Technological Feasibility Analysis

October 27th 2016

Five Pixels

Sponsor:
Joy Knudsen

Faculty Mentor:
Dr. Mohamed Elwakil

Team Members:
Brandon Garling
Xiangzhi Cao
Matthew Nielsen
Mohammad Alsobhi
Clarissa Calderon

Rev 1.0

Introduction

Joy Knudsen, currently operates a non-profit organization called Flagfriends. This project connects international students with hosts so that they can develop friendships and learn from one another. Her current method to achieve this is to have applicants fill paper applications and then manually sorting the applications manually, matching hosts and students to her best abilities. Although this method currently achieves the goals of the Flagfriends project, this approach is extremely labor intensive and does not leave much time for outreach to grow the project further. 5 Pixels is a senior capstone project team who is teaming with the Flagfriends project to develop a web application to better facilitate the international student and host application and matching process. This will give Joy more time to focus on expansion. This web application will have three types of users: hosts, students, and administrators. The website will allow hosts to match with students based on a list of attributes and questions that each user must answer upon registration. To make the application suitable for more organizations we decided to make the application customizable and configurable so that another organization can easily install the application, and configure it to suit their needs.

The purpose of this document is to explain the technical feasibilities, analyze the technological challenges and provide solutions based upon currently available technologies.

Technological Challenges

The project brings up unique technological challenges. Starting from the top we will need a way to easily generate a responsive frontend interface. A responsive interface ensures that we users will enjoy using the web application and return to it in the future. We also need a way to provide a RESTful API for the frontend to communicate with that also handles all our backend logic. Having a backend that runs without being invoked by a page request is essential, we will need to support notifications and other time based events even if no user is currently using the frontend of the application. We will want a secure way to store user data for our service in a database. When the application is ready for testing and deployment we will need a place to host it in the cloud. By prepping the site prior to deployment, we can perform load testing and user testing to ensure that the application is performing optimally. We need to allow users to upload images meaning we will need a way to store images on the server and serve them back to the client. Email will help support notifications to users who are not actively using the application. We will need a way to secure the application and encrypt data going between the client and server as well as provide a secure form of authentication. Instant communication while a user has the application open allows two or more users to have a conversation in real time. Initial setup of the application will need to be completed at an initial application startup so that a user can easily configure the application to suit their needs. Administrators will also need a way to customize the application after deployment and change page contents easily and intuitively.

Reports will need to be generated and displayed for the administrator based on user data in the database.

Technology Analysis

This section will analyze each of the preceding issues in more detail and follow each one with alternatives, our chosen approach, and how we plan to prove feasibility of our chosen approach. Proof of feasibility will come with a feasibility demo at the conclusion of this semester. Some challenges already have proven solutions in a demo form, we will also present these at that time. The tables included in the chosen approach section of each challenge represents each alternative and our overall researched evaluation in terms of usability, functionality, and documentation. Usability represents our ability to easily pick up the technology based on our current skills and work with it. Functionality represents how well the solution fits based on our requirements. Documentation represents the ease of finding documentation, examples, and help from communities for the solution. Each rating is between 0 and 5, with 5 being best and 1 being worst in that category.

Create a Responsive Frontend Interface

Our client wants a Web 2.0 application that is responsive and easy to use. Using a single page application (SPA) allows us to achieve these requirements. One big hurdle is selecting a frontend framework (if any) to use to provide this responsive and easy to use interface.

- **Alternatives**
  The main alternatives we considered include: plain HTML with Javascript, AngularJS, AngularJS 2, and ReactJS.

  Using plain HTML with Javascript would be unfeasible, it would add unnecessary weeks to the project to create a single page responsive web application.

  After researching AngularJS, we found that using the original AngularJS was difficult and provided a large learning curve for more complex logic. AngularJS is also slow at times for large complex applications. However, it was a step in the right direction and lead us to consider AngularJS 2.

  AngularJS 2 is a complete rewrite of AngularJS to compete with ReactJS in terms of speed. It leverages a language called TypeScript which is a strongly typed derivative of Javascript. AngularJS 2 provides the whole frontend Model View Controller (MVC) architecture. It is a new library, but has a large user base with many resources available to get help and find answers for the different pieces of the framework. AngularJS 2 also comes with many testing tools such as Karma for unit testing, and Protractor for end-to-end testing. AngularJS 2 provides an internal router that can be used out of the box. It also provides HTTP REST request functionality out of the box without using a third-party library like jQuery.

ReactJS is like AngularJS but only provides the View portion of the MVC architecture. This makes it complicated to get up a fully functional MVC frontend up and running without using various other libraries and frameworks to provide the Model and Controller pieces of the architecture. Using ReactJS provides less of a learning curve, however it is difficult to set up correctly and effectively. To implement routing in ReactJS we must download another library that works with React. HTTP REST requests need to go through some sort of third party library like jQuery.

- **Chosen Approach**
  We ended up going with AngularJS 2 after much research and debate amongst the team. We went this route because it supports everything we need out of the box without the need to change much and allows us to get started almost instantly thanks for helper services such as angular-cli that helps in generating an initial project structure for follow for AngularJS 2 projects. Although it was close, the table below shows our discussion and ratings of each category we looked at for our choice.

Table 1: Frontend Frameworks

Platform/Attribute	Ease of Use	Functionality	Documentation
AngularJS	3	4	4
AngularJS 2	5	5	3.5
ReactJS	3	2	4
HTML/JS	1	1	1

AngularJS 2 is all around easier to learn and use in comparison with the other frontend frameworks. Functionality wise AngularJS 2 provides all the functionality we need in one package rather than needing to figure out which additional packages we want to try to get to work with ReactJS. It also provides more functionality than AngularJS 1. The documentation available for each framework is nearly consistent across all frameworks. AngularJS 2 is newer so it has a little less overall documentation and tutorials than other frameworks.

- **Proving Feasibility**
  We will create a simple webpage that uses AngularJS 2 and shows using components effectively and routing using the controller. It will also provide a simple model that can be manipulated.

Provide an API for Frontend to Backend Communication

For AngularJS 2 to work effectively in a single page application manor we need to provide a way for it to retrieve and manipulate the state of a backend server and database. The best way to do this using AngularJS 2 is through a RESTful API. This will serve as the actual backend of our web application.

- **Alternatives**
  Many options are available to host the backend logic of our application. Two most prominent we are considering are PHP and NodeJS as most of us have experience with one or the other.

  PHP is a server side scripting language that is supported by most web servers. We would need to expose REST APIs through PHP which would be a bit difficult to manage as our list of REST APIs expanded due to each REST endpoint needing to be a separate page. A big drawback of PHP is its lack of ability to run without being invoked. If we wanted to have a script that ran to check if any notifications needed to be sent out every minute, we would have to schedule a CRON job on the server to execute a custom script to do it for us.

  Using NodeJS allows us to leverage one of NodeJS's extensively used library, Express. This library makes it easy to create REST API endpoints, process the request, and send back a response. Using NodeJS means we can take advantage of the built in javascript timers in order to handle any events that need to be timed and ran regardless of any external users currently using the system. In addition, NodeJS has a package manager called Node Package Manager (NPM) that allows us to leverage libraries that already exist and install them easily.

- **Chosen Approach**
  We decided to go with NodeJS and Express because it is easy to handle notifications and other events that do not rely on user interaction as well as process and return JSON data easily. Table 2 below shows our ratings for NodeJS and PHP after our investigation and internal discussion.

Table 2: Backend Communication

Platform/Attribute	Ease of Use	Functionality	Documentation
NodeJS	3	5	4
PHP	3	3	4

NodeJS is about as easy to use as PHP, however functionally it makes more sense to go with NodeJS due to its ability to run scheduled jobs in the background and the ability to use NPM to help handle dependencies and packages. Documentation wise they are nearly the same, NodeJS is a bit newer but there is no shortage of documentation available online.

- **Proving Feasibility**
  We will create a simple REST API that supports the CRUD operations for some simple kind of model using NodeJS and Express.

Secure Way to Store User Data

We need a way to store user data and program data persistently between application restarts. It needs to be compatible with our chosen web service backend software (NodeJS). Whatever approach we decide on will need the ability to dynamically create tables and columns to the database to add customizability to the application. It also needs to support storing blobs of data such as images.

- **Alternatives**
  There are many NodeJS compatible database services, the two big ones are MongoDB and MariaDB.

  MongoDB is a NoSQL style database meaning it is a non-relational database allowing you to store “documents” that are JSON like in structure easily and quickly. This is the most used database when working with NodeJS. It is nice because it requires no separate setup, however it is slower than the MariaDB alternative. None of our team has any experience using MongoDB or any other NoSQL style database.

  MariaDB is a SQL style relational database very like OracleSQL. It is a derivative of MySQL which works better on a small server with fewer resources. It allows us to leverage our knowledge of SQL databases in our application. All of our team has experience working with a SQL style database.

- **Chosen Approach**
  After taking into consideration our experiences we have decided to go with MariaDB. We all have experience with SQL and understand how it works, none of us have much experience with MongoDB or similar non-relational databases. We, again, rated both database choices using our standard categories below in table 3.

Table 3: Database

Platform/Attribute	Ease of Use	Functionality	Documentation
MongoDB	2	5	4
MariaDB	4	5	4

MariaDB is going to be easier for us to pick up and use immediately due to our previous

experiences. Functionality wise both MongoDB and MariaDB will fit our needs. Documentation is about the same for both database implementations based on our research.

- **Proving Feasibility**
  We will show connecting to the database using our NodeJS backend and adding, retrieving, and modifying table data in the database. We will also show NodeJS creating tables and columns dynamically.

Host this Service in the Cloud

We need somewhere to host the actual application itself. With the technologies, we've chosen we could run on any array of machines, NodeJS runs on Windows and Linux. MariaDB has a windows installer if we decide to go that route. With any web application that stores private data, we need to implement a way to keep our service running well.

- **Alternatives**
  Given that our chosen backend NodeJS service can run on any operating system this gives us many options. There are many, many available web Virtual Private Server (VPS) hosting services. Some hosting services only provide access to NodeJS and no more of the system, others provide access to an entire virtualized operating system. Some main competitors in this space are Amazon Web Services, Microsoft Azure, Linode, and Digital Ocean.

  Amazon Web Services (AWS) is a container based VPS service focusing on scalability. They have many specialized servers for several different applications and services. Most of their containers run Linux which works fine with NodeJS. Two main drawbacks are their complex pricing models and number of different specialized services that they offer. It makes it extremely difficult to manage specifically which service would work best for our specific situation. Also, we should not need to scale the application and host multiple instances of the same application, our user base will not be big enough to need that. Pricing is done at a per hour rate. None of us have any experience with AWS.

  Microsoft Azure offers similar container bases VPS services, with the addition of Microsoft Windows containers. Their pricing is slightly less difficult to understand; they also charge a per hour rate. None of us have any experience with Microsoft Azure.

  Linode is a standalone VPS service that offers high performance SSD Linux based servers. They have a very straight forward pricing model. Their cheapest option that would fit our need is $10/mo. They advertise a 99.9% uptime guarantee. None of us have any experience with Linode.

  Digital Ocean is another alternative offering high performance SSD Linux based servers.

They too have a very straight forward pricing model. Their cheapest option that would fit our need is also $10/mo. They advertise a 99.99% uptime guarantee. One of our team members has experience using Digital Ocean.

- **Chosen Approach**
  We chose to go with Digital Ocean because we have a team member who has experience with it and its pricing model is easy to follow for our sponsor so she can understand what she's paying for. We can host our database and web application on a single VPS without much of an issue for the number of users in our user base that we have. Using our three categories, we ranked the alternatives in table 4 as follows:

Table 4: Cloud Hosting

Platform/Attribute	Ease of Use	Functionality	Documentation
Digital Ocean	4	5	5
Microsoft Azure	3	5	5
Linode	4	5	5
Amazon	3	5	5

All cloud hosting services provide the optimal functionality we need. Linode and Digital Ocean are slightly easier to understand in terms of providing updates, installing and configuring the application, and installing and configuring SSL certificates. Documentation in this case refers to the documentation available about the cloud hosting service in terms of setting up servers, updating servers.

- **Proving Feasibility**
  We can use Digital Ocean to host a simple NodeJS application and MariaDB database and get a feel for how everything works as well as ensure that there are no slowdowns when running both services on the same VPS with limited resources.

Store Images in a Database

A web application that connects two people/groups needs support with pictures. We need to allow users to upload pictures of themselves and view pictures of possible matches. Due to this, we need a database capable of storing images. We also need a good method to communicate those images to and from said database.

- **Alternatives**
  One alternative is to use an image hosting website such as Imgur.com. By leveraging such a site, we can offload image uploading to their servers and simply store the resulting image link and use that when serving images to clients. This brings up privacy

issues as anyone can see the images that are uploaded to most image hosting sites. In addition, this adds an external dependency that we will need to consider when evaluating the reliability of our application.

On the other hand, we can store images in the database itself. By doing this we can ensure that user privacy is upheld and there would be no external dependency on another service for our application to run correctly. This does bring up an issue with uploading and serving images using only our REST API, as well as storing the image data correctly in the database.

- **Chosen Approach**
  Even with the outstanding challenges of uploading and serving images through a REST API we have chosen to store images in our database. This decouples us from another external dependency and allows us to protect users' privacy. Below in table 5 we list our considerations for the different platforms.

Table 5: Image Storage

Platform/Attribute	Ease of Use	Functionality	Documentation
Imgur	4	3	4
Database	4	4	4

Hosting images using Imgur or the database are around the same ease of use from our perspective. Functionality wise hosting the images in the database will give us more functionality that we want over using Imgur due to the privacy issue described above. Imgur provides a lengthy API documentation guide describing uploading and retrieving images from their site. There are many tutorials and documentation online about uploading images using NodeJS with Express that we can leverage.

- **Proving Feasibility**
  We will produce a demo application which demonstrates uploading images and serving images using a REST API, and store images in the database.

Send Emails to Users

A big part of the project is notifying users about events that happen in the web application while they are offline. These events might include: a new match, a new message from a user, an announcement from the administrator, a check in notification, etc. By notifying users about these events quickly via email we can ensure that they do not have to be logged in at all times to view their notifications. In order to make this work we need to have a mail service to provide sending and receiving capabilities on the domain. We will need at least two mailboxes, webmaster@flagfriends.org and no-reply@flagfriends.org. We also need a way to communicate with that mail service from the web application backend.

- **Alternatives**
  For hosting the mail service we can leverage existing services such as: Google Apps, Namecheap, or Zoho. We could also consider hosting our own mail service in a Linux environment. For web application backend, we can leverage one of the many available node modules that allow sending emails using SMTP.

  Using our own mail service in a Linux environment is very complex for us to set up and and difficult for sponsor to use and keep up to date. Many messages have a chance of not being delivered due to being considered spam. The very basic version of this also provides no virus scanning on incoming emails.

  Leveraging an existing email service such as Google Apps, Namecheap or Zoho is ideal because it allows us to simply connect and send emails and not have to worry about keeping the mail server up to date.

  Zoho is a mail service that offers a free tier that perfectly fits our requirements, it allows up to 10 mailboxes and 5GB of storage per mailbox. It is reliable and supports SMTP email messages so we can connect from the web application backend and easily send emails.

  Namecheap offers a mail service for a price of $28.88/yr with up to 10 mailboxes and 10GB of storage per mailbox. They are also reliable and support SMTP email messages.

  Google Apps provides one of the best custom domain email service available, but they are much more expensive than other services starting at $10/mo for the level of service that we need. They are also reliable and support SMTP email messages.
- **Chosen Approach**
  We settled on using Zoho for sending emails as this provides the most while keeping the cost non-existent because we are going to use the free tier. After talking with our sponsor we realized that the other mail services would be overkill and unnecessary, we simply need a way to send emails and receive emails, we do not need file storage or document editing. With this in mind, we laid out our options and considered each one in the next table.

Table 6: Email Service

Platform/Attribute	**Ease of Use**	**Functionality**	**Documentation**
Google Apps	5	3	4
Namecheap	5	4	3
Zoho	5	5	3

Self-Hosted	1	3	1

Terms of ease of use any pre-existing mail service is far easier to use than self-hosting a mail service. Zoho provides the most functionality for the money, allowing us to host up to 10 mailboxes for free. Google Apps provides the best documentation compared to the other available platforms. There is very little reliable documentation online on self-hosting a mail server successfully for an extended period of time without having to worry about maintenance in some way.

- **Proving Feasibility**
  We plan to test this by creating a free tier Zoho account and link it with the flagfriends.org domain. Then set up and configure two different mailboxes, webmaster@flagfriends.org and no-reply@flagfriends.org and ensure that we can send emails using a NodeJS module.

Secure Interaction and User Authentication

Secure interaction with the web application ensures that users can authenticate with the service using a unique username/password combination while ensuring that the username/password combination is correctly encrypted in the backend database. In addition, secure interaction ensures that no “man in the middle” attack can be used to obtain a user’s username/password when authenticating to the service.

- **Alternatives**
  There are a few different technologies we can leverage to overcome this challenge. We can obtain, setup, and use a Secure Socket Layer (SSL) certificate to address the secure interaction portion of the challenge. To resolve the user authentication portion, we can bind each user to a unique id (username or email) and password, and then encrypt the password on the backend ensuring that no one can access the user’s raw password by viewing the database. Alternatively, we can leverage existing services such as Google or Facebook authentication.

  Obtaining a SSL certificate is easy and free thanks to new services such as letsencrypt.org. On the opposite side, maintaining a SSL certificate can sometimes be time consuming and troublesome. A new certificate will need to be obtained before the valid SSL certificate becomes invalid. This can be easily remedied by using letsencrypt’s command line interface (CLI) utility and a simple CRON job to check for an expiring certificate and renew it automatically with minimal to no downtime.

  There is no good alternative to using SSL that would ensure that data is encrypted between client and server without using a homemade solution and building it directly into the web application.

Securing user information by binding each user to a unique ID and password ensures that if the database was compromised user's passwords would still be secure.This means we need to build and maintain functions that encrypt and verify passwords.

Securing user information by using existing services such as Google or Facebook would allow us to simply integrate with their services using an API and have user's log in. However, each and every user needs to have a Facebook or Google account, and some will not. This is not a "one size fits all" solution.

- **Chosen Approach**
  After consideration, and talking with our sponsor we decided to opt for obtaining and using a SSL certificate and building our own account management system by storing email addresses and encrypted passwords in our database. Each option was rated accordingly in the table below.

Table 7: User Authentication

Platform/Attribute	**Ease of Use**	**Functionality**	**Documentation**
SSL	4	5	5
Custom encryption	1	0	1

Creating, distributing, and managing a SSL certificate is far easier than creating our own custom encryption implementation between the client and server. Functionality wise SSL gives us the security we need without having to do manual encryption. There is not much documentation about using custom encryption between a client and server. However, there are many SSL certificate configuration tutorials online that we can leverage.

- **Proving Feasibility**
  We plan to test our chosen approach by obtaining and using a SSL certificate from letsencrypt as well as building a simple account management system that allows users to create an account and login.

Communicate Between Client and Server for Events

We need a way to have a quick client-server connection in which we can easily emit an event on the server and have the client respond to the event and produce a notification for the user. By establishing a bidirectional communications system we can quickly act and respond to events on the client and have the server be notified instantly. This will eventually serve as a basis for the application's internal user-to-user messaging system.

- **Alternatives**
  There are a few technologies out there that will allow us to achieve this level of synchronous communication. We could expose a REST type API for events that we can poll every few seconds or we could leverage WebSockets.

  Exposing a REST API for notifications would give us more coverage on every type of browser, however it would be more difficult to manage as we would need to write the logic to write the entire event manager workflow. This adds time to the project development in addition to testing that will need to be done to verify full functionality. In order to quickly respond to an event the client will need to hit some sort of event REST API endpoint describing their reaction to an event. Essentially creating a bidirectional communication stream between the client and server would be difficult if we went this approach.

  Leveraging WebSockets would allow us to easily and quickly establish a near real time persistent bidirectional communication between the client and server by using events. According to caniuse.com (a website that tracks different web technologies and what portion of users can use a certain service) WebSockets can be used by 91% of all users. However, socket.io provides a polyfill implementation of WebSockets upping this usability percentage. There are many WebSocket libraries that support NodeJS, one main one is socket.io.

- **Chosen Approach**
  We decided to go with WebSockets because they allow us to have an easy bidirectional communication with the server using events rather than using a REST API for event driven communication. These two options were rated again, but this time with the current challenge on our minds.

Table 8: Client/Server Event Communication

Platform/Attribute	Ease of Use	Functionality	Documentation
REST API	4	2	3
WebSockets	3	5	3

Using a REST API for events would be easier to use to some extent as we are all somewhat familiar with using RESTful APIs. WebSockets would be more difficult to use initially, but provide exactly the functionality we need without having to implement much helper code. Documentation is available that describes using a REST API using polling for bi-directional event handling. The socket.io library lacks a little bit of documentation, but there are many examples available that we can analyze and use.

- **Proving Feasibility**
  We will show bidirectional event based communication using the backend NodeJS server and the frontend AngularJS 2 client.

Administrator customizable interface

Customizability is another big piece of the project that our sponsor wants to incorporate. By providing the application in the most generic way we can allow administrators to customize specific pages for their needs. They would need to specify certain attributes like the site name, site logo, site landing page content, FAQ page contents, etc.

- **Alternatives**
  This can be done during initial setup and modified by an administrator at a later time. Alternatively, we could directly modify template files for each use case, but that would require each different instance of the application to be customized prior to being deployed in a production environment, which gets away from the generalizability and ease of use that our sponsor wants out of this product.

  By allow each page to be partially customized we can get the best of both worlds, this still allows for enough customizability to allow for other organizations to easily modify the application for their own use. If an organization wanted to they could go in and modify hard-coded template files to change the direct layout of the application easily because of our use of AngularJS 2. One big issue with this is ensuring that there is no page load lag when getting the page contents from the server, we may be able to pre-render pages on the server with the correctly customized content before serving it to clients.

  Hard coding each layout contents directly would greatly diminish the customizability a general administrator has control over. This would mean that anytime a static page needed to change they would have to turn off the service, modify the HTML in the page themselves, and then restart the service. The advantage of using this is that there is no page content load lag when loading static pages like the homepage or FAQ pages.

- **Chosen Approach**
  We decided to go with allowing each page to be partially customized, each page will still have similar layouts but the textual contents of each page can be changed to suit whichever organization may be hosting the application and what it's being used for.

Table 9: Client Customizability

Platform/Attribute	**Ease of Use**	**Functionality**	**Documentation**
Partial Customizability	3	4	N/A
Hard Coded Layouts	5	2	N/A

Hard coding our layouts is the easier option, however, to make the application as customizable as possible we will need to support some sort of partial customizability through our REST API.

- **Proving Feasibility**
  To show that this works we will have a simple interface in which you can change some textual field on the site and show that the change will be applied for everyone.

Generate reports

The administrator should be able to generate custom reports based on user attributes and statuses. For example, the administrator should be able to generate a list of users who have not completed their check-in forms in the last 30 days, or the administrator should be able to generate a list of hosts who have not matched with any student yet.

- **Alternatives**
  To support the frontend of the reporting mechanism we could use a javascript library like d3.js, or leverage AngularJS 2.

  Using d3.js allows us to create interesting looking visualizations of data easily using HTML5 canvases. However, it may be a little overkill for our needs. It would be another Javascript library to learn on top of AngularJS 2.

  Leveraging AngularJS 2 would allow us to easily create a dynamic table of results that could easily convey the message we across to the user. If we wanted to add other types of visualization we could easily implement them using pure AngularJS 2.

  To support the backend of the reporting mechanism we could use a robust REST API that allows for complex queries or WebSockets.

  Utilizing a REST API would allow us to be more in line with the rest of our application, some of these reports could take time to generate depending on the complexity and REST APIs are perfect for queries that might take a long time to process (either by the database or by the backend itself). However, it means we need to build either one extremely flexible endpoint that allows for a variety of parameters to be specified like text filters or other special filters. Or we need to build multiple endpoints each supporting one single piece of the functionality which can also easily complicate the project.

  Using WebSockets would allow for instant communication of results, from the server and a persistent connection that may produce results slightly quicker based on the fact that we can leverage knowing the context of the previous request easily. However, this can also be done using a REST API and session storage. Some requests may take time to process which means we would need to handle knowing when we should be getting a response back or building in a “progress” indicator of some kind.

- **Chosen Approach**
  Using a technology such as d3.js would be great if we had data that could be better visualized outside of a table of entries. In our case, all our reporting should just generate tables of data that the administrator can view and evaluate. We may have some simple pie charts but that can be accomplished using javascript and some styling.

  After looking at our selected technologies we decided to go with using a single endpoint REST API as a kind of generalized report data generation endpoint. This makes the backend slightly easier as we only have one endpoint, but also slightly more difficult because there are a lot of parameters to consider.

Table 10: Client/Server Report Communication

Platform/Attribute	Ease of Use	Functionality	Documentation
REST API	4	4	3.5
WebSockets	3	4	1

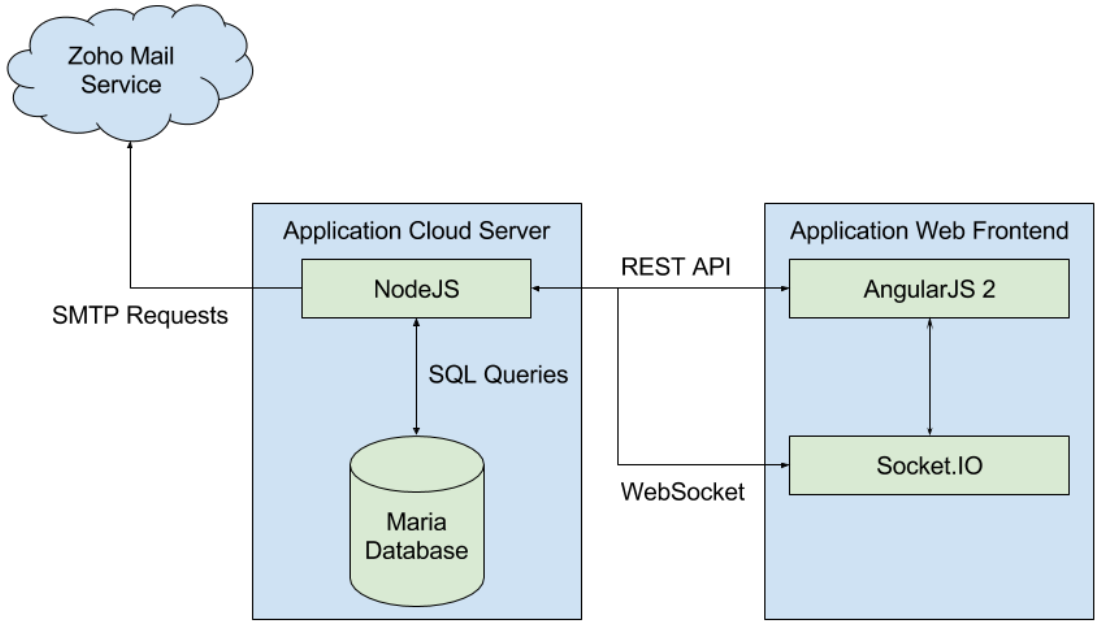
  Using a REST API will be the easier option as we all have more experience using REST APIs over using WebSockets. Both technologies can be used to achieve around the same functionality in the end. There is more documentation available online for using REST APIs to get report data than using WebSockets for this purpose.

- **Proving Feasibility**
  We will create a simple reporting page that shows getting data using a REST API endpoint on the server that supports a variety of different distinct parameters.

Technology Integration

Integrating our technologies based on the above proposed solutions and chosen approaches should be straightforward. By selecting technologies that work well together and have online examples and documentation available we can quickly begin integrating the different mini-solutions described above into one cohesive piece of software. Although some integration details are already covered by the above sections, additional integration will be needed. Figure 1 below shows a system diagram outlining many of the technologies described above.

Figure 1: System Diagram



Looking at Figure 1 shows the two distinct pieces of the application, the Application Cloud Server that is hosted on an external server and the Application Web Frontend that is the webpage itself as a user would see it. Our server will be hosting both our NodeJS application as well as the MariaDB instance. By combining these into one server we can save our client some money. The application frontend primarily relies on AngularJS 2 that provides an access to a simple connector that performs RESTful requests. This is augmented with using WebSockets through socket.io for instantaneous bidirectional communication between client and server. In terms of connections between services, we will need four different communication methods: SMTP Requests, SQL Queries, REST, WebSockets. Installing NodeJS on the cloud server will work no matter what OS is used, in our case hosting with Digital Ocean will give us a Linux environment. MariaDB also works on Linux without an issue. Having NodeJS communicate with the MariaDB can be achieved by using one of NodeJS's available libraries that provide support for either an object relational model (ORM) connection or pure SQL queries. NodeJS supports opening and hosting WebSockets using socket.io. Again, NodeJS can be used to communicate with the Zoho mail service using an existing NodeJS library.

Conclusion

Our sponsor needs a way to avoid lengthy paperwork and the manual operation of Flagfiends. Our solution is to create a general purpose redistributable web group matching application. This application will take advantage of new web technologies in order to provide a easy to use, responsive web application. The purpose of this document is to explain the technical feasibilities, analyze the technological challenges and provide solutions based upon currently available technologies.

This project will allow our sponsor to digitize her entire workflow and bring it into the modern era. This project will provide a way for one group of people to connect with another group using a fast-interactive online website interface. In table 11 below we have listed a brief overview of our challenges, chosen solutions, and confidence level for each challenge. This should help us to look further into challenges that we do not feel as confident with.

Table 11: Challenge Summary

Challenge	**Chosen Solution**	**Confidence Level**
We will need a way to easily generate a responsive single page frontend interface	Use AngularJS 2	90%
We will need a way to provide a RESTful API for the frontend to communicate with that handles all our backend logic	Use NodeJS and Express	90%
We will need a secure way to store user data in the web service	SQL with MariaDB	90%
We will need a way to host this service in the cloud	Use Digital Ocean to host server.	90%
We will need a way to retrieve, upload, and store images to a database	Use express REST API to upload, retrieve images. Use the database to store uploaded images.	80%
We will need a secure way to send emails from the web service to users	Use Zoho for email hosting.	100%

We will need secure interaction and user authentication on our web service	Obtain a SSL cert from LetsEncrypt, use a unique email and password combination for each account. Each password is encrypted on the server.	80%
We will need a way to quickly communicate between client and server for chat messages and other instantaneous events	Use WebSockets with socket.io.	70%
We will need a way to provide a customizable interface for the web service	Allow each page to be partially customized, storing customizations in the database.	70%
We will need a way to generate reports based on metadata	Use a single REST API endpoint with customizable parameters	70%

This analysis process allowed us to look intently at our project and extract major potential challenges and research what some solutions to them might be. By going through each challenge we were able to discuss and grow as a team while thinking about our best possible approaches based on our current knowledge and the current requirements for the project. Our choices of technologies above will allow us to complete our project effectively and efficiently. We are looking forward to working with the project to make it a success. The resulting general purpose software will allow any number of organizations to take advantage of the software quickly and easily without much effort.