MigrantHub

Grade for Release 1
Grade for Release 2

Release 3: Slides

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	50	Backend, Frontend, Cloud, ML
Alexandre Masmoudi (27755473) *a-masmoudi@hotmail.com*	Smoudii	52 + 21 (491)	Backend, Frontend, Cloud, ML
Laxman Velauthapillai (40000111) *laxman_20@hotmail.com*	iamlax	56	Backend, Frontend, DevOps, Cloud
Tusman Akhter (40003476) *tusman96@hotmail.com*	tusmanakhter	61	Backend, Frontend, DevOps
Tajbid Choudhury (40002177) *tajbidhussain@gmail.com*	CodeTaj	44	Backend, Frontend
Mazen Nahle (27003315) *nahlemazen@gmail.com*	mazennahle	47	Backend, Frontend
Rajeevan Vairamuthu (40000112) *rvairam10@gmail.com*	rajee10	43	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.

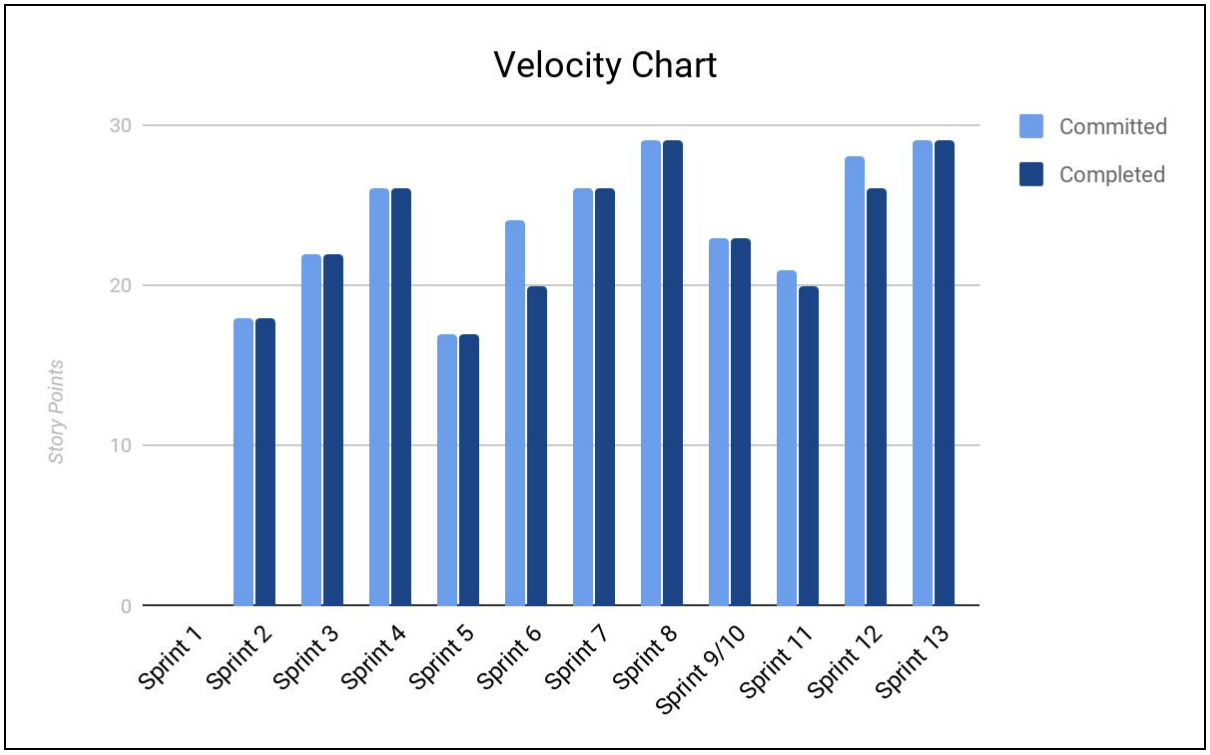
In order to mitigate these issues the site will need moderators or an approval method when users want to post services. The application will use security standards in order to prevent hacking.

Hardware/cloud/compute requirements

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

Velocity

Project Total: 104 stories, 274 points over 29 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 (10 stories, 23 points)

During this sprint, we had a much-needed bug bash. After sitting down with multiple users and recording their live feedback when using our app for the first time, we were able to compile a list of bugs or things that should be changed, all ranging from the user’s login/signup process to the profile to events and services and better display of errors/alerts. We also worked on improving the chatbot and building on the bare foundation that we had set up in previous sprints.

Iteration 10

This sprint ended up being joined with Iteration 9.

Iteration 11 (13 stories, 20 points)

During this sprint we improved the login and signup user experience based on feedback. We also changed and applied a new color scheme throughout the website. We also continued with the bug bash by sitting with more users and recording their feedback to ensure that our system is really ready for live release. Much of our time was spent sitting with them and getting acquainted with what they expected from the system and what they didn’t like about it. We also manually went through many of the services on our system to find and post descriptions for the migrants to see in the service cards. This will be continued next sprint. We also continued working on the chatbot, improving the user experience of it.

Iteration 12 (12 stories, 26 points)

In this sprint, we performed some user experience/feedback requests to show that we're processing the users requests visually through a load animation. Also we added terms and condition, improved the side navigation bar, added a dashboard where users can pin their most used services, added infinite loader to lists, client check for emails being taken, improved edit profile and added additional service descriptions(Part 2). We also added a feature of job postings that users can view and businesses can post and a page for admins to view bug reports. On the ML side, we ensured that the recommendation models are updated automatically on a regular schedule (as opposed to manually). Also, the chatbot is now able to fully help users with various scenarios.

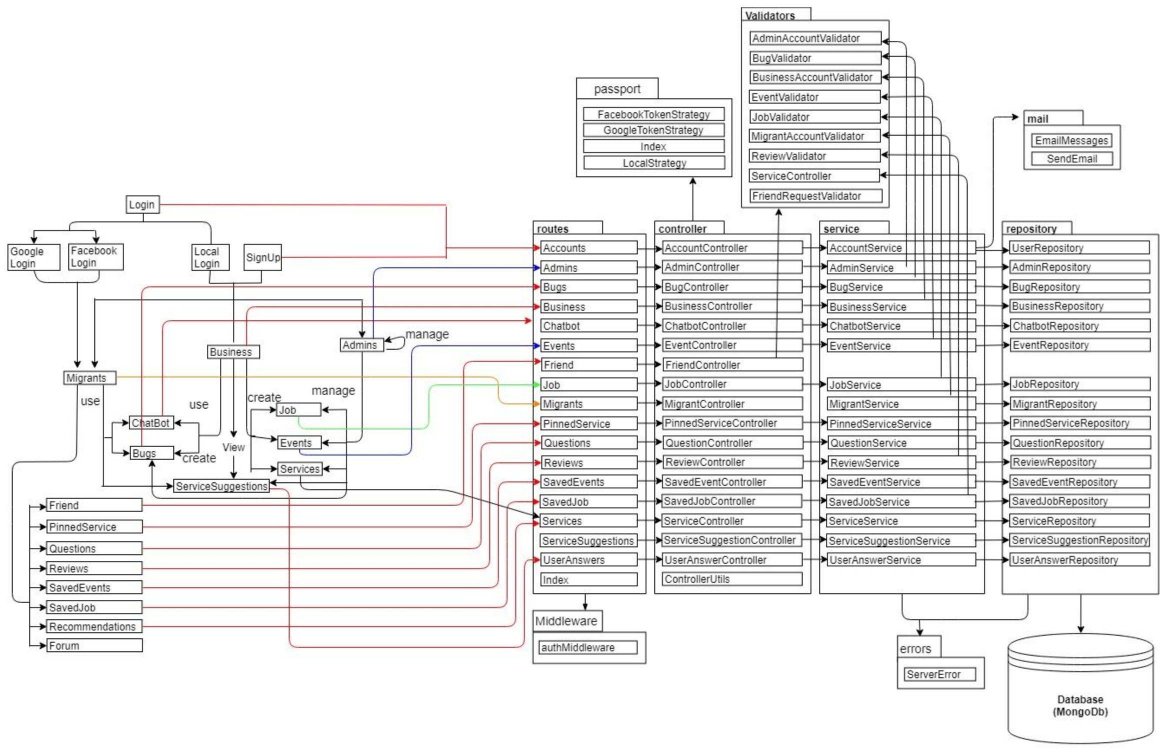
Iteration 13 (11 stories, 29 points)

In this sprint, we performed some user experience/feedback requests to the event cards. We added time created to service and event cards and added a checkbox to order them by newest first. We added a guided tour of the website upon first sign in. We improved service details displayed. We also added the ability to save events to dashboard. Admins are now able to resolve bugs. Users can now search jobs on the website. Also we added a forum for any questions users have. The recommended services now factors in age to the recommendations given. Furthermore, we polished the chatbot and improved its conversation tree. Lastly, we configured our backend server to send logs to Loggly's log management tool during production and added the necessary documentation in the GitHub Wiki.

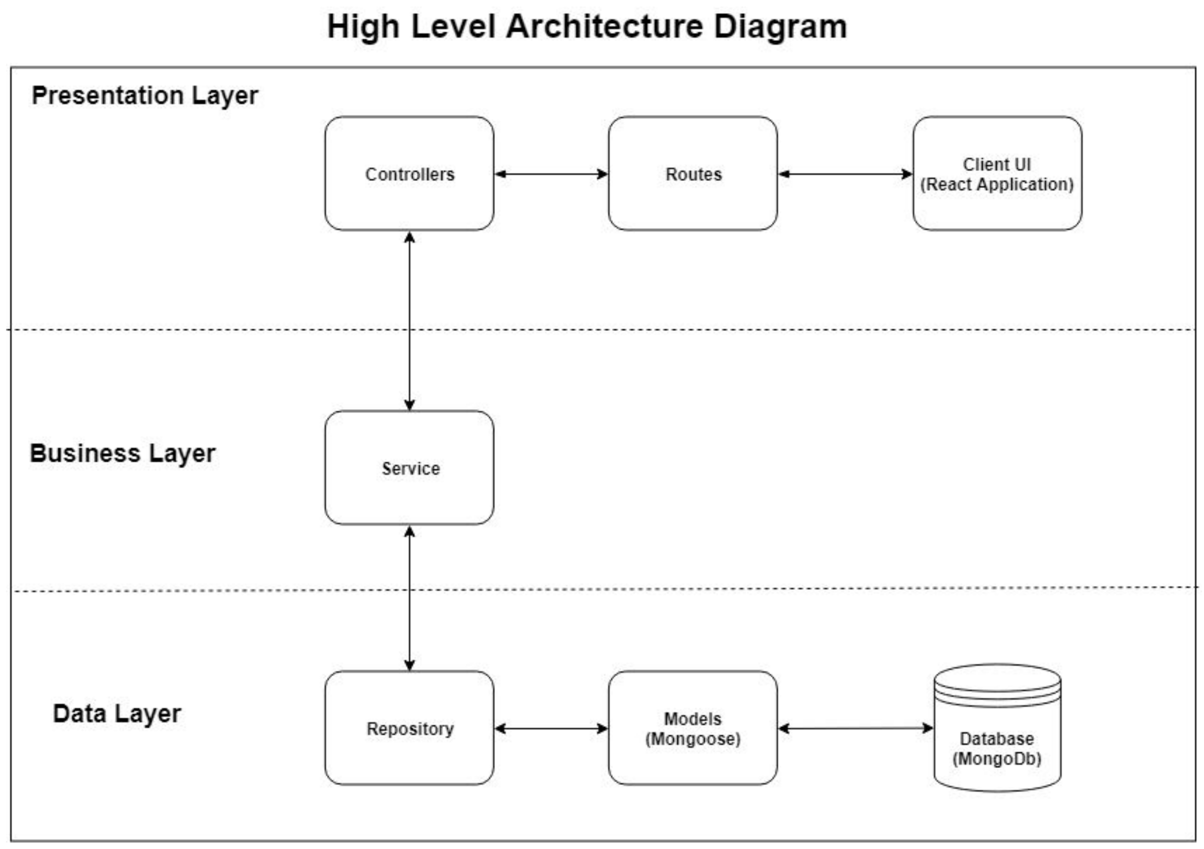
Documentation:

- Readme
- Wiki

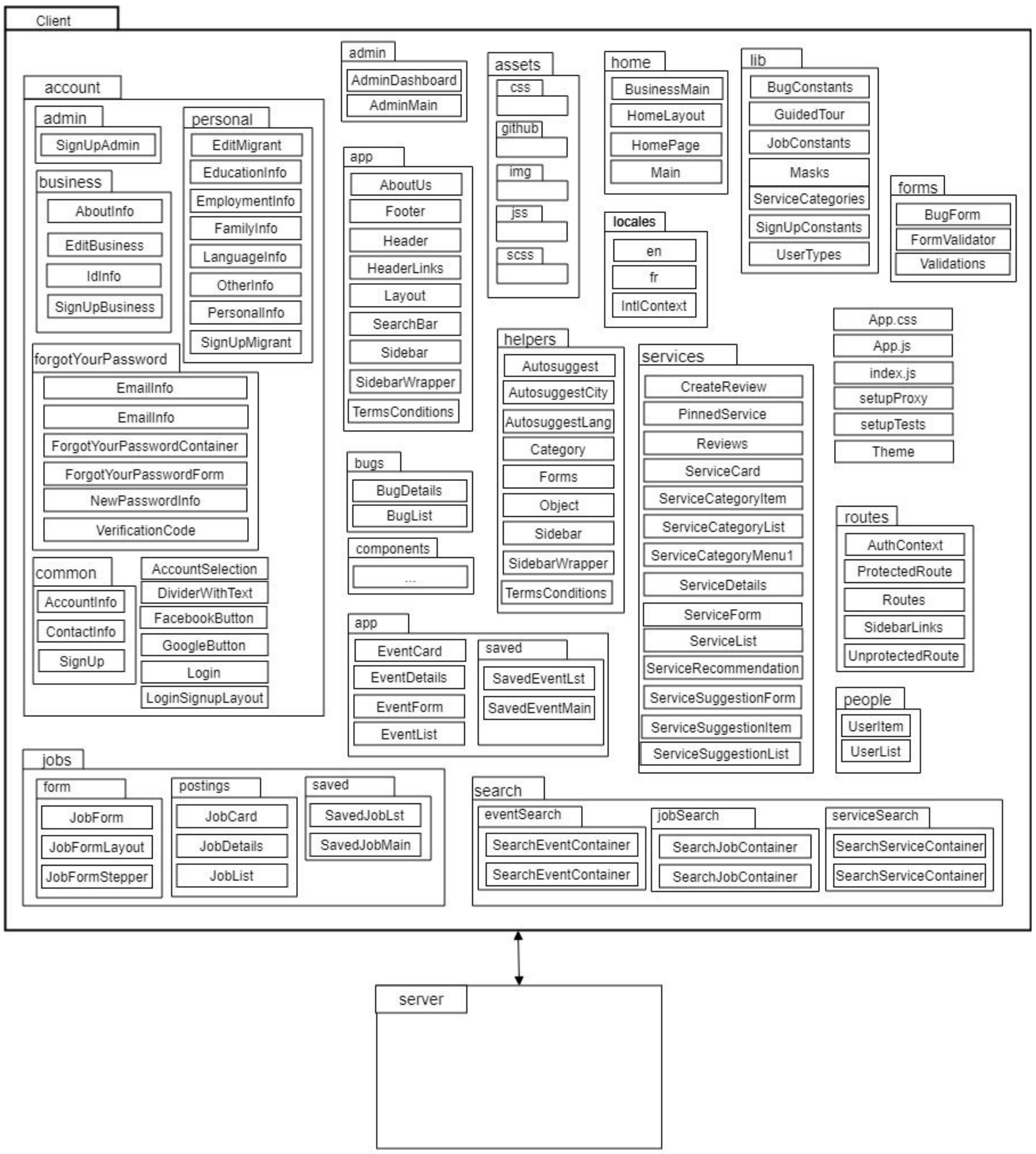
Overall Arch and Class diagram



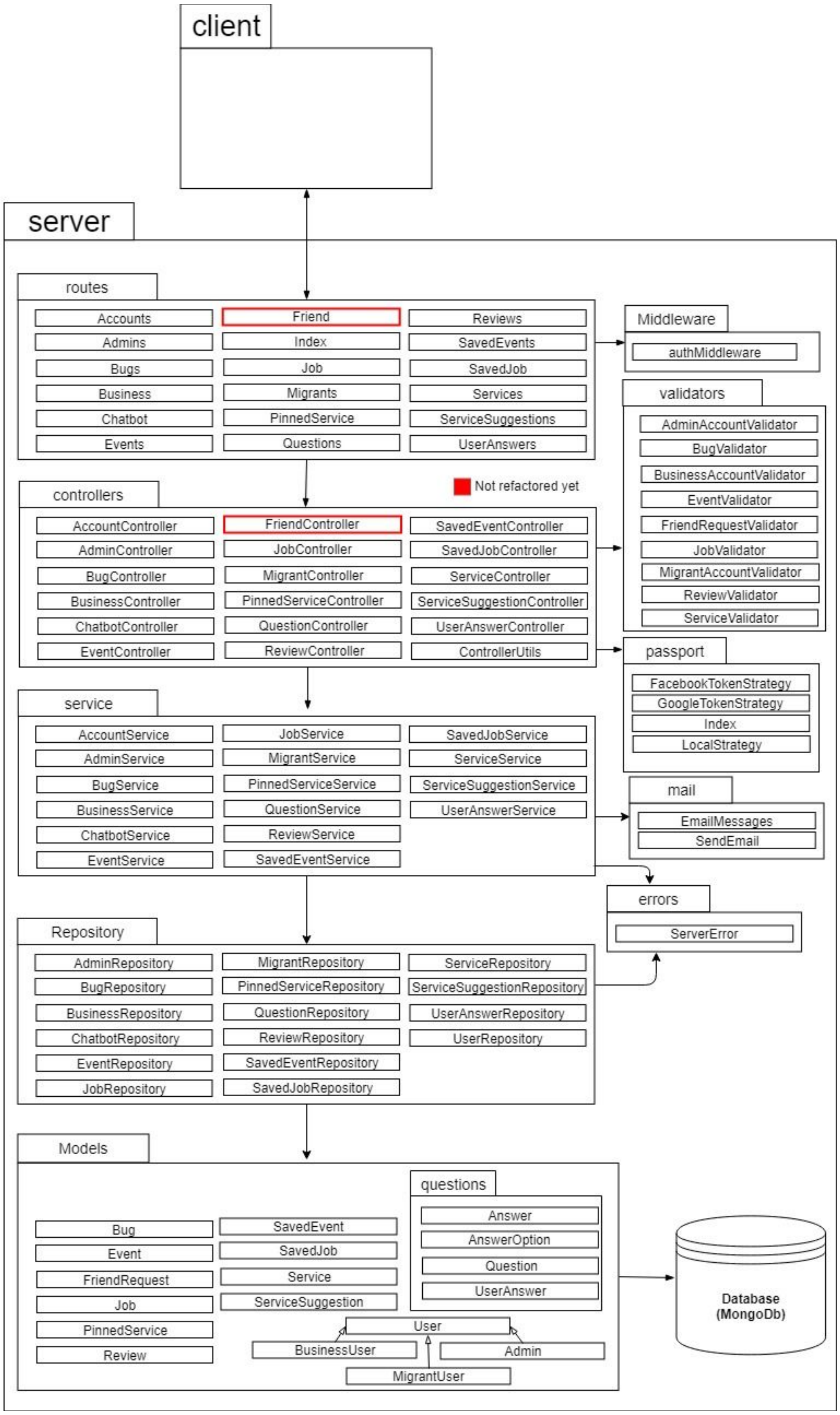
This is the overall architecture diagram showing the current general main components of the application and the links between them.



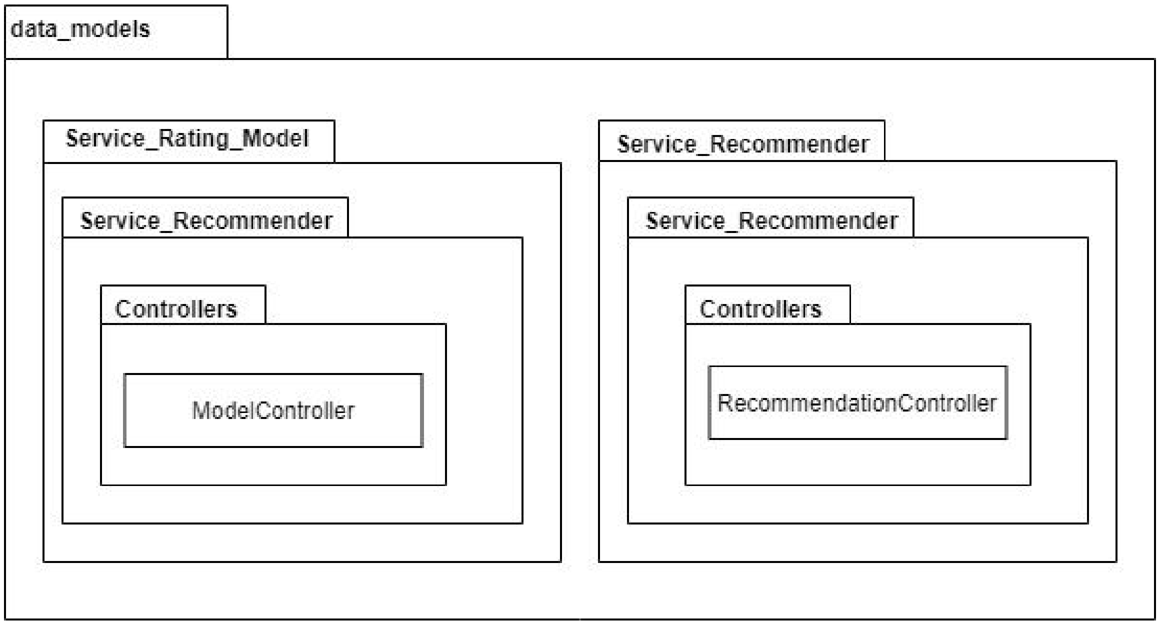
This is the high level architecture diagram.



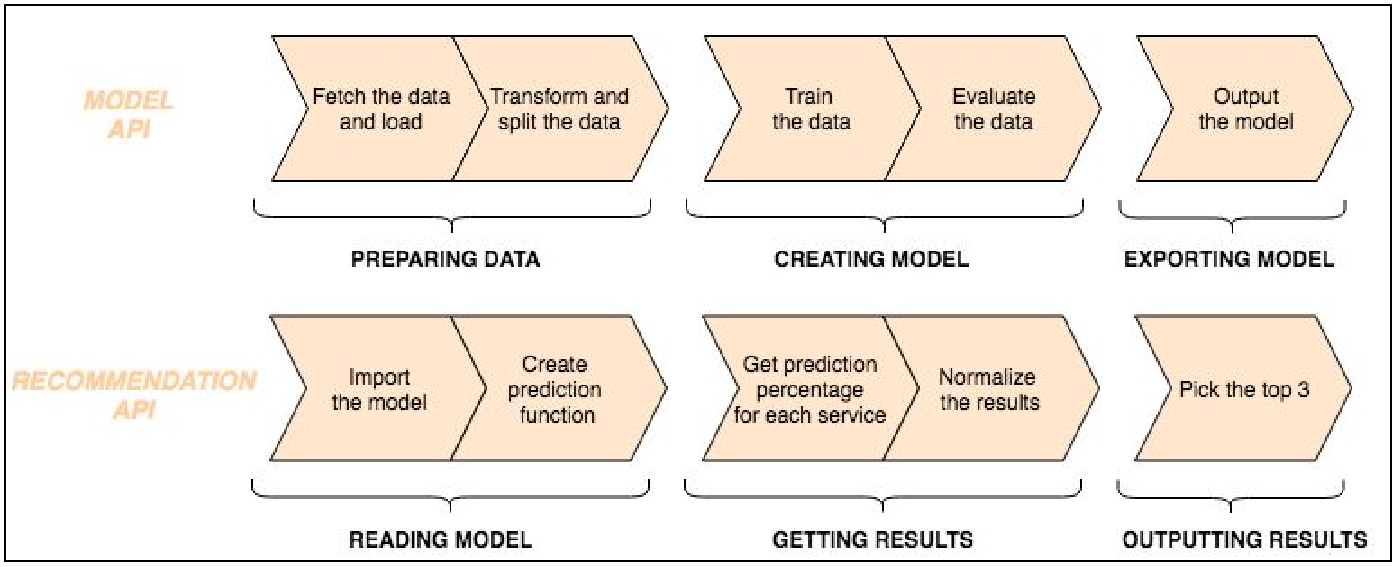
This is the overall client side class diagram.



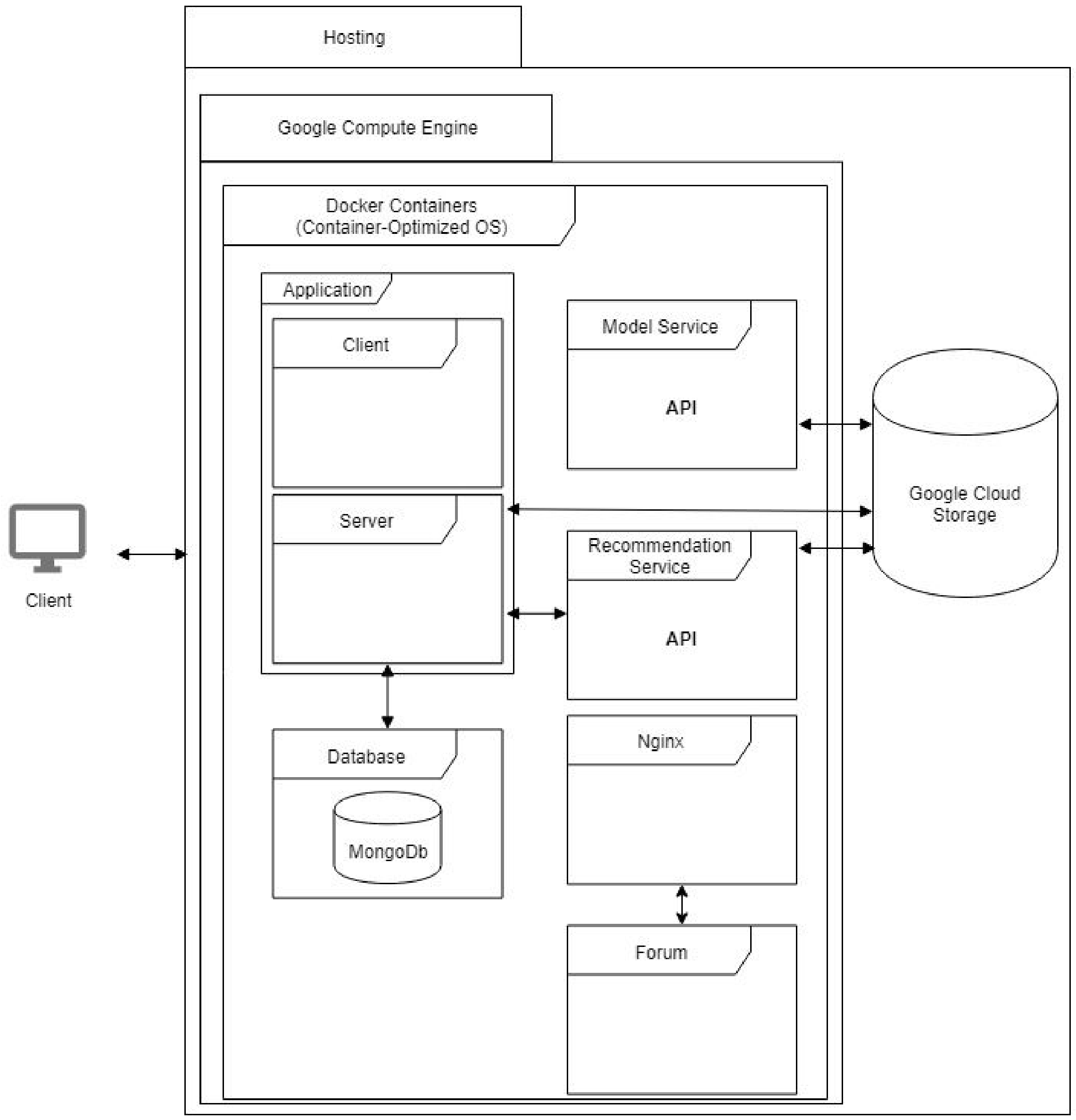
This is the overall architecture and package diagram for server.



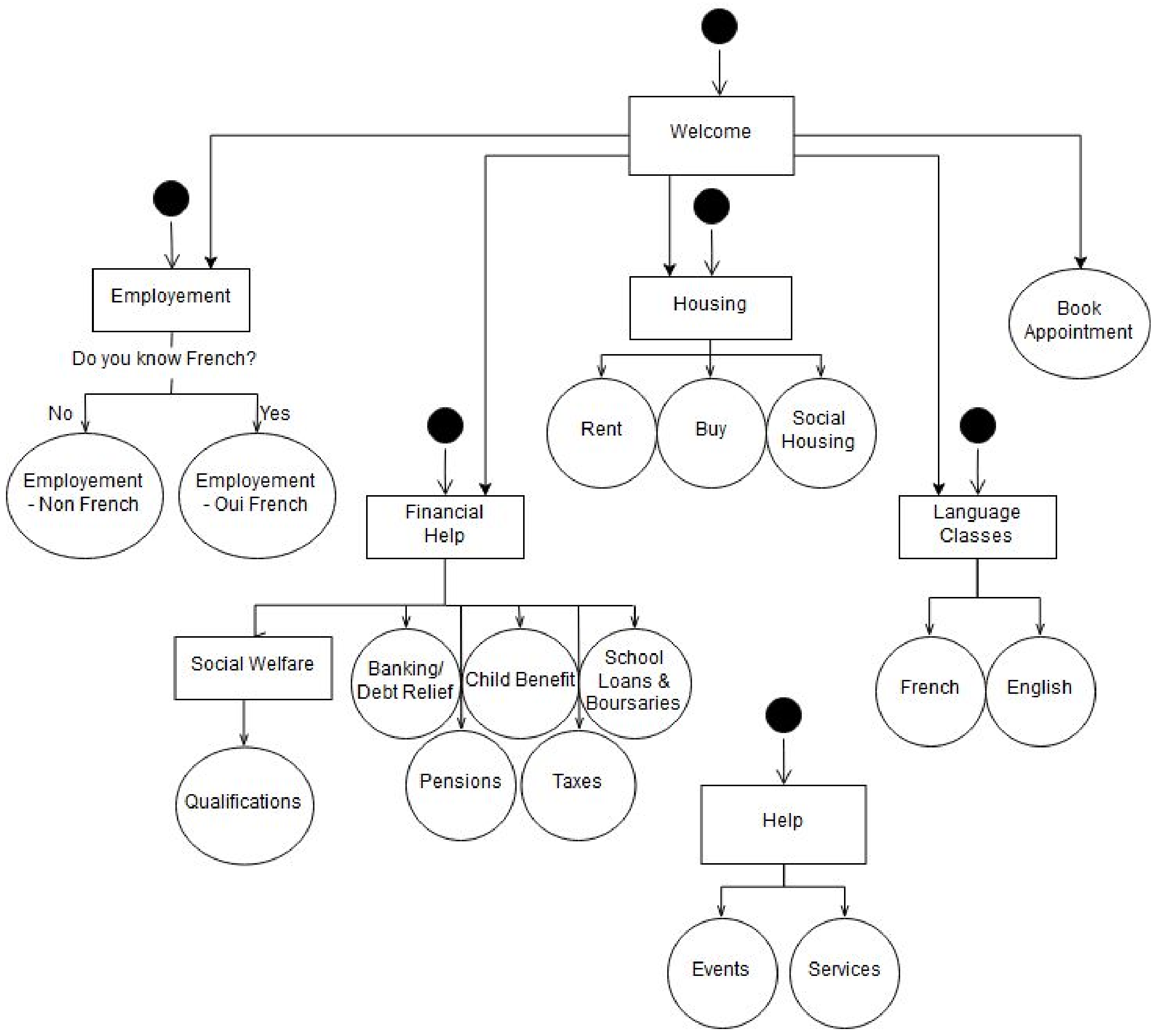
This is the overall architecture and package diagram for data_models.



These are the steps of the two ML APIs.



This is the overall cloud setup for the live application. The application is hosted on the Google Compute Engine using docker-compose to containerize the Machine Learning APIs, the application and database.



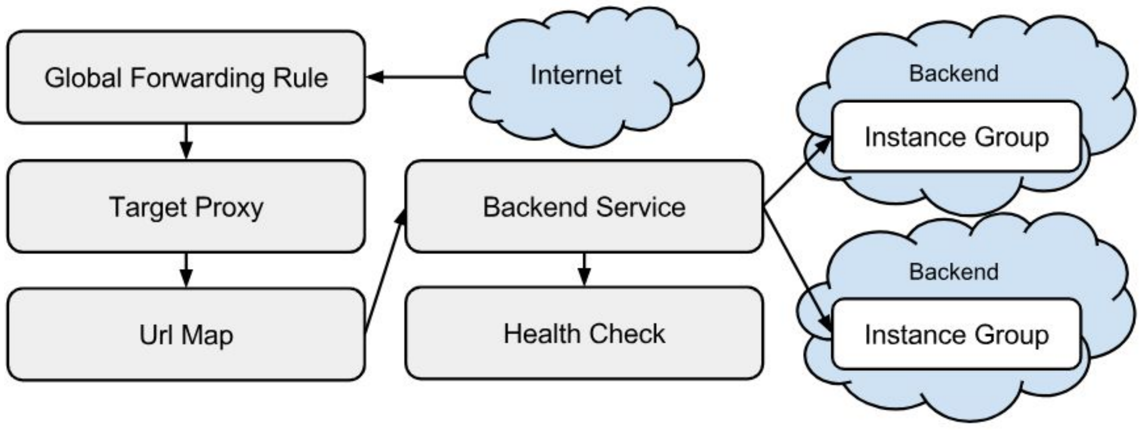
This is the Chatbot conversation tree implemented with DialogFlow API.

Infrastructure

Hosting

The application is hosted using Google Compute Engine. Using a containered OS we are able to run the docker-compose-production.yml file to build the application. This allows the application, database, service rating model and service recommender microservices to be separate containers in one Google Compute Engine Instance we created.

The application is setup up using Google Cloud Platform's load balancer setup. The load balancer is able to redirect the users from a target proxy to one of our backend instance groups. This can be visualized from the image below from Google's documentation. Currently, to lower costs we have only setup 1 backend instance that the users can access. However, it can be configured to dynamically/manually create any number of backend instances. This will allow the application to support larger amount of users in the future.



Cross-region load balancing diagram

Image: https://cloud.google.com/load-balancing/docs/https/

The forum is not part of the docker-compose yaml files that we have since it requires its own deployment method. We have used discourse and they suggest not to deploy using a docker-compose file. Thus we deploy the forum separately before or after running the other containers through docker-compose.

An nginx reverse proxy is used to properly route the traffic of the forum subdomain running on the default http ports to the proper application and the rest of the traffic to the server. This is also running in a docker container on the Google Compute Engine.

Docker
Application is containerized using docker.
Two docker-compose files:
- Development: docker-compose.yml
- Production: docker-compose-production.yml

Application is built on the cloud using the docker-compose-production.yml:

Logging
- Server-Side Logging using Winston package
- Information we are logging:
  - Database failures, Authorization failures, Application errors and system events, Code location e.g. script name, module name, etc..
- Format:
  - ip method url httpVersion status referrer customMsg errorMsg
- More Info on logging: Link
- Production logs are sent to loggly
  - Loggly is a log management tool, where the logs can be analyzed

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 recommendation 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.

Purpose	Package	Description
System	cross-env	Run scripts that set and use environment variables.
	dotenv	Load environment variable from .env file
	http-proxy-middleware	Configure proxy middleware.
	immutability-helper	Helper to manipulate data.
	react-dom	React server rendering
	react-cookie	JavaScript cookies
	react-router	React Routing
	react-router-dom	React router binding helper
	react-scripts	Required scripts for react application
	lodash	Utility to work with arrays, numbers, objects, etc.
	react	Library for creating user interfaces
	react-app-polyfill	Create react app helper for browser support.
HTTP Requests	axios	used to do http requests

Login	react-facebook-login	Facebook login component for authentication.
Forms/UI	react-google-login	Google login component for authentication
	canada	Get cities and provinces from Canada
	classnames	Join class names together helper
	country-data	Country Data helper
	@material-ui/core	Components that use Google Material Design
	@material-ui/icons	Google Material Design icons
	animated	Animations Library
	autosuggest-highlight	Helper to highlight test in autosuggest
	draft-js	Help build text editors
	font-awesome	Font css
	form-data	Create form multipart data to sender to server. Used to send image with form data.
	material-dashboard-react	Material UI design components
	material-ui-rating	Rating component.
	moment	Time and date component.
	node-sass	Stylesheet helper library.
	prop-types	Type checking for react props and objects
	qs	Parser.
	rc-menu	Menu component.

	react-add-to-calendar	Add to calendar button used in events & services.
	react-autosuggest	Auto-suggest component used in forms
	react-bootstrap-sweetalert	Alert used to confirm user selection
	react-chat-widget	Widget used for chatbot
	react-datetime	Date time picker
	react-draft-wysiwyg	A customization description box
	react-google-maps	Google maps component
	react-icons	Icon pack
	react-infinite-scroller	Allow pages to scroll to load data.
	react-input-mask	Input masking component
	react-intl	Allow language translation support.
	react-number-format	Format number input
	react-rating	Rating component
	react-star-rating-component	Rating component
	react-table	Table component
	react-text-mask	Format pattern input
	react-toastify	Send toast alerts to notify user.
	react-transition-group	Manage animations
	reactour	Create a tour of the application components.
	recompose	higher-order components utility
	styled-components	Styling helper

Validation	validator	library of string validators
Tests	react-test-renderer	Snapshot testing
	enzyme	Testing utilities
	enzyme-adapter-react-16	Testing utilities
Eslint	eslint	JS code pattern check
	eslint-config-airbnb	Airbnb eslint styleguide
	eslint-plugin-import	Lint syntax support
	eslint-plugin-jsx-a11y	Eslint static helper
	eslint-plugin-react	React linting rules

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.

Purpose	**Package**	**Description**
System	express	Framework
	express-session	Session middleware

	dotenv	Load environment variable from .env file
	cookie-parser	Parse HTTP requests
	debug	Debugging utility
	concurrently	Run command concurrently
	body-parser	Parsing Middleware
	app-root-path	Access application root path
	morgan	HTTP request logging middleware
Database	mongoose	MongoDB object modelling
	mongoose-auto-increment	Increment mongoose schema
Logging	winston	Logging Library
	winston-loggly-bulk	Winston Transport to send Winston logs to Loggly.
Login	passport	Authentication middleware
	passport-facebook-token	authenticate user using Facebook
	passport-google-token	authenticate user using Google
	passport-local	authenticate user
	bcryptjs	salt hash library that is used to store data securely and provide security against brute force attacks.
Validation	express-validator	predefined rules for validating different types of strings are provided which save us time from writing our own regex.
	validator	library of string validators

Storage	@google-cloud/storage	Cloud storage client library
Chatbot	dialogflow	API client library
HTTP Requests	axios	used to do http requests
Send Email	nodemailer	Send email
Store Images	fs-extra	File system methods
	multer	Middleware to handle file uploads
Helpers	qs	String parsing library
	string-similarity	String comparison tool
	faker	Generate fake data
	nodemon	Automatic restart during file changes
Test	chai	assertion library used with Mocha to perform unit/integration tests. Provides numerous assertion capabilities to our tests.
	chai-as-promised	Assertions with promists
	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.
Lint	eslint	JS code pattern check
	eslint-config-airbnb-base	Airbnb eslint rules
	eslint-plugin-import	Lint syntax support
Code Coverage	istanbul	Code coverage tool

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.

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	
MigrantHub/server/service/PinnedServiceService.js	All functions related to pinned services to dashboard (add, update, remove) are in this class.
MigrantHub/server/service/JobService.js	Contains backend logic for jobs 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/JobServiceTest.js	Test the job posts CRUD operations and retrieving jobs.
MigrantHub/server/test/service/SavedJobServiceTest.js	That the saved job methods are correctly working (get, add, remove)
MigrantHub/server/test/service/SavedEventServiceTest.js	That the saved event methods are correctly working (get, add, remove)

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.