Boston University
Electrical & Computer Engineering
EC463 Senior Design Project

Second Prototype Testing Plan

by

Team 3
DivaDocs

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

<u>Required Materials</u>

Software:

- Front End
    - Wireframes
        - Home Page
        - Application Form
    - React.ts Components
        - NavBar
        - Footer
    - Pages
        - Home Page
        - Coming Soon Page
        - Application Form
        - Admin Portal
- Backend
    - Python3 MongoDB Database Scripts
    - MongoDB Database Environment
    - Python3 Models
    - Dummy JSON Object for the Application Form
    - Python3 FastAPI Post Requests
    - Stripe Payment API (Test Mode)
    - Postman Test Requests

<u>Set Up</u>

The setup for our project is split up into three different components:

- Design
    - The design for the front end uses a software called Figma. Inside of Figma we have created mock designs of what the website should look like, and how it should function. With Figma we are able to show what the buttons on the page look like, as well as where they take you. The wireframes specify the size of the HTML objects, the colors, as well as the different components needed for each of the web pages.
    - The design for the backend is a system design diagram which shows how the front end connects to the backend using an API, as well as where the website is hosted. The system design also includes the connection to the database that we are using.
    - The designs are sent to the client for approval before we begin programming them. Currently we have approval for the home page as well as the application form for

    - the backend. We are still awaiting the final approval for how the application will look in the front end.
- Front End
    - Using React.ts the front end is split up into multiple pages with various components. Currently we have the homepage and application form completed. We also have a navbar at the top of all the pages and a footer at the bottom of all the pages to be able to visit different pages easily.
    - We have a separate React.ts project created for the admin portal which contains a table of all applicants and the ability to click on the individual applicants.
- Back End
    - There are several python scripts, each script serves its own purpose.
        - Database Script allows a connection between the API and the MongoDB database. Data can be written to, and read from the database, including handling the payment API and automatic email sending functionality.
        - The FASTApi script has a post request endpoint setup which accepts a json object and uses pydantic to parse the JSON object and store them into python classes.
        - The models script uses pydantic to create python classes that have the various fields found in the application form.

Pre-Testing Setup Procedure

Front End:

1) Run react with all components from the working directory.
    a) npm start

Back End:

1) Run python3 script using uvicorn that runs the FastAPI backend.
    a) uvicorn main:app --reload (must be in working directory)
2) Open Postman with preloaded POST requests to the end point for the application form
    a) The POST requests will have different request bodies to show the different aspects of the API.
3) API is now deployed to DivaDocs heroku and can be accessed using the following link to view documentation for the API:
    a) https://se-diva-docs.herokuapp.com/docs

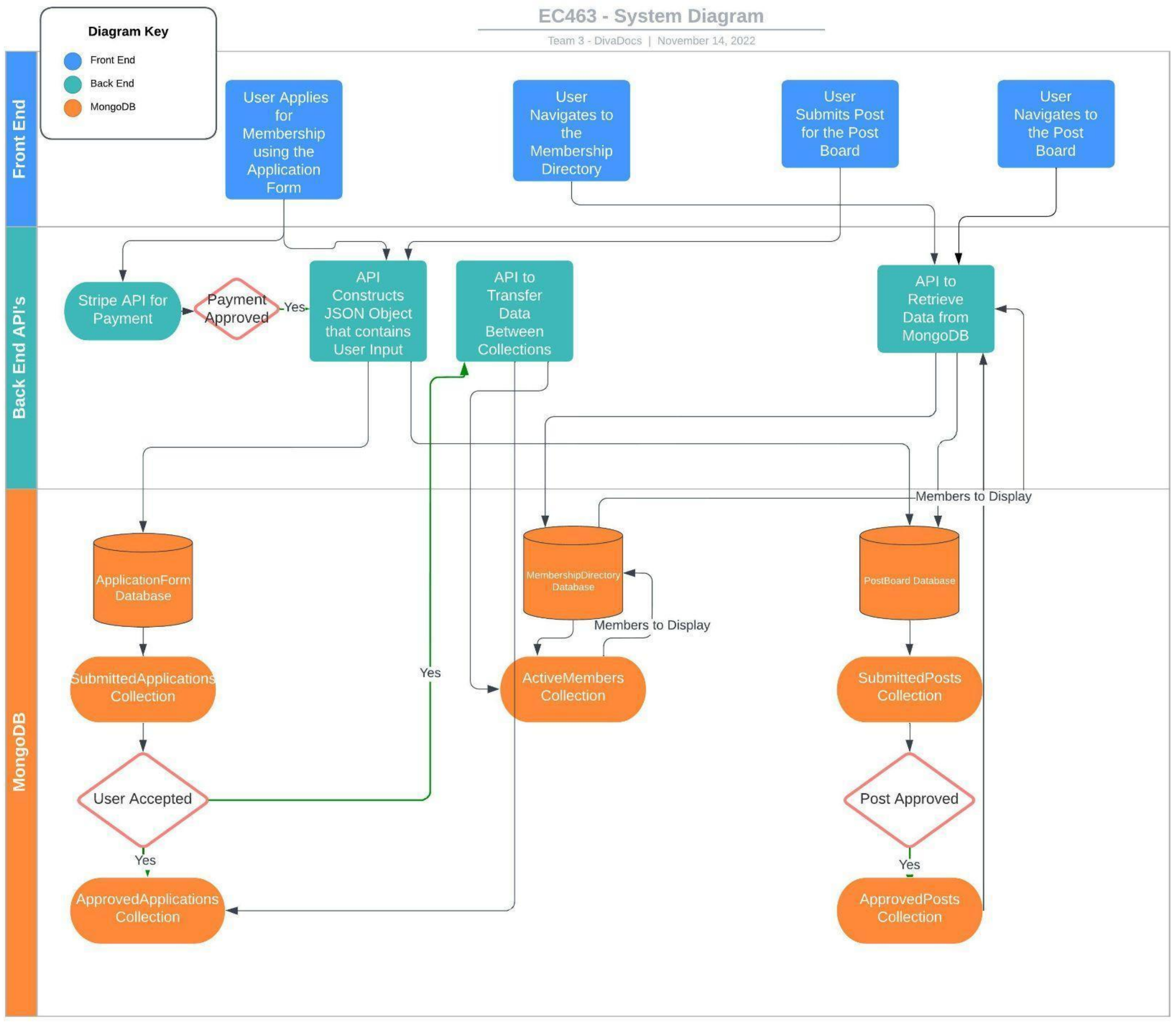
EC463 - System Diagram
Team 3 - DivaDocs | November 14, 2022
Diagram Key
Front End
Back End
MongoDB
Front End
Back End API's
MongoDB
User Applies for Membership using the Application Form
User Navigates to the Membership Directory
User Submits Post for the Post Board
User Navigates to the Post Board
Stripe API for Payment
Payment Approved
Yes
API Constructs JSON Object that contains User Input
API to Transfer Data Between Collections
API to Retrieve Data from MongoDB
Members to Display
ApplicationForm Database
MembershipDirectory Database
PostBoard Database
Members to Display
SubmittedApplications Collection
Yes
ActiveMembers Collection
SubmittedPosts Collection
User Accepted
Post Approved
Yes
Yes
ApprovedApplications Collection
ApprovedPosts Collection

Figure 1: System Design of BWMDN Website

Testing Procedure

Test Different Components

- Front End:
    - For the frontend, the component that needs to be tested is the navbar, the footer and the application form. The navbar is on the top of the screen and the footer on the bottom of the screen. The application form is a separate page and can be accessed by clicking "Join" on the navbar. To test the navbar and footer, one will click on each item within each to make sure it takes the user to the correct webpage.

1) Run react.ts code and open localhost.
2) Click through the various pages, and scroll on the home page.
3) Show that all the pages open, show design, show navbar, and show footer.
4) Show Application Form
5) Fill out dummy data in application form and submit
    a) Show that is the submit is successful the form moves to a success page
    b) if the submit is the successful, an error message shows
    c) show that if the user goes back to a different step of the application, that their data is all saved
    d) if a required data piece is missing have an error message (tells user to complete it)
6) Show submission in admin portal

- Back End:
    1) Since FastAPI script is hosted show the docs
    2) Show the different endpoints created, and explain each endpoint
    3) Show the approval process by submitting the application to approval DB.
    4) Show stripe test payment.

Measurable Criteria

The criteria for a successful test is as shown:

For Front End

1) All buttons must be clickable, and must take the user to the correct page.
2) All objects must appear correctly, the user should not notice any significant bugs on the home page.
3) Application Form must be fully editable with all fields.
4) Application form should be able to successfully submit and show an error message if not.
5) Admin portal must show all current users in applicants DB.

For Back End

1) A good POST request must result in a 200 OK response from API
2) The json object should appear in the mongodb database after successful request.

3) A bad POST request with either an empty request body or a missing field should result in "422 Unprocessable Entity", and an error message stating which field is missing should be returned.
4) Upon sending a bad POST request, the data should not go into the database.
5) A good GET request should pull all of the current applicants data from the database and return that data.
6) All POST or GET requests should complete in under 1 second.
7) After approving the applicant, send the applicant an email with a stripe link.
8) After paying through stripe, the link shows that the applicant is approved.

Score Sheet

FRONT END SCORE SHEET

Navbar Option	Webpage	Display	Page Open?(Y/N)
BWMDN	http://localhost:3000/	Home Page	Y
About	http://localhost:3000/about	Coming Soon	Y
Membership	http://localhost:3000/membership	Coming Soon	Y
Resources	http://localhost:3000/resources	Coming Soon	Y
News	http://localhost:3000/news	Coming Soon	Y
Sign In	http://localhost:3000/signin	Coming Soon	Y
Join	http://localhost:3000/join	Membership Form	Y

Footer Option	Webpage	Display	Page Open?(Y/N)
Home	http://localhost:3000/	Home Page	Y
About	http://localhost:3000/about	Coming Soon	Y
Membership	http://localhost:3000/membership	Coming Soon	Y
Resources	http://localhost:3000/resources	Coming Soon	Y
News	http://localhost:3000/news	Coming Soon	Y
Contact Us	http://localhost:3000/	Home Page	Y
Follow Us	http://localhost:3000/	Home Page	Y

Application Form	(Y/N)
Able to fill in form	Y
able to go back and see previous filled out data	Y
able to successful submit	Y
if did not fill out all requirements, show error message	Y

Admin Portal	Expected Response	(Y/N)
Display table of applicants	Matches list of applicants in database	Y
Click on applicant and display specific applicant data	Matches data for applicant in database	Y

BACKEND SCORE SHEET

Request/Action	**Expected Response**	**Achieved Expected Response (Y/N)**
POST Request with Good JSON Object	200 OK Data is added to database Applicant info sent is shown in Postman Response Body	Y
POST Request with Bad JSON Object (either empty or missing a field)	422 Unprocessable Entity Data is not sent to database	Y
GET Request Returns all data from SubmittedApplications collection	200 OK Data is shown in Postman response body	Y
All Requests Fulfilled in under one second	All requests should complete in under 1 second as shown in Postman.	Y
POST Request for applicant approval	200 OK Applicant is moved to approved DB. Email is sent to applicant with Stripe Link for payment.	Y

The initial testing for our frontend and backend was broken down into test cases to ensure feature functionalities. By breaking down each into subsections and smaller tests, we can test the functionality of the website to prepare the MVP for deployment. In the front end, we focused on running all the active components from the directory and testing the links to each subpage. During our testing, we clicked on each of the tabs on the navbar, such as Join Us, Directory, Resume Bank, Headshot, Find an Expert, coaching, etc. Each of these link to a custom, easy to read URL that links to the correct page from the host. All front-end tests were successfully completed. The Backend initial prototype test was focused on testing POST Request with JSON objects.

We attempted to push entered information for the membership application into a data object and return OK or Bad Request depending on the scenario. The information is received by our MongoDB database and the information is presented as a JSON object. In addition, GET request can pull data from the database, which will be displayed using our front-end to show the user’s profile data. Additionally, through our APIs we were able to successfully demonstrate the entire process of an applicant being approved. Our test cases for the Membership directory were successful and all requests were completed in under 1s. As presented in the scoresheet, it allows us to follow a set of test cases to ensure end-to-end functionality testing.

The second prototype test was an important milestone in the development of the website, as it allowed the team to further refine and test the developing functionality of the website. With the initial prototype test providing insights into the presentation, organization, and processing of information, the second prototype test focused on validating the website's more advanced features and functionality.

The front-end testing focused on the user's flow through the website and ensuring that each page's design was intuitive and user-friendly. This testing involved a series of use cases, such as navigating through different pages and interacting with different components, to ensure that the user experience was seamless and straightforward. On the back-end, the team tested more complex features, such as user authentication and authorization, search functionality, and API integration. The testing of these features was crucial to ensure that the website's core functionality was working as intended and that the website was secure and scalable.

Overall, the second prototype test was a success, and the insights gained from this testing phase were used to further refine and improve the website's design and functionality. The team was able to identify and fix any issues and ensure that the website was ready for the next phase of development.