MyHelix App

Table of Contents

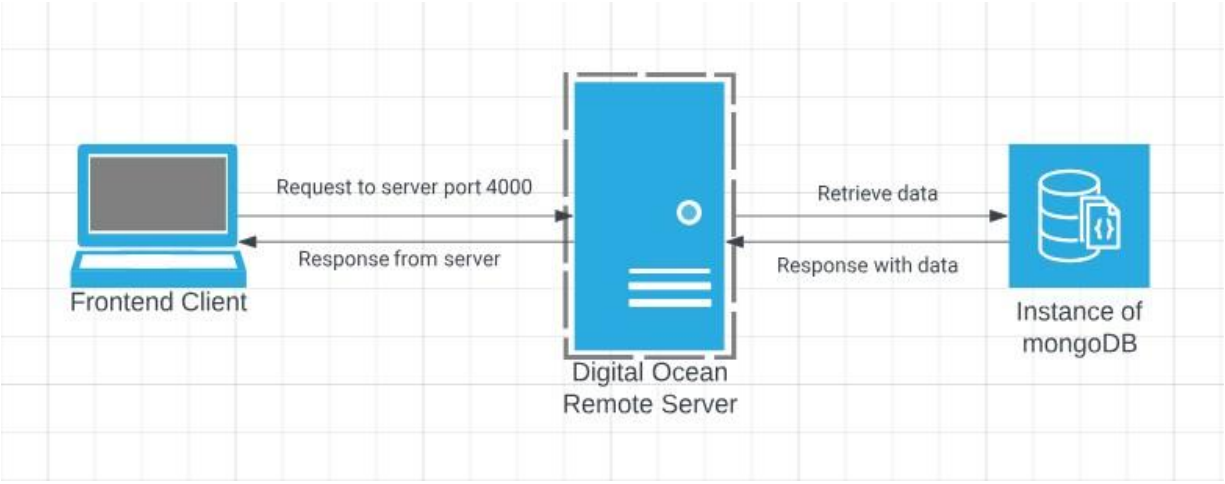
Introduction

Preface (Skippable):

Welcome to the documentation for My Helix, a social media application that has been in development under the stewardship of our key stakeholder, Patrick Reed. This documentation serves as a comprehensive guide to the project, chronicling our progress and detailing the intricacies of the flexible backend we've diligently crafted throughout the Fall 2023 semester. Our dedicated team encountered numerous challenges with the original product/program, which ultimately led to a consensual decision between group and stakeholder to restart the entire backend application. In the spirit of transparency and continuous improvement, this documentation aims to shed light on the issues we've faced, their root causes, and the effective solutions we've implemented. Additionally, we have aimed to provide more transparency in the code itself with detailed explanation throughout this document, and through in-line comments provided in each Javascript file. We hope that all the information laid out here and in the code will provide you the tools to easily extend the applications functionality as per Patrick's request.

Architecture (Skippable):

The main architecture is outlined in the figure below. In essence, this is a very typical backend application, where the client communicates to the remote server via API calls. The backend waits to retrieve data from mongoDB, and returns a response back to the client.



Key Configurations (Optional):

This section is optional since Patrick or Alex will provide you with the necessary instructions and keys to connect to the remote server via local machine. However, just in case (as Slack tends to delete messages after 30 days), here are the instructions to ssh into the remote server:

1. Store provided ssh keys in a particular directory (i.e ~/.ssh)
2. In your terminal, run the following command:
   ssh -i ~/.ssh/helx root@164.92.92.179
   NOTE: Make sure to replace ~/.ssh/helx with whatever directory you have saved your ssh key in

Once connected, you may see two directories. Do not fret, as one of the folders was the previous directory responsible for running the backend. You will be continuing work on the directory named myhelixbackend.

Droplet (Optional):

As mentioned, the backend application is expected to run on a remote server, hosted on a DigitalOcean droplet. The server runs on Ubuntu (a Debian-based Linux distribution), and already has Docker installed. While we will not go over specifics of ubuntu terminal commands here, we recommend that you touch up on some syntax just for easier understanding of the utility.

Getting Started

Setting Up Dev Environment:

We have provided two different ways to continue development on your local machines. One is through a Docker container, and the other is running the entirety of the app locally. Both methods are documented properly below. As for initial instructions, please clone the backend repository to any directory on your local machine. The github repository will be provided to you by Patrick or Alex.

Docker:

If you are unfamiliar with Docker, fear not as you will not need to know the intricacies of Docker and its functionalities. Just follow the instructions here, and you should be able to run a Docker container for the backend application without having to worry about installing Node and a package manager on your local machine.

1. Install Docker Desktop from here: https://www.docker.com/products/docker-desktop/
2. Follow the installation instructions and run it upon installation
3. Assuming that you have cloned the backend repository, navigate to the directory on a terminal window
4. Once in the project folder, run the following command:
   `docker build -t [your-node-app] .`
   where [your-node-app] is whatever name you prefer to call the application.
   IMPORTANT NOTE: Do NOT remove the period at the end of this command
5. Wait for the container to finish building. Once it has been completed, you can run the container with this command:
   `docker run [your-node-app]`
   Once again [your-node-app] is the name that you have given your app.

Now that you have a container for the application, at any time you want to run the application again, just navigate to the project folder, and run the above 'docker run' command.

- Another way to run the application without using the terminal is to head to Docker Desktop, and do the following:

Head to Containers → Toggle off "Only show running containers"→ Click on the container with your app → Click the play button on the top right

Local Environment:

While it may certainly be easier to run the application in a Docker container as described above, the application can be run locally. We highly recommend installing npm (this was the node package manager that was used by previous groups and us) as well as the most recent version of Node.js (at the time of writing these instructions was 18.18.0).

1. First install Node.js and npm (Please refer to online sources in case you have trouble installing the two)

2. Once you have installed both the runtime environment and the package manager, open up a terminal window and navigate to the project (assuming you've cloned the repo)

3. Run the following command (this should install all packages and dependencies for the backend application):

   ```
   npm install
   ```

4. You can then start running the backend application on localhost with the following command:

   ```
   npm start
   ```

MongoDB

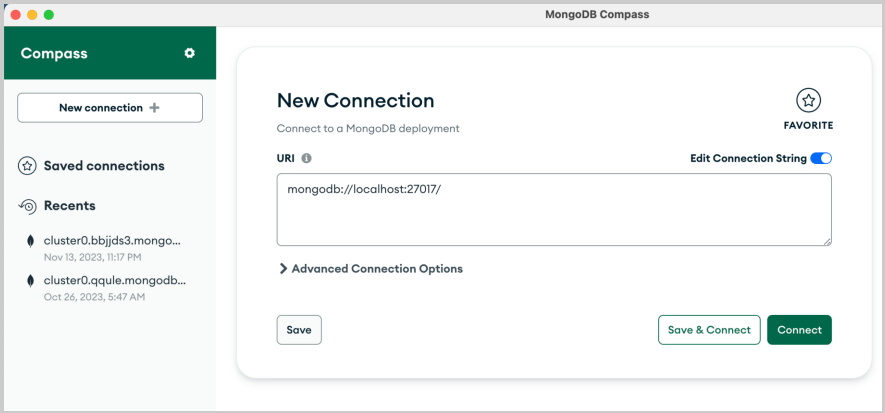
Setting Up:

As you may have seen with the image above, the backend architecture relies on MongoDB as the database instance, forging a robust foundation for efficient and scalable data management. Here we'll provide quick instructions to set up the mongoDB compass, and a couple of words and suggestions before starting development on the application.

1. Install the mongoDB compass here: https://www.mongodb.com/products/tools/compass

2. Once installed create an account or login

3. When you are logged in, you will come across this page (If not, just restart the compass)



4. Take the database url: mongodb+srv://cindyxie:5NFypnqPyADrUjTi@cluster0.bbjjds3.mongodb.net/myhelix?retryWrites=true&w=majority and paste it into the URI section as shown above.

5. Hit the Connect button, and it should redirect you to the database instance

IMPORTANT NOTE / SUGGESTION:

Originally, the creator of this database instance was responsible for a multitude of features on the backend. However, due to an unforeseen emergency, this member was unable to retain any form of contact, and thus we ended up losing access to the database. Although you can still make API calls, we HIGHLY recommend that you create a new database instance so as to retain full control in the development process.

Once you have created a new database instance, you can copy the URL, go to the app.js file in the backend repository, and change the value of the dbURI variable to your newly formed database URL.

Versioning Issues

Runtime Environment Issues:

Node.js is the javascript runtime environment that we use for the backend application. Part of the reason why we were required to start the entire project over was that the Node.js version between the frontend and backend were mismatched. Hence, we've created an extra guide to help prevent versioning issues getting in the way of testing/development.

Runtime Env Fixes (READ ONLY IF TESTING ON LOCAL):

At the time of writing these instructions/commands, the most recent & stable version of Node.js is node-v18.18.0

(Dependent on which package manager you use [We as a group used npm])
To update node version (MAC VenturaOS):

```
sudo npm cache clean -f
sudo npm install -g n
sudo n lts
```

To update node version (Windows OS):

```
npm install -g n
n latest
```

To update node version (LinuxOS):

```
nvm install node --reinstall-packages-from=node
sudo rm -rf node_modules
sudo rm package-lock.json
npm i
yarn add expo
npx expo install --fix
```

To update node version (Debian/Ubuntu):

```
sudo apt-get install nodejs
```

Afterwards, on all systems, run (command should be universal across all systems if you have nvm):

```
nvm install **.**.**
```

Note: the stars will be the version installed when you run the commands to install the most updated LTS version of node js.

Runtime Env Fixes (READ ONLY IF TESTING ON DOCKER):

Go to the Dockerfile in the backend repository
Update the first line:

`FROM node:18-alpine` → `FROM node:[new-version-of-node]`

Package/Dependency Issues:

While this is an app restart, there is a possibility for the current stakeholders or future groups to come across versioning issues with different packages. This is one of the various issues that contributed to our motivation behind restarting the project and modifying its infrastructure. We can not guarantee that the packages used will have continued maintenance, which may or may not affect its compatibility with other up-to-date packages. That being said, here is a set of instructions to make sure that the project's package.json, for both frontend and backend are kept up to date (We also recommend running the series of commands when cloning into the project repositories):

- For Windows/MacOs/Linux, run the following command in the terminal (this applies if you are running a container as well):
  `npm install npm-check-updates`
  Note: Make sure that this command is run inside the backend repository
- Navigate to the directory's package.json file
- To check any out-of-date dependencies, run:
  `ncu`
- To update all out-of-date dependencies to the most recent version:
  `ncu -u`

Application Dependencies/Packages

Summary (Skippable):

This is really just an FYI, in case you are looking at some of these packages and wondering why we need them in the first place:

bcrypt (^5.1.1): A library for hashing passwords, enhancing security by securely storing sensitive information. For the backend application, we use modules from this package to hash our user passwords and store them into our database.

cookie-parser (^1.4.6): Middleware for Express that simplifies handling of HTTP cookies, facilitating cookie parsing and manipulation. More specifically, we use it to provide a cookie to keep our users logged in for a period of time

cors (^2.8.5): Middleware for handling Cross-Origin Resource Sharing (CORS) in Express, enabling secure communication between different origins. Currently not being used in the application as of yet.

dotenv (^16.3.1): A zero-dependency module that loads environment variables from a .env file, easing configuration management in development. Although we aren't necessarily using environment variables yet, we expect you will need this package in the future, so we left it in here.

express (^4.18.2): A fast, unopinionated, minimalist web framework for Node.js, simplifying the development of robust web applications and APIs. Essential for being able to run the server.

jsonwebtoken (^9.0.2): Implementation of JSON Web Tokens (JWT) for secure communication between parties, commonly used for user authentication and authorization. We use this module to keep our user doc tied to our browser for quick updates on user information

mongoose (^7.6.0): A MongoDB object modeling tool for Node.js, providing a straightforward schema-based solution for interacting with MongoDB databases.

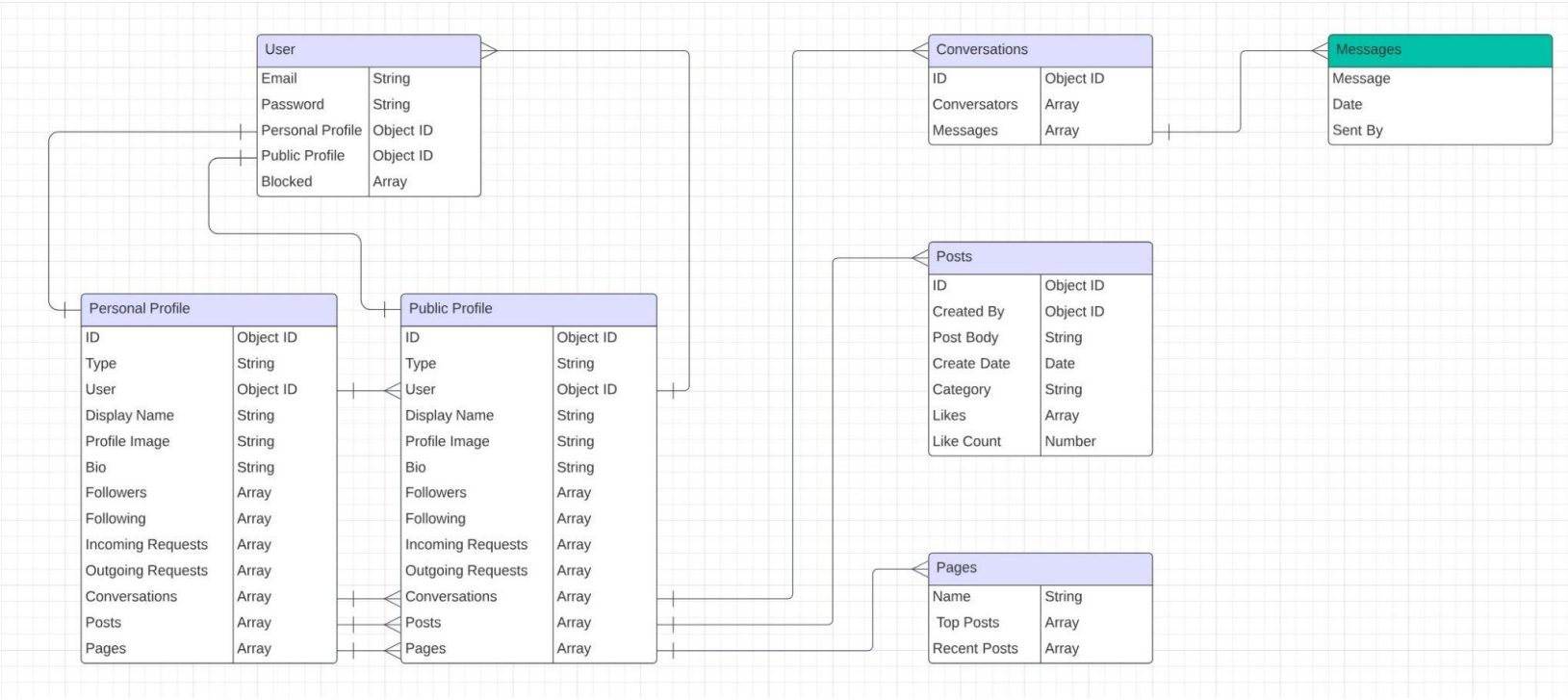
node (^18.18.0): The core runtime environment for executing JavaScript code on the server-side.

swagger: This will be explained in more detail in the Swagger section below

Schema

Summary (Skippable):

Much of the information regarding the individual schemas are heavily commented on in the code itself, and is further elaborated on in the Swagger Documentation. However, if you still prefer a visual representation of the current schemas thus far, refer to the figure below.



Additional Notes (Suggestion):

The way we structured the Incoming Requests, and the Outgoing Requests, are just an array of objects that include a Number and an Object ID to a profile. Our thought process was to have a number represent a specific request (i.e 0 == Friend Request, 1== Message Request). However, because we have not yet implemented the functionality for this, it is ultimately up to future groups to redefine this to their needs.

There are various other Objects in our schemas that have yet to be provided with specific functionality, but we have provided a foundation to a more concise database relation that should make it easier for you to implement and modify certain components to your needs.

Swagger Documentation

Summary: