# Boston University
# Electrical & Computer Engineering
**EC463 Capstone Senior Design Project**

# First Semester Report

# Membership Portal for the Diva Docs Black Women MD Network

Submitted to:

Philomena Asante
881 Commonwealth Avenue, Boston, MA
617-353-3575
pasante@bu.edu


By: Team #3 - Diva Docs

**Team Members**:
Paige DeVeau | pdeveau@bu.edu
Mya Turner | mjturner@bu.edu
Akhil Sehgal | asehgal1@bu.edu
Vinay Metlapalli | vinaymet@bu.edu
Abhinoor Singh | abhinoor@bu.edu

Submitted: December 8, 2022

**Customer Sign-Off** _____

# Table of Contents

# Membership Portal for the Diva Docs Black Women MD Network

Paige Deveau, Mya Turner, Vinay Metlapalli, Akhil Sehgal, Abhinoor Singh

**Executive Summary**- Our mission is to correct the injustice that has kept Black women doctors from achieving leadership positions by providing a tier-based membership website portal accessible nationwide that enables them to network with peers, find mentors and new opportunities that are specifically looking for diverse candidates.. Our proposed technical approach is to use React TS, MongoDB, and a full suite of APIs to develop a website that will have an easy-to-understand homepage highlighting website features, an application form with payment fulfillment and membership tiers, and a searchable membership directory. Our goal in building this is to not only improve health access and outcomes for the Black population but also to diversify the healthcare leadership and governance workforce in the communities they serve.


- *Paige DeVeau is with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215. Email: pdeveau@bu.edu*
- *Mya Turner is with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215. Email: mjturner@bu.edu*
- *Vinay Metlapalli is with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215. Email: vinaymet@bu.edu*
- *Akhil Sehgal is with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215. Email: asehgal1@bu.edu*
- *Abhinoor Singh is with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215. Email: abhinoor@bu.edu*


## 1 Introduction

Our client, Diva Docs, is a 501c3 nonprofit organization that prepares Black women doctors for senior positions in medicine and healthcare through professional development programming, leadership training, one on one coaching, and member partner mentorships and sponsorships. Diva Docs is currently Massachusetts-based but wants to transition their platform nationwide through a membership portal website. The goalsfor this new website are to provide new users a platform to apply for one of the membership tiers, and, once accepted, view other approved members in a "Membership Directory." The platform will provide resources such as a resume bank for recruiters to access and the ability to reach out to a mentor and offer leadership and management training and networking events in various communities. Users will sign up using our portal and provide information about their background, work history, and resume before it is vetted by an internal system. The nature of our project requires an approach that has two major components: front-end development of website design, user experience, and the overall user interface of the website, and back-end development of the database infrastructure where data from the website is stored and retrieved and creating custom APIs to drive the functionality of the website. Members of our team are divided into these two teams but work together to ensure that the back-end and front-end components continue to work in unison as overall development continues. Our approach ensures that our client will have a fully functional website capable of fulfilling the ambitions outlined above but can also handle nationwide user traffic. The homepage and overall website design are led by a professional UX/UI designer in conjunction with our team and the client. The website also fully facilitates the new user application process. This process presents the user with a client-designed form and a secure payment portal embedded right into the website. Additionally, the website has a "Membership Directory" in which approved users can search, filter, view, and interact with other approved users. The website's back-end is robust with a fully integrated MongoDB database environment and a suite of APIs. The MongoDB environment has several databases, with corresponding sub-collections to ensure that data is organized and can be retrieved efficiently. The APIs facilitate the transfer of data to and from the MongoDB environment and fulfill subtasks as required by the front-end.

## 2 Concept Development

Diva Docs approached our team with the project of expanding their company nationwide through a tier-based membership portal website. Our team understood the problem and concluded that creating a new membership-tier-based website that could handle the nationwide expansion of Diva Docs was necessary. Furthermore, we concluded that in order to solve this problem, the minimum viable product would be a website that had a homepage, an application form for new users, and a place where accepted users could view, filter, and interact with other approved users. With these general concepts in mind, we began development on each of them. A significant part of any website is the design; the first major concept we established was the overall website design. As a team, we knew that we would need to develop the design concept for the three major components mentioned above. For each major component, we worked closely with a professional UX/UI designer to develop a full suite of wireframes. Our team and the client agreed that our design should be accessible. The portal is required to have an overall training time of fewer than two minutes, with the application process taking less than five (view Appendix I for comprehensive engineering requirements). The overall color palette was chosen by the client after presenting the wireframes that displayed several options. Wireframes were developed for all three major website components: the homepage, application form, and membership directory. These wireframes helped us conceptually develop the website and simultaneously figure out the more intricate functionality such as navigation and search query. Additionally, it allowed us to clarify the client's

requirements and how to meet these metrics. During our several design reviews, we determined that using React TS as our front-end platform was optimal. We chose React TS as opposed to alternatives such as Inferno JS, Ember JS, and Backbone JS, as we had more experience with React TS and it fulfilled our performance metric requirements (outlined in Appendix I, Engineering Requirements). As we developed the wireframes and selected the final candidate, we developed conceptual technical designs for the front-end. These designs distilled major design elements like the homepage, down into technical React components such as navigation bars and forms. This component-based conceptual development ensured that we could break larger front-end tasks into smaller ones (view Appendix II for front-end agile sprint plans). Our team followed this same approach for the conceptual development of the back-end of the website. As outlined in our minimum viable product description above and by our client, the application form functionality carried the most weight. The application form has several inputs from the user and therefore requires a database architecture that is robust enough to receive, send, and filter data from the users. We prioritized flexibility for the database provider as, over time, the application form itself could change. Additionally, we needed a database that made retrieving data based on queries and filters efficient as the functionality of the membership directory depends on it. We began researching SQL and NoSQL databases. We discovered that NoSQL databases promised increased flexibility with data input and retrieval without a decrease in performance over SQL databases. Additionally, NoSQL databases enable faster application development, handle highly diverse data types, and manage applications more efficiently at scale. The ability to handle diverse data types was a significant factor for our team as during the application process, users can attach a resume and professional photo; being able to store this kind of data was necessary for us as well. After deciding on NoSQL for these reasons, we began investigating NoSQL database providers. We selected MongoDB as the provider because it stores data as JSON-like documents (binary JSON) which means that, if the application form is changed, previous data will not get affected, because, in effect, each document has its own schema associated with it. With MongoDB now selected as our database provider, we began to conceptualize how we organized the database architecture. Each MongoDB environment houses several databases with associated sub-collection databases. We conceptualized organizing the environment into two databases: one for the application form page and one for the membership directory. The application form database has two sub-databases (called "Collections" in MongoDB): one is called "Application Submissions" and one is called "Approved Applications." The "Application Submissions" collection houses the application form data of all user applications. If the user's application gets approved, their data is then moved to the "Approved Applications" database. The Membership directory database has one sub-collection that houses all of the users currently listed on the membership directory. We debated pulling data directly from the approved application collection for the membership directory but wanted to have an added layer of flexibility in case some members are removed from the membership directory but are still members. For example, if a member does not pay their membership fee on time, or violates community guidelines, they can still be members stored in the "Approved Application" collection but will be removed from the membership directory collection. All database operations such as reading, writing, and transferring data across collections are handled in Python using the PyMongo library. The final component of our development for the website was to determine which platform we would host our APIs on. From the start, our team realized that in order for each component of our conceptual development to function together we would need a full suite of APIs to facilitate the transfer of data and functionality between our front-end user interface and back-end functionality. We considered two RestAPI platforms for our API: Flask and FastAPI. Our team was all familiar with Flask, as we have used it on several previous projects and were familiar with its functionality and requirements. Our team has also been working closely with BU Spark, and one Spark member suggested FastAPI as an alternative to Flask. We ended up choosing FastAPI as our API hosting solution because it is a framework fundamentally used for building robust production-grade APIs. Flask, on the other hand, is used primarily for prototyping. In addition, FastAPI is also significantly faster from a performance perspective, with only Starlette and Unicorn offering faster performance. Our overall conceptual design provides an overall system architecture using React TS as a front end, MongoDB as our NoSQL database provider, and FastAPI as our API hosting service.

## 3 System Description

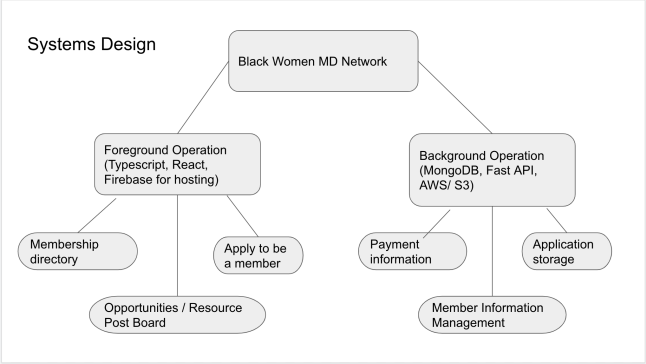



*Fig 3.1: Block Diagram of the Black Women MD Network membership portal website.*

Fig 3.1 shows the block diagram of our website. Many of the components discussed in the conceptual description are shown. The block diagram describes the front-end design flow as "Foreground Operation" which includes the components of the design that the user interacts with as well as the technologies that support them. The front-end components are driven by React Typescript (React TS), the coding language used to display and drive the functionality behind all the website components such as buttons, images, and user inputs. Firebase is the technology component used to host the website. Google Firebase reduces development workload and time and comes with a full suite of Google tools such as Google Analytics that our client can use to monitor website traffic and overall outreach of the website. The front-end consists of three major components: the membership directory, a place to apply to be a member, and an opportunities or resource post board. Each of these front-end components reacts directly with elements of our

back-end system design. The member application front-end component interacts directly with the payment information and application storage components of the back-end. When a user applies on the website, they will be required to pay a fee associated with their particular tier of membership. To facilitate the payment process, we integrated "Stripe," a third-party payment fulfillment platform, directly into the website. We decided to employ a third-party platform for payment as we did not want to be liable for insecure payments or obtain the necessary licenses to facilitate payments. Additionally, based on time constraints, the level of security we could develop for the payment would not be up to the standard of a well-established third-party provider like "Stripe," which is used widely across the internet on other websites. It is important to also mention that as we offload the payment process to Stripe, we will not in any capacity be storing the payment information of an applicant. However, the data that is inputted by the user into the form itself is stored in our MongoDB environment in the "Application Form" database within the "Application Submissions" collection. This process is highlighted in the system described as "Application Storage." This data transfer process is facilitated by one of our custom APIs that uses the Python library for MongoDB "PyMongo" to write, read, and transfer data between the various databases and collections. Similarly, the membership directory front-end component also interacts directly with the back-end component outlined in the system described as "Member Information Management." Once an applicant is approved, their data is transferred to the "Membership Directory" database in MongoDB. Additionally, when an accepted user navigates to the membership directory front-end component, data is pulled directly from the membership directory database to inform the front-end display in the directory. This process of pulling data from the right database and sending it to the front-end is also handled by one of our custom APIs hosted on FastAPI.

The third front-end component described in the system is the "Opportunities / Resource Post Board" which also interacts directly with MongoDB and our APIs. A separate database in the MongoDB environment handles this component. In the MongoDB environment, there is a "Post Board" database with two collections. One for submitted posts and one for approved posts. When a user wants to post on the post board, the content of the post will be sent to the submitted posts collection, which can then be retrieved and reviewed before moving it to the approved posts collection. Then the post board front-end component will pull data directly from the approved posts collection (again, through one of our custom APIs). The following pseudocode describes the process of writing and reading to the MongoDB databases:

```
function read_write_to_mongo(data):
      establish a connection to Mongo
      select DB to read/write
      select collection within DB
      if writing:
            construct JSON obj
            write obj to collection
      if reading:
            for doc in collection: print
```

As shown in the pseudocode, reading and writing from a MongoDB database and its associated collections is straightforward. Our database architecture also ensures that the data is well organized and can be manipulated efficiently. Additionally , resumes and applicant pictures are also submitted and stored along with the applications. MongoDB can also handle these forms of data and use their internal GridFS application. However, the pictures and resumes must be stored in different databases than where the applications are stored. To combat this, each application component is assigned a unique identifying key that is attached to all forms of a user's application. That way, it is easier to pull all documents in the database for a particular user.

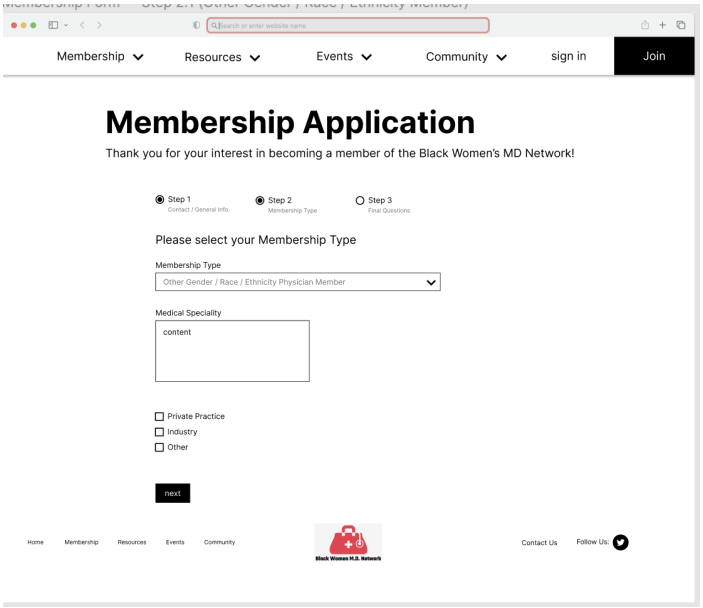



*Fig 3.2: Initial wireframe for the membership application front-end component.*

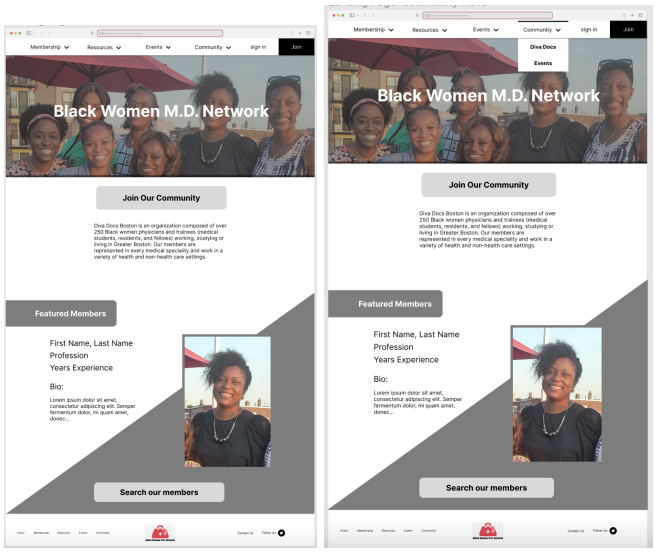



*Fig 3.3a (Left) & Fig 3.3b (Right): Initial wireframes for the landing page of the Black Women MD Network Membership Portal Website.*

Figures 3.2 and 3.3 show the initial wireframes for the Member Application form and the landing page components of our front-end (a link to all of the wireframes will be provided in the appendix). Currently, the wireframes for the Membership Directory are in development and will be implemented in the next

evaluation period (outlined in the Technical Plan, Section 5). It is important to mention that all of the system design technical components described in Figure 3.1 have been successfully implemented and are being used for development, including the website already hosted on Firebase at (blackwomenmdnetwork.com) which was a client-provided domain and a requirement for where our website was to be hosted.

## 4 First Semester Progress

During the current performance period, our team implemented Agile project management techniques using Atlassian products like Jira and Confluence. Sprints were set at two-week time periods; tasks within those sprints were assigned to the appropriate team members depending on whether they were part of the front-end or back-end team. These sprint boards were on Jira using the Kanban Board application.

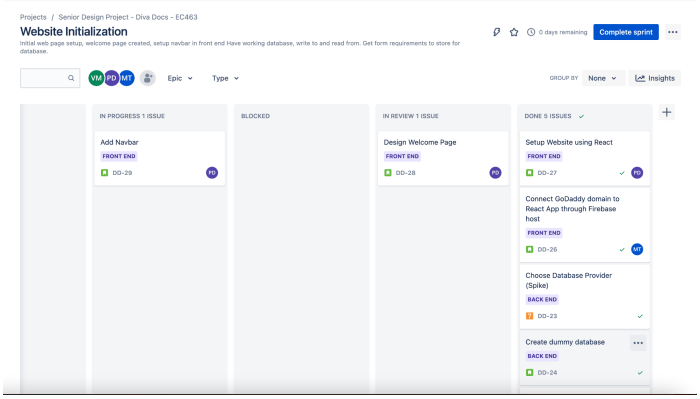



*Fig 4.1: Jira Kanban Board for the Website Initialization Sprint. Link to our Jira Sprint Boards will be provided in the Appendix.*

By breaking down our development into sprints, we are able to complete initial versions of the website, test, and make modifications as the client requested. Additionally, for documentation and meeting notes we used Confluence. The progress made in this performance period is best broken into the two major components of our website, the front-end and the back-end.

**Front End Progress:** The first major accomplishment of this evaluation period was successfully developing wireframes for the landing page and application form. We worked with a professional UI/UX designer to develop these wireframes and presented them to the client to ensure the wireframes encapsulated the functionality and design requirements. With the wireframes finalized, we successfully hosted the website on the client-provided domain www.blackmdnetwork.com using Firebase as our hosting platform. Next, we successfully began implementing the designs. The landing page now has a fully functional navbar that navigates to the appropriate URL associated with each of the pages on the website. The landing page matches the color scheme of the wireframes to the exact hex color profiles as the wireframe. As is shown in the wireframe, we added a search bar to the landing page, as well. Additionally, we successfully began implementing the designs for the application form page. The application form page is also paired with a fully functional API that writes the data to the appropriate MongoDB database (more details in the following section outlining accomplishments made on the back end).

**Back-End Progress:** *Accomplishment B1: Research and Decide on a NoSQL or SQL database*. We successfully conducted in-depth research into both solutions and decided on NoSQL as the best format of database to use (a detailed description of this decision can be found in Section 2). *Accomplishment B2: Research NoSQL Database Providers and Choose One.* We successfully conducted research on NoSQL databases and chose MongoDB as the database provider for the website (a detailed description of the reasoning behind this decision can be found in Section 2). *Accomplishment B3: MongoDB Data Architecture Setup.* We successfully created a MongoDB environment for the website and created the three necessary databases to fulfill the functionality of the entire website. We created three databases: Application Form data, Membership Directory data, and Post Board data. Each of these databases had sub-directories associated with each to organize the data. The Application Form data had two sub-collections: one for submitted applications and one for approved applications. The Membership Directory database has one sub-collection for active users on the membership directory. The Post Board database has two subcollections: one for submitted posts, and one for approved posts. MongoDB is a document-based database that allows data to be inputted without the constraints of a schema. This ensures our database architecture is robust and can handle changes in the front-end. *Accomplishment B4: Python Scripts to Read & Write to the Mongo Databases.* We successfully wrote and tested Python scripts to read and write form data (including resumes and images) to several databases in the MongoDB environment for the website. The Python scripts established connections with the databases and wrote and read data using MongoDB's Python library named "PyMongo." These test scripts constructed complex sample JSON objects with nested data to ensure that the application form data will display appropriately in MongoDB and, more importantly, be retrieved correctly. These Python test scripts served as the foundation for how our APIs will interact with MongoDB. *Accomplishment B5: Python Script to Transfer Data from One Database to Another*. We successfully wrote a Python script to transfer data from one database to another. This functionality is crucial to the website's back-end because, for example, once a user is approved, their data is moved from the collection of applications submitted to applications approved. Another example is moving data from submitted posts to approved posts. *Task B6: Research & Determine API Hosting Service.* We successfully researched and chose FastAPI as the service we would use to host our APIs (the reasoning for this decision is outlined in Section 3). In summary, we chose FastAPI for its performance and scalability to enterprise-level production APIs. *Task B7: Write an API to Gather Application Form Data and Send the Data to MongoDB.* We successfully wrote and comprehensively tested Postman API to write our databases in MongoDB. This process will be repeated several times throughout the website as we continue to develop the front-end.

## 5 Technical Plan

Back-end Tasks:

Task 1: Secure all FastAPI Endpoints with OAuth 2.0
Use OAuth 2.0 to secure the different endpoints to make it simple and secure for the front end. Lead: Abhinoor Singh, Assisting: Vinay Metlapalli

Task 2: Write payment endpoint in FastAPI using third party Stripe.
Use Stripe and integrate the Stripe API into FastAPI to create an endpoint where payments can be processed based on the subscription tier the user chose. Lead: Vinay Metlapalli, Assisting: Abhinoor Singh

Task 3: FastAPI Endpoints for Approval/Decline of Applicants in admin portal.
Create endpoints for approval/decline of applicants. Approval will call a function that sends a verification email to the user and prompts them for a registration, this will also move the applicant into the approved members database in MongoDB. Declining will send an email that the application was denied and the user will be moved to the declined user database. Lead: Abhinoor Singh, Assisting: Vinay Metlapalli

Task 4: MongoDB Scripts to move data around
Create different scripts that will allow data to be moved into various collections based on the endpoint called in FastAPI. Lead: Vinay Metlapalli, Assisting: Abhinoor Singh

Task 5: Postman Testing Suite
Create POST/GET requests to test the new endpoints implementations. Test all endpoints for querying data and over all functionality. Load test deployment when API is hosted. Lead: Abhinoor Singh, Assisting: Vinay Metlapalli

Task 6: FastAPI Endpoints for Membership Directory
Create endpoints for membership directory similar to the admin portal to be able to display user data onto the webpage. Lead: Abhinoor Singh, Assisting: Vinay Metlapalli

Front-end Tasks:

Task 1: Create Admin Portal using React Typescript.
This portal will only be available to the administrator and will display the applications that have not been approved and give the administrator the ability to approve or decline the application.
Lead: Paige DeVeau, Assisting: Mya Turner

Task 2: Create login page using React Typescript.
The sign in page will prompt the user for the username and password for their account and check that the user exists before directing them to the available features for the network's members.
Lead: Mya Turner, Assisting: Paige DeVeau

Task 3: Create Member Directory using React Typescript. The directory will display all members of the network in an organized table. These members can be filtered by location or by job title.
Lead: Paige DeVeau, Assisting: Mya Turner

## 6 Expected Budget Estimate for Production Deployment

| Item | Description | Cost |
|---|---|---|
| 1. | Domain | $35 Biannually |
| 2. | Shared MongoDB Server | $57/month |
| 3. | Stripe API Processing Fee | 2.99% + $0.30/ transaction |
| 4. | MailGun Subscription | $35/month |
| 5. | Python3 FastAPI | $0 |
| 6. | React.ts | $0 |
| 7. | Jira Board to track sprints and feature developments | $0 |

Our budgeting total for our development environment is currently $0. The doman is provided by our client and we are using a free version of MongoDB, FastAPIs, and Jira to develop our MVP. The highest cost of the initial deployment is the Stripe payment API that will take a cut of each transaction the user will make. After acquiring an initial user base through DivaDocs, we plan to scale our platform nationwide. Through the growth of our user base past 5000+ active users, we will need to subscribe to a premium version of MongoDB and obtain a dedicated server rack to store user data. In addition, we will need to scale our MailGun subscription to reach more users monthly through emails and login confirmations. Our initial deployment will be able to handle the initial set of users but will require more infrastructure to support user growth.

# 7 Attachments

## 7.1 Appendix I - Engineering Requirements

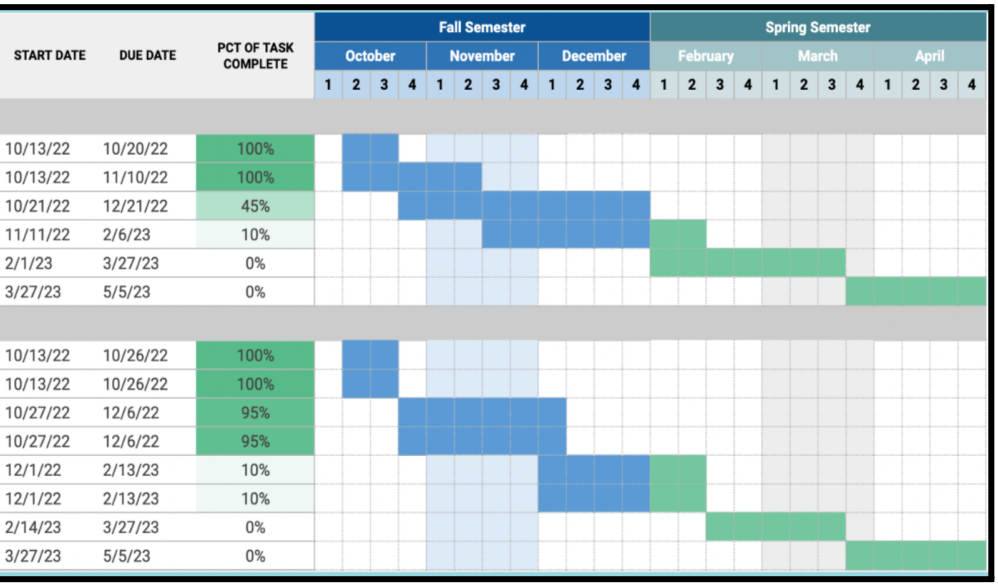
**Team #_03_____ Team Name: DivaDocs**

**Project Name: DivaDocs Membership Portal Website**

| Requirement | Metric |
|---|---|
| Website Must Be Compatible on Multiple Platforms | accessible on 100% of browsers |
| Pages must load efficiently | Website pages must load in <5 seconds<br>Backend API's will have a response time of < 900 milliseconds |
| Website must be easily accessible and easy to use | Learning how to use the website should take < 2 minutes |
| Filling out the application form should be simple | Filling out the application form should take < 5 minutes |

## 7.2 Appendix II: Gantt Chart

| WBS NUMBER | TASK TITLE | TASK OWNER | START DATE | DUE DATE | PCT OF TASK COMPLETE |
|---|---|---|---|---|---|
| 1 | **Frontend** | | | | |
| 1.1 | Deploy Website | Mya | 10/13/22 | 10/20/22 | 100% |
| 1.2 | Homepage | Paige | 10/13/22 | 11/10/22 | 100% |
| 1.3 | Application Form | Mya | 10/21/22 | 12/21/22 | 45% |
| 1.4 | Admin Portal | Paige | 11/11/22 | 2/6/23 | 10% |
| 1.5 | Member Directory | Mya | 2/1/23 | 3/27/23 | 0% |
| 1.6 | Testing | Mya | 3/27/23 | 5/5/23 | 0% |
| 2 | **Backend** | | | | |
| 2.1 | Create Database | Vinay | 10/13/22 | 10/26/22 | 100% |
| 2.2 | Create API | Abhinoor | 10/13/22 | 10/26/22 | 100% |
| 2.3 | Database for Application | Vinay | 10/27/22 | 12/6/22 | 95% |
| 2.4 | API for Application | Abhinoor | 10/27/22 | 12/6/22 | 95% |
| 2.5 | Database for Directory | Vinay | 12/1/22 | 2/13/23 | 10% |
| 2.6 | API for Directory | Abhinoor | 12/1/22 | 2/13/23 | 10% |
| 2.8 | Connect to Frontend | Abhinoor | 2/14/23 | 3/27/23 | 0% |
| 2.7 | Testing | Vinay | 3/27/23 | 5/5/23 | 0% |



## 7.3 System Design