use the Django testing framework and `factory_boy` to write unit tests for each API call, which helped discover some bugs. Accordingly he also wrote the Test section of this report.

Jason Jiang worked side by side with Frederick on developing the front end. He is responsible for most of the information handling and passing between React components. Jason developed the base/foundation and skeleton of the pages and coordinated with Frederick on styling the pages. Jason and Frederick decided to breakdown the frontend into a series of components with three key stages of development: Skeleton components, component functionality, and polish (css). Jason worked on the skeleton components on the majority of the pieces as well as the functionality of the components (making api calls) to give Frederick enough time to style all the components to a complete degree.

James Kang worked on implementing the backend API, in particular the search functionality that is involved in getting user accounts and project pages, working with Michael. For the report, he wrote the user benefits and design sections, and created both the use case diagram and the UML class diagrams.

Frederick Kennedy worked a lot with Jason on developing the front end. He styled all the pages and decided with Jason on what React libraries to use. A huge portion of the application is making the project work by reusing React components instead of developing one for each page/functionality. Frederick developed the Profile page and the Edit (Profile, Project) pages.

Algan Rustinya implemented a large portion of the backend API and updated several end-points. He created the login and registration part of the front end while handling the authentication in the backend. He also made sure that the API endpoints worked with the frontend. In this report he wrote the feature section.

Michael Wang was responsible for design/API decisions and worked with Algan to resolve issues with views/models. He implemented the tags API and part of the projects API. He directed the frontend on how to use the backend API from a REST perspective, and helped finalize the site layout. For the report, he wrote the non-functional requirements section and API description. He and Jason made the demo video.