MigrantHub

Grade for Release 1

Team members

Name and Student id	GitHub id	Number of story points that member was an **author** on.	Role
Mira Marhaba (27497148) *miramarhaba@gmail.com*	miramarhaba	30	Backend, Frontend, Cloud, ML
Alexandre Masmoudi (27755473) *a-masmoudi@hotmail.com*	Smoudii	38 + 8 (491)	Backend, Frontend, Cloud, ML
Laxman Velauthapillai (40000111) *laxman_20@hotmail.com*	iamlax	31	Backend, Frontend, DevOps
Tusman Akhter (40003476) *tusman96@hotmail.com*	tusmanakhter	37	Backend, Frontend, DevOps
Tajbid Choudhury (40002177) *tajbidhussain@gmail.com*	CodeTaj	29	Backend, Frontend
Mazen Nahle (27003315) *nahlemazen@gmail.com*	mazennahle	29	Backend, Frontend
Rajeevan Vairamuthu (40000112) *rvairam10@gmail.com*	rajee10	29	Backend, Frontend

Project summary

A social network application geared at international students, immigrants, refugees, and prospective refugees. This web app aims to help newcomers transition into their new life in Canada. The application aims to centralize services for newcomers in one place for easy access. It also aims to create a space where newcomers can interconnect and help one another. A user can create a profile and join communities that he or she is interested in. They can also register for services that will be integrated into the application. These services can be provided by both governmental and non-governmental organizations to help migrants. The application will gather user data about the services used by the migrants. This will include questions they need answers to or specific services that are most helpful to them. The data will be used to provide migrants with better choices.

Risk

In order for our application to provide accurate and relevant suggestions to our users, there is a high risk associated with the need to gather data at such an early stage of development. Without it, users will be unable to later get the guidance that is needed. To attack this most significant of risks, we made sure to begin with issues that cover the sign up and login phase (#4, #9, #16). By doing this, we can make sure to collect much of the later-needed data from the user about their status. We also ensured that they have the ability to modify their information (#2) to ensure that our data is not limited to a snapshot at one point of time, but that conclusions can also be drawn by analyzing the data over time. It is also important that the data obtained is widespread over many features, which is why we tried to implement early versions of as many features as we can (ex. Friends, Services/Reviews, Events), to maximize and diversify these aforementioned conclusions. Gradually, this information will be used to begin with the data analytics that will enhance the application's performance and user experience.

Another risk is that we must have actual users to use this application in order to get the data required. To solve this issue of getting critical mass, our stakeholder will provide us with the network of migrants and people that he has contact with. Another risk he will help us mitigate is providing us with data on multiple services for newcomers that he already knows about, this will help us centralize the services in one place without having to do too much research.

Finally, we will use Google Analytics once the application is being hosted to gather data on our users, how they use the application, and which pages are more visited. We will also use various tools from the Google Cloud Platform, such as TensorFlow and the Cloud Machine Learning Engine to craft models with the collected and already existing data. These models will allow us to learn insights on our users and create a recommendation system to provide them with a personalized experience on our application.

Legal and Ethical issues

- Data being acquired by unauthorized users and used in a way that can harm the migrants.
    - Solution: We only expose data that does not pose harm to users. All sensitive information is kept private from unauthorized users. Endpoints serving sensitive information are protected.
- Organizations creating services or events to take advantage of migrants
    - Solution: This is harder to identify, but if this content is flagged, it will be removed from the website.

- Spam
    - Solution: Monitoring the website through an admin dashboard and removing spam content will reduce this.
- Security concerns: hacking
    - Solution: Use of best practices in securing api endpoints, client routes and the database will prevent hacking. Also, force users to use a secure password.
- Lack of moderation: i.e. someone creating a random event.
    - Not adequately moderating reviews causing misrepresentation of services due to the fact that these services could be crucial
    - Solution: Admin managing and reviewing posted events and services, removing useless content.
- Third Party Content/Copyright Infringement: i.e. someone posting a text, picture or video from another site.
    - Solution: We transfer the right of copyright to the users
- Criminal Activity:
    - Solution: Use of the application for illegal purposes are forbidden. Ban the users that are using the website for criminal activity.

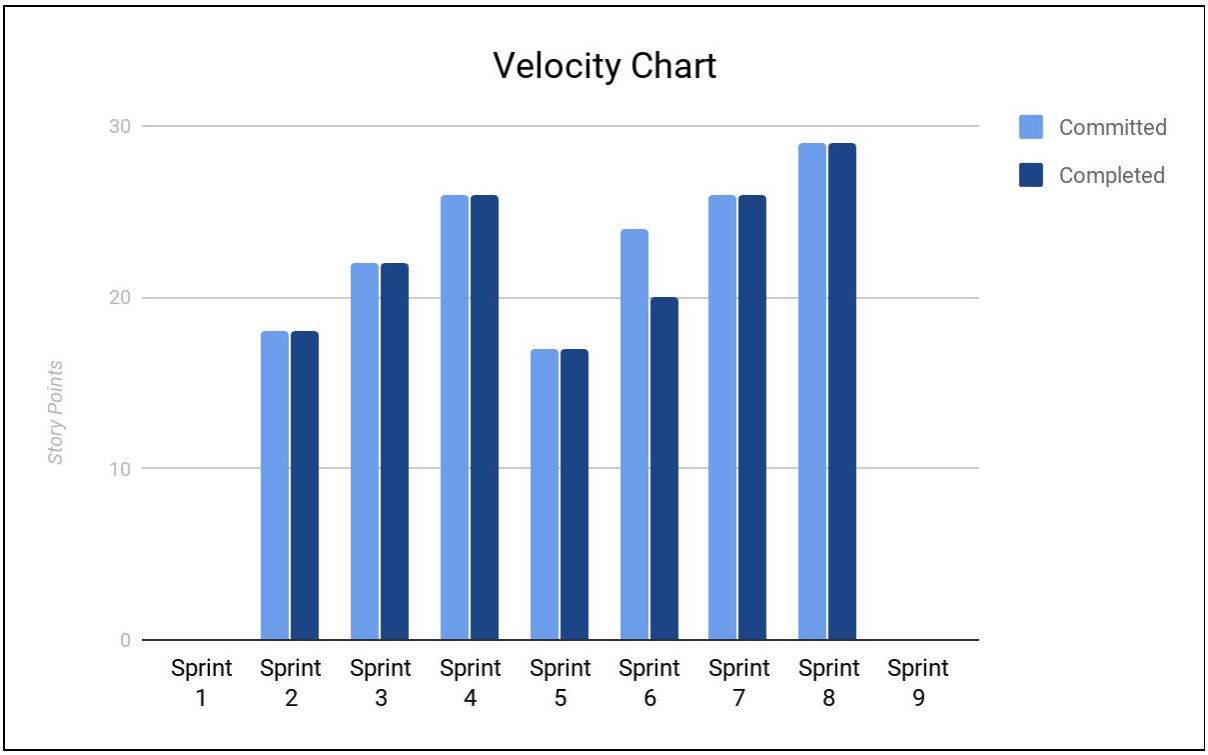
In order to mitigate these issues the site will need moderators or an approval method when users want to post services. The application will use standards of security in order to prevent hacking.

Hardware/cloud/compute requirements

- NodeJS
- MongoDb
- Docker
- Google Cloud Platform → with Kubernetes Engine
- Visual Studio

Velocity

Project Total: 51 stories, 161 points over 21 weeks



Iteration 1 (0 stories, 0 points)

Our main achievement for this iteration was getting the requirements from the stakeholder and coming up with user stories. Also, through discussion with the stakeholder, we decided on the tools and frameworks we are comfortable using and we setup a base architecture to use for the project.

Iteration 2 (3 stories, 18 points)

In this sprint, we setup the main signup functionality for both migrants and businesses as well as login. These forms are not normal signup forms since they include much more information than a normal website signup. We also created a homepage for users to be redirected to upon login. We also setup our testing infrastructure for both the backend and frontend.

Iteration 3 (7 stories, 22 points)

In this sprint, we added the following functionalities: the ability for an admin to sign up to an account, a migrant being able to add, accept/reject and see friends, creating events, creating and viewing services and editing your migrant or business profile. We also updated our testing infrastructure to have better tests.

Iteration 4 (9 stories, 26 points)

In this sprint, we added the following functionalities: the ability for a user to edit/delete an event, edit/delete a service while also being able to review other services and searching for services, view a service's reviews, remove a friend from friend list and finally managing admins. Furthermore, we ensured that all our system is properly storing users data without ever deleting them as it is a crucial step towards our goal of performing data analytics.

Iteration 5 (8 stories, 17 points)

In this sprint, we worked on being able to delete a service in certain cases, delete a review, and referring a service to other migrants. We ensured that a migrant/merchant can view their profile as opposed to just being able to edit it in the previous iteration. We are now also able to search for people and delete events. The Data Analysis (recommender engine) has also been given a preliminary look, to allow us to plan for the coming sprints.

Iteration 6 (6 stories, 23 points)

In this sprint, we finally started to take a look at the recommender engine portion of the project, which is extremely important as it plays one of the largest roles for our interface and helps achieve the ultimate goal. To do this, the most important step was to deploy and host the app online in order to start gathering user data since Google Analytics requires an app that is already hosted. We also inputted the data we already had about the different existing services, so that we are ready to implement a data model. Since we were unable to finish implementing the data model (mentioned in more detail below under 'hosting'), we assumed only 3 of the 8 points as completed. We refactored our sign up to ask for less information and added a feature to ask questions to the questions while they are browsing the site. We also integrated Google login. In addition we refactored our architecture to have service and repositories for better separation of concerns and added better api endpoint checks and protection.

Iteration 7 (7 stories, 26 points)

In this sprint we redesigned the entire app including events, services, homepage, signup, and login pages. We also added a feature to allow migrants to suggest services to MigrantHub in order for us to expand our services provided. We changed search functionality to do a global search. Directions from google maps were added for services and events. In addition, we started adding a chatbot to our application. Lastly, we continued the second part of the recommender engine portion, in which we created the ML Engine that we will use for our first data model which will recommend services based on previous service reviews.

Iteration 8 (11 stories, 29 points)

In this sprint we continued the work brought on by the UI refactor, by revamping the layout of some of our forms and the profile display and admin dashboard, and by ensuring that this UI can also be used on mobile screens as well. We gave the user the ability to add items to a calendar, and to also be able to translate the website into any of the languages that Google supports. We created a section for recommended services and after finishing the ML model we connected the two so that the users can have services suggested to them. Lastly, we also continued working on the chatbot to be able to demonstrate it for the end of the release.

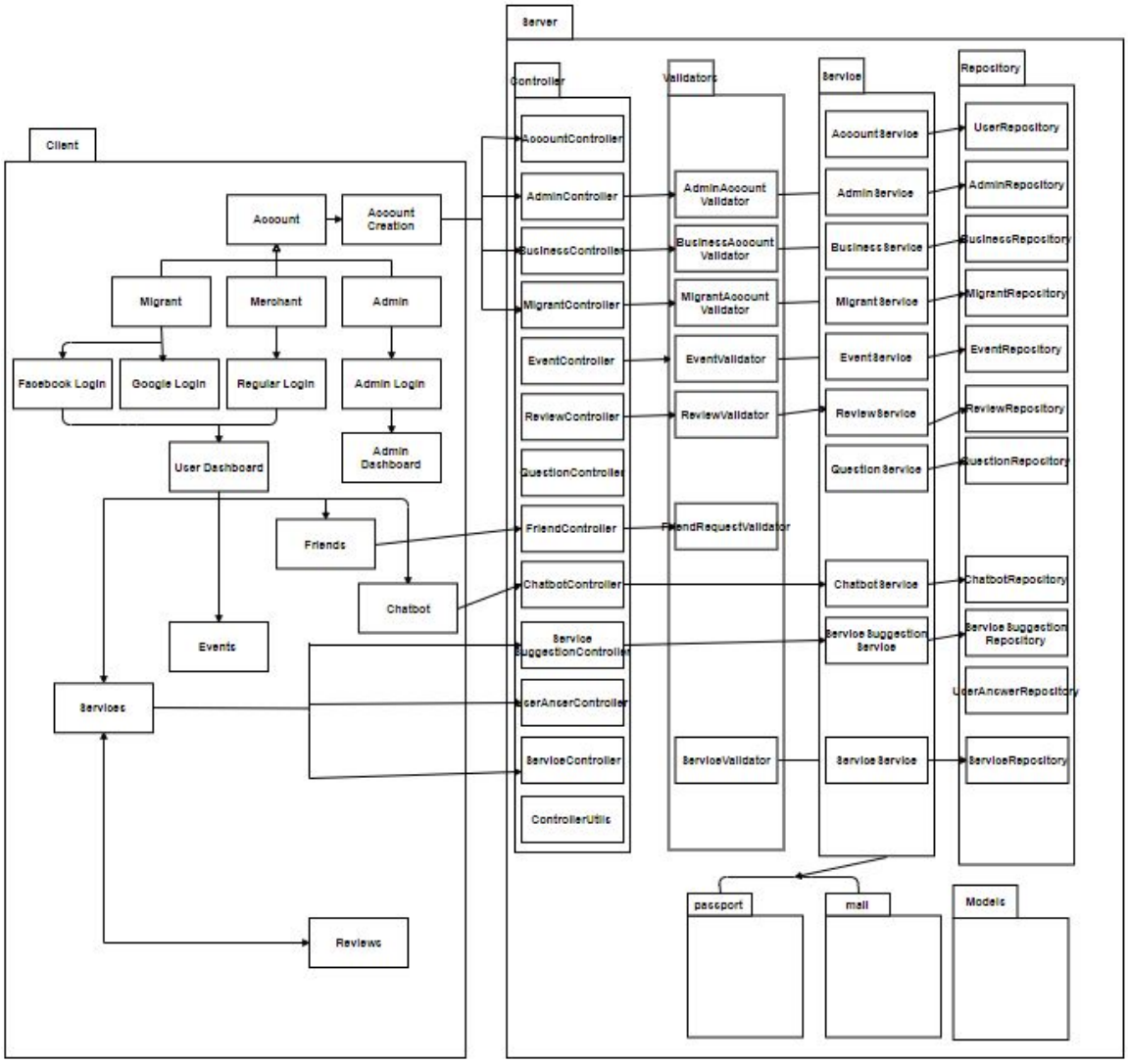
Iteration 9 (7 stories, 26 points) (PROJECTED)

In this sprint we will add many new features to the chatbot, such as having it navigate to different parts of the page and recommend services to the user. It will also be given alot more dialog capabilities. On the ML side, we will ensure that the recommendation models are updated automatically on a regular schedule (as opposed to manually) and that the service recommendations use some extra features in the calculations. We will create a page to post jobs, another page to post housing information, and another page which will include forum functionality for the migrants. On the UI side we will fix browser compatibility for the user.

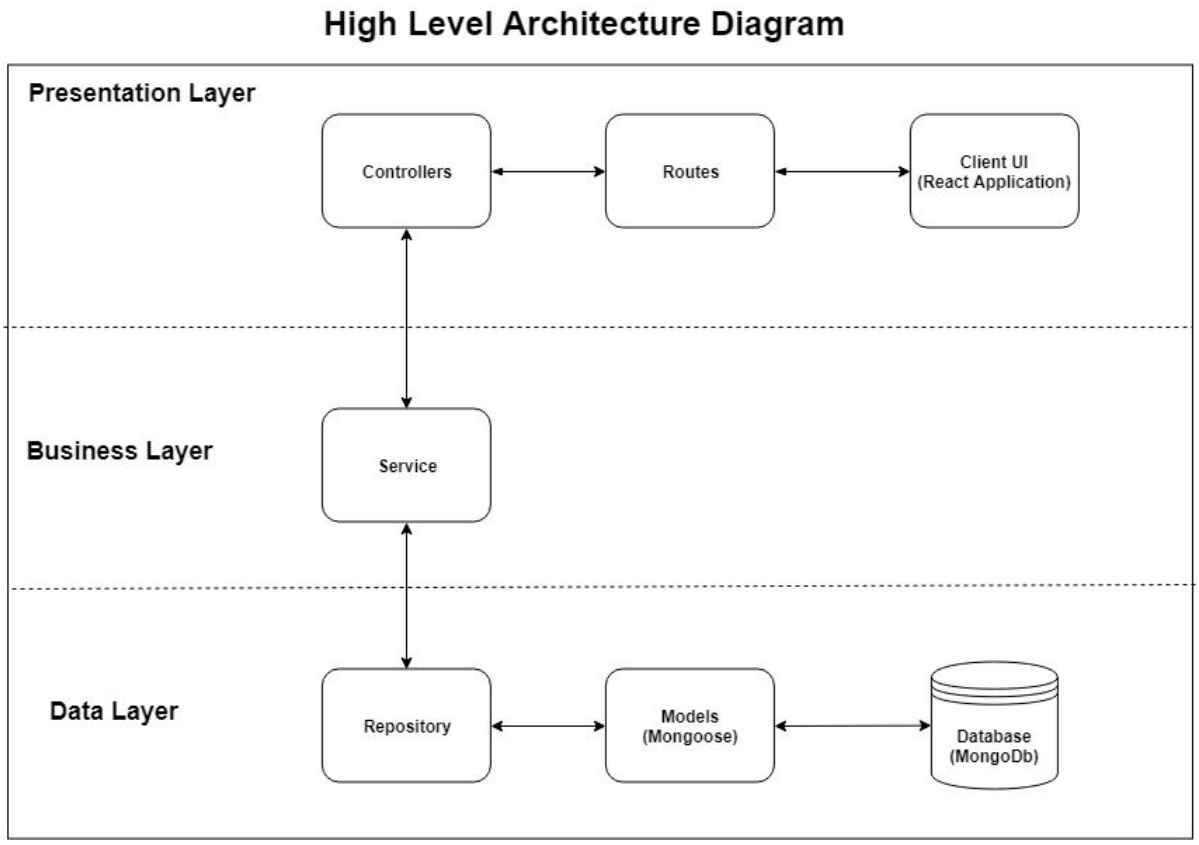
Iteration 10 (6 stories, 25 points) (PROJECTED)

In this sprint we are planning to implement group page functionality. The user will be able to create a group and to join a group. The admin will be able to remove any specific users from any group. We will also create a page where merchants can post external resources which will be paid. We will allow a migrant to edit their privacy settings for a future messaging feature that we will implement. We will also implement the ability to host workshops via teleconferencing.

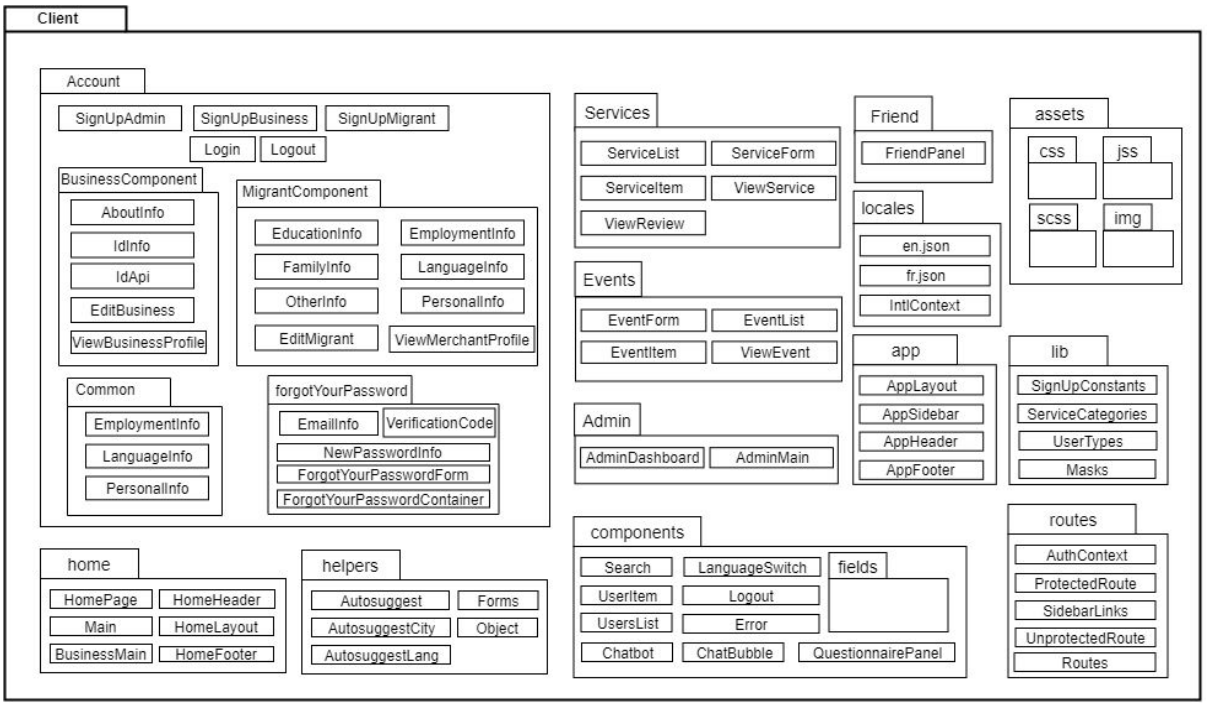
Overall Arch and Class diagram



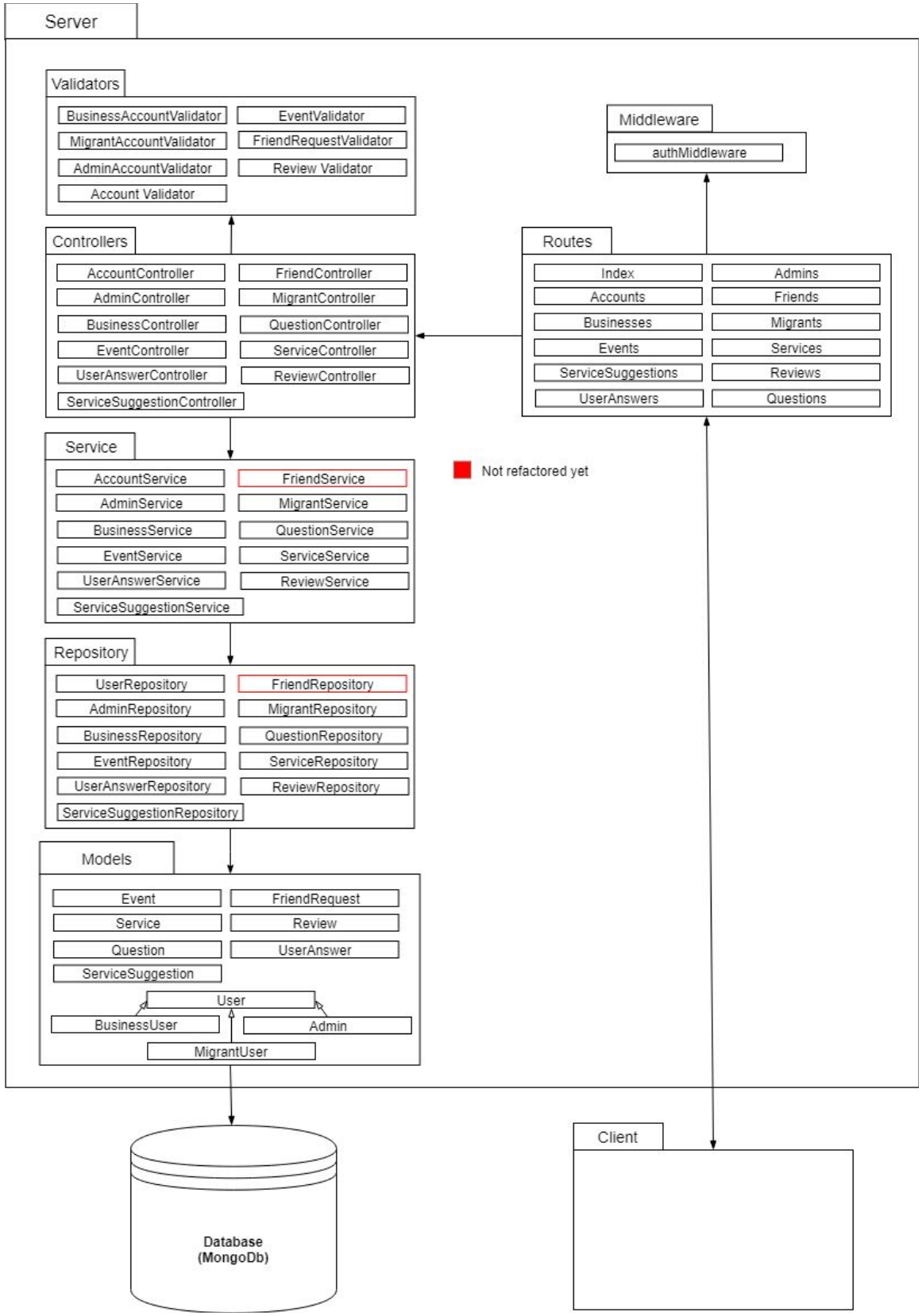
This is the overall architecture diagram showing the current general main components of the application and the links between them.



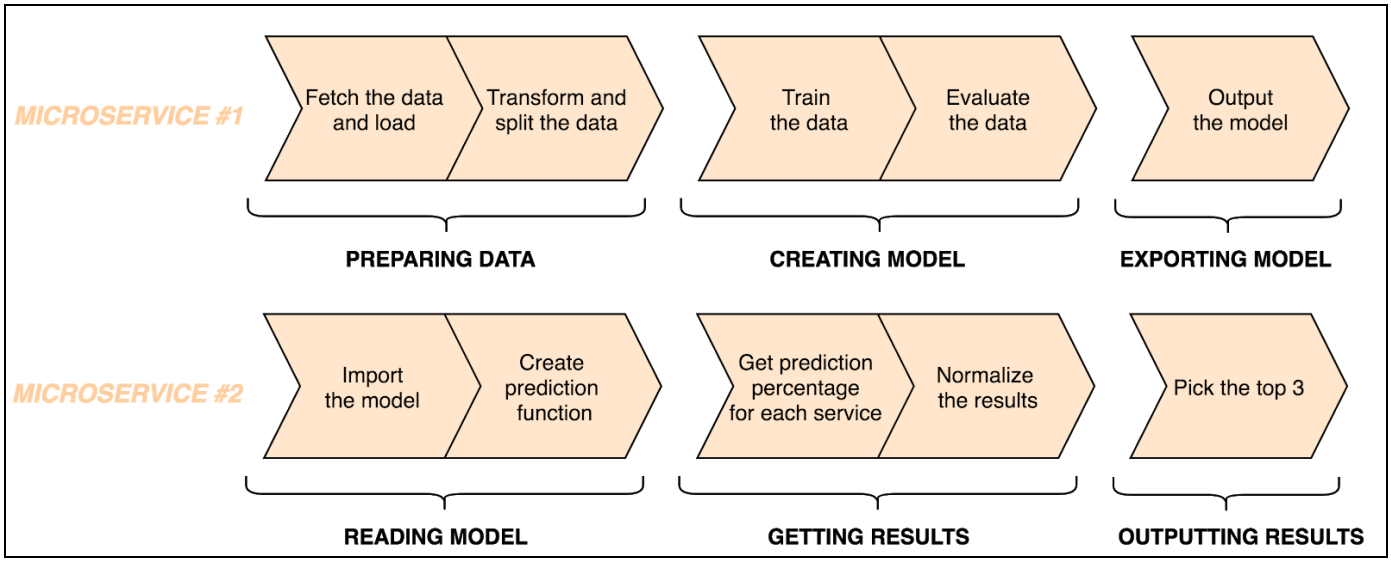
This is the high level architecture diagram.



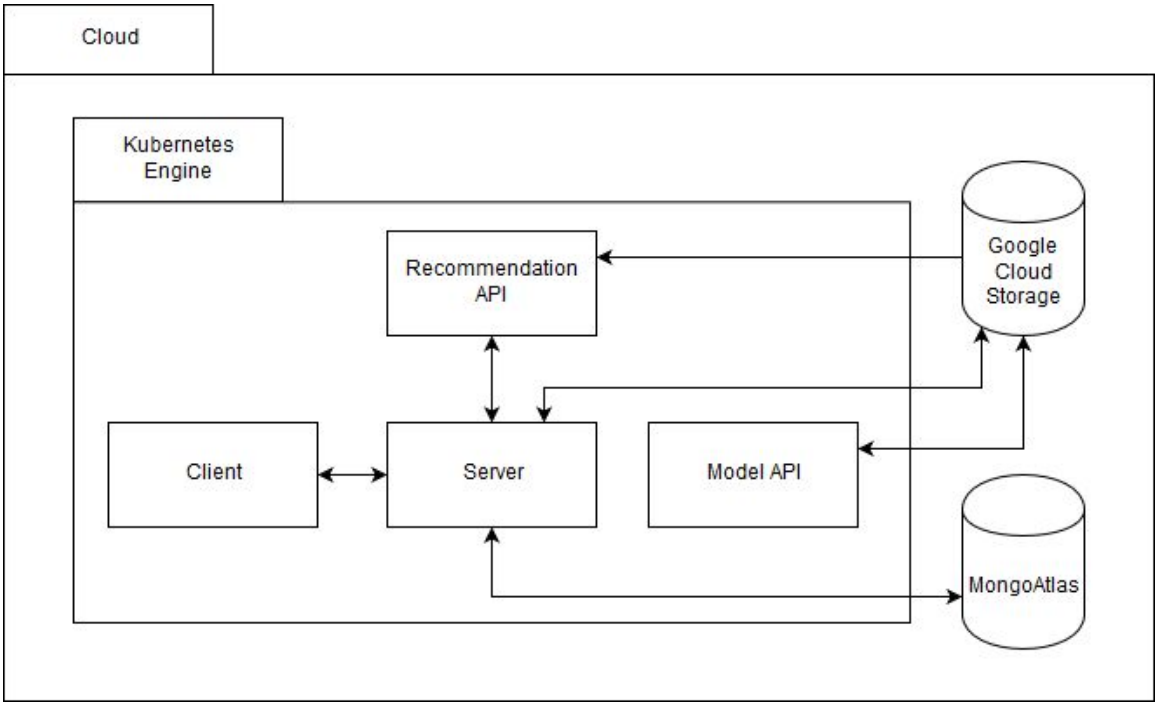
This is the overall client side class diagram.



This is the overall architecture and package diagram for server.



These are the steps of the two ML APIs.



This is the overall cloud setup for the live application. Kubernetes holds the Machine Learning APIs, the client service and server service.

Infrastructure

Hosting

The application is hosted using Google Cloud and their Kubernetes Engine which allows us to deploy our app as a docker container into the cloud without too much pre-configuration. Since this step was expected to be straightforward and simple, we originally included it as one of many tasks for the issue to implement the first data model (#91). Unfortunately, it seems that this was a task that we heavily underestimated, since we under-pokered it and only expected it to take a few hours. Throughout the sprint, we realized that this task was actually beginning to be much more difficult than it seemed. The issue was in trying to deploy the app as a whole, by pulling directly from our repository and using a joint dockerfile for both the server and client. Most of the time, though we were exposing two ports, only one (if any) was actually being exposed and was accessible to the public web. It took much more than the 8 points we originally thought it would. After having realized how much longer it was taking, we had a group meeting and agreed to reduce the scope of this issue. The main task had become to simply deploy and host the application for this iteration. We moved the data model implementation to the next sprint.

Frontend

Frameworks

DialogFlow

This is the framework that is used to create, train and update the Chatbot. On the service's console, we create intents, give them training phrases and responses and let the service take care of the training.

ML.net

This is the framework that we are using to build our ML engine. We used the framework to set up, train, and evaluate models, as well as to apply these models with a prediction function to get service recommendations (more information below, under ***Recommendation Engine***)

React

Since our application is a web application, we will be using HTML, CSS and JavaScript. To make development faster and to build a good UI, we are deciding to use the React framework. This will allow us to encapsulate our application into components that we can use in many places, eliminated code duplication. This framework will also allow us to keep the DOM separate from our components logic and allow us to understand our code easier.

- Testing Tools
    - Jest: a javascript test runner that allows us to run javascript tests for react. Also generates coverage reports for the client.
    - Enzyme: allows us to mount single react components for unit testing and conduct unit tests on the user interface
    - faker.js: used to fake some data in order to populate database or generate test data.
- React.js Tools
    - react
    - react-autosuggest
    - react-dom
    - react-input-mask
    - react-number-format
    - react-router
    - react-router-dom
    - react-scripts
    - react-text-mask
    - react-transition-group
    - react-tables
    - React-datetime
- Node.js Tools
    - Dotenv
    - cross-env: specify src as start of paths
- UI Tools
    - Material-UI: The main framework for UI design.
    - MDBootstrap: Secondary framework for UI design.
    - animated
    - autosuggest-highlight
    - lodash
    - react-google-maps
    - recompose
    - @material-ui/core
    - @material-ui/icons
    - Classnames: join classNames together
    - font-awesome
    - Form-data

    - Material-dashboard-react
    - Material-ui-rating
    - Mdbreact
    - Moment
    - Node-sass
    - perfect-scrollbar
- Validation
    - canada: library of canadian provinces and cities.
    - country-data: library of languages
    - validator: library of string validators
    - qs
- Server- Client communication
    - axios: used to do http requests
    - http-proxy-middleware
- Login
    - React-facebook-login: authenticate user using facebook
    - React-google-login: authenticate user using google

Server

The back end of the application will be made using Node.js + Express.js.

The Node.js runtime environment will allow us to use JavaScript on our backend so that we can use one language across the stack and make our development easier. Our application will also need real-time functionality for messaging and Node.js will make this convenient with web sockets with a library such as socket.io.

Express.js framework will provide us with many functions that will speed up our development such as routing and middleware functionality. Express will also allow freedom for our choice of backend and project structure that frameworks such as Meteor or Sails would not allow. We will add libraries as we need them since Express has good package support.

- Login security
    - Bcryptjs: salt hash library that is used to store data securely and provide security against brute force attacks.
        - Used to hash a user's password
- Client-Server communication
    - Body-parser: parse incoming request bodies and get data available in the request body.
        - Used to communicate between client and server.

- Client request validation
    - validator: predefined rules for validating different types of strings are provided which save us time from writing our own regex.
- Testing Tools
    - Chai: assertion library used with Mocha to perform unit/integration tests. Provides numerous assertion capabilities to our tests.
    - Mocha: testing engine for express to run unit tests.
    - Sinon: JavaScript test spies, stubs and mocks
    - Sinon-test: Utility tool for Sinon, that allows cleaner tests.
    - Istanbul: Used to generate code coverage reports.
- Helper Libraries
    - Nodemon: allows for continuous restart of the web app when developing the app, this allows for changes to be seen instantly for the express server.
    - Concurrently: allows for us to run two node applications with one command, this allows us to run the express backend and react frontend at the same time.
- Node.js Tools
    - Express: provides features to build Node.js applications.
    - Morgan: HTTP request logger middleware.
    - Cookie-parser: allows usage of cookies
    - Debug: javascript debugger
- Store images on server
    - multer: Middleware to upload files(i.e. Images in our case)
    - fs-extra: Provides file system methods that are used to correctly upload images.
- Logging
    - Winston: logging library
    - App-root-path: specify path for log files
- Login
    - passport-facebook-token: authenticate user using facebook
    - passport-google-token: authenticate user using google
    - Passport-local: local user authentication

Recommendation Engine

There are two significant parts to the recommendation engine (two separate RESTful API services both currently running on GCP):

- *Service_Rating_Model:* after calling its endpoint, this microservice will fetch a list of all the services ratings, prepare it for binary classification, split the data into training and testing data, encode the two features, and then create and export the model of the dataset.
- *Service_Recommender:* after calling its endpoint, this microservice will fetch the previously created model and a list of all the services. Using the ML.Net framework, it will then create a prediction function based on the imported model, and create a list of all the services associated with a normalized percentage of the match with a specific user in question (passed as a parameter when calling the endpoint). It will then return the three top service suggestions to later be used by the system to populate the suggested services section.

Database

Mongodb

MongoDB is a NoSQL database. We will be able to easily store JSON formats of data, that is typically used in a web application. It will also allow us to be more flexible with our schemas since they will be unstable in the beginning and using a relational database would create lots of overhead to change the schema.

- Mongoose: Object modelling tool that allows us to create a object format in mongoose with CRUD functionality.
- MongoAtlas: Mongodb database for the cloud. Used to store data for the cloud version of the application.

Name Conventions

AirBnb JavaScript Style Guide: This style guide supports react.js as well as node.js. In addition, there is a eslint airbnb package that is available with/without react.js support that can be added to the project.

Code

File path with clickable GitHub link	Purpose (1 line description)
MigrantHub/server/service/AccountService.js	All functions related to accounts (create, edit, login, logout) are in this class.
MigrantHub/server/service/ReviewService.js	Contains backend logic for reviewing a service.
MigrantHub/server/service/ServiceService.js	Service for all functions related to services.
MigrantHub/data models/Service Rating Model/Service Recommender/Controllers/ModelController.cs	Controller for the microservice that exports the model (microservice #1)
MigrantHub/data models/Service Recommender/Service Recommender/Controllers/RecommendationController.cs	Controller for the microservice that reads the model and returns recommendations (microservice #2)

Testing and Continuous Integration

Test File path with clickable GitHub link	What is it testing (1 line description)
MigrantHub/server/test/service/ReviewServiceTest.js	That the review service methods are working correctly (add, delete, get, etc.)
MigrantHub/server/test/service/ServiceServiceTest.js	That the services service methods are working correctly
MigrantHub/server/test/service/AdminServiceTest.js	Admin authorization approval and rejection methods
MigrantHub/server/test/service/QuestionServiceTest.js	Testing for correct retrieval of unanswered question from database
MigrantHub/server/service/UserAnswerServiceTest.js	Testing the user answer service (e.g make sure no duplicated answers)

Our continuous integration environment is setup using TravisCI. The following is a link to our project: MigrantHub.

We have set this up to use the latest LTS version of node and npm, this then builds our project, in the client folder and in the server folder. After the builds are complete the test suites are run in the client folder and in the server folder. The client folder runs tests using Jest which conveniently is also is able to provide a coverage report. The server folder uses mocha to run tests and istanbul to create code coverage reports. The code coverage reports are sent to codecov after each build to help us track the progress and difference. We have also added a linter to our application using ESlint, we fail builds that do not pass the linting for the server. This continuous integration pipeline is set to run after each commit as well as on each branch pull request. It is also run on master every time something is merged.