2023

HYPE: SYSTEM DESIGN DOCUMENT

BY GIT-HYPE TEAM

ALEX RUSSELL, GISELLE WANG, JAWAD ARSHAD, PANTYSH GHURBURRUN, DENISE OU, YUXIN CHEN, SANJAY MYLANATHAN

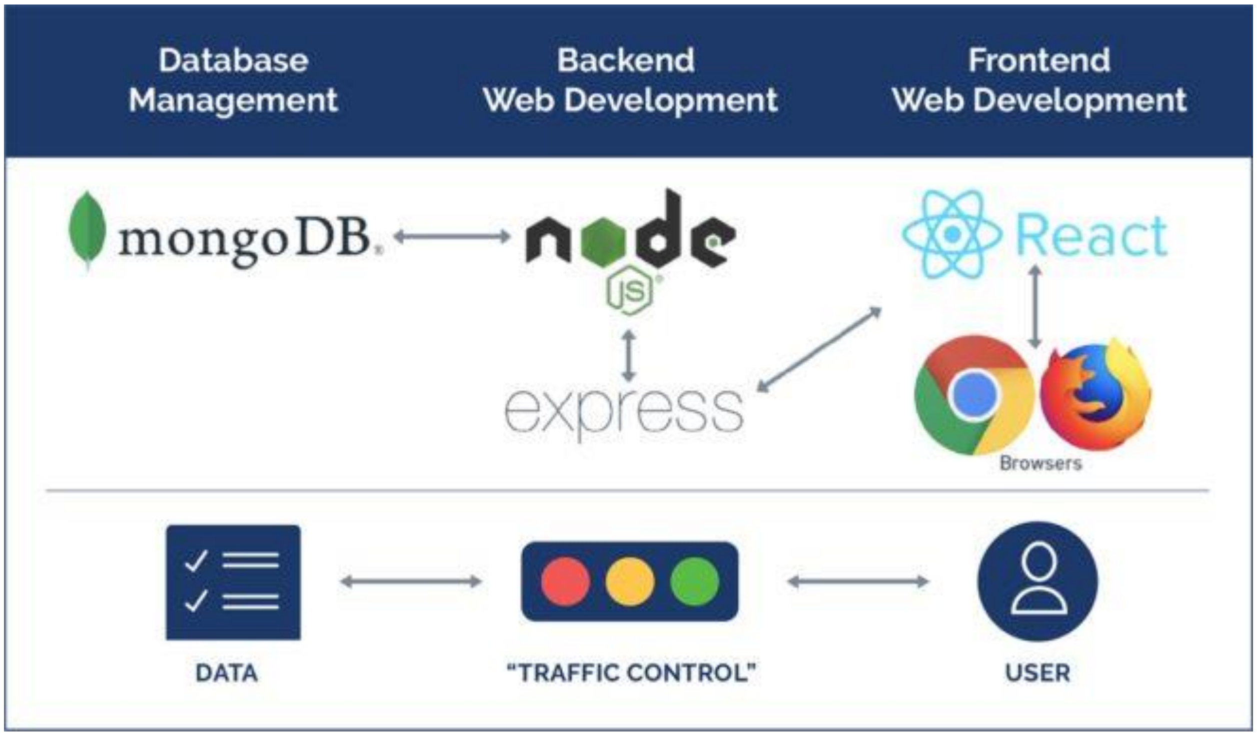
CSCC01 | Pankaj Agrawal

INTRODUCTION:

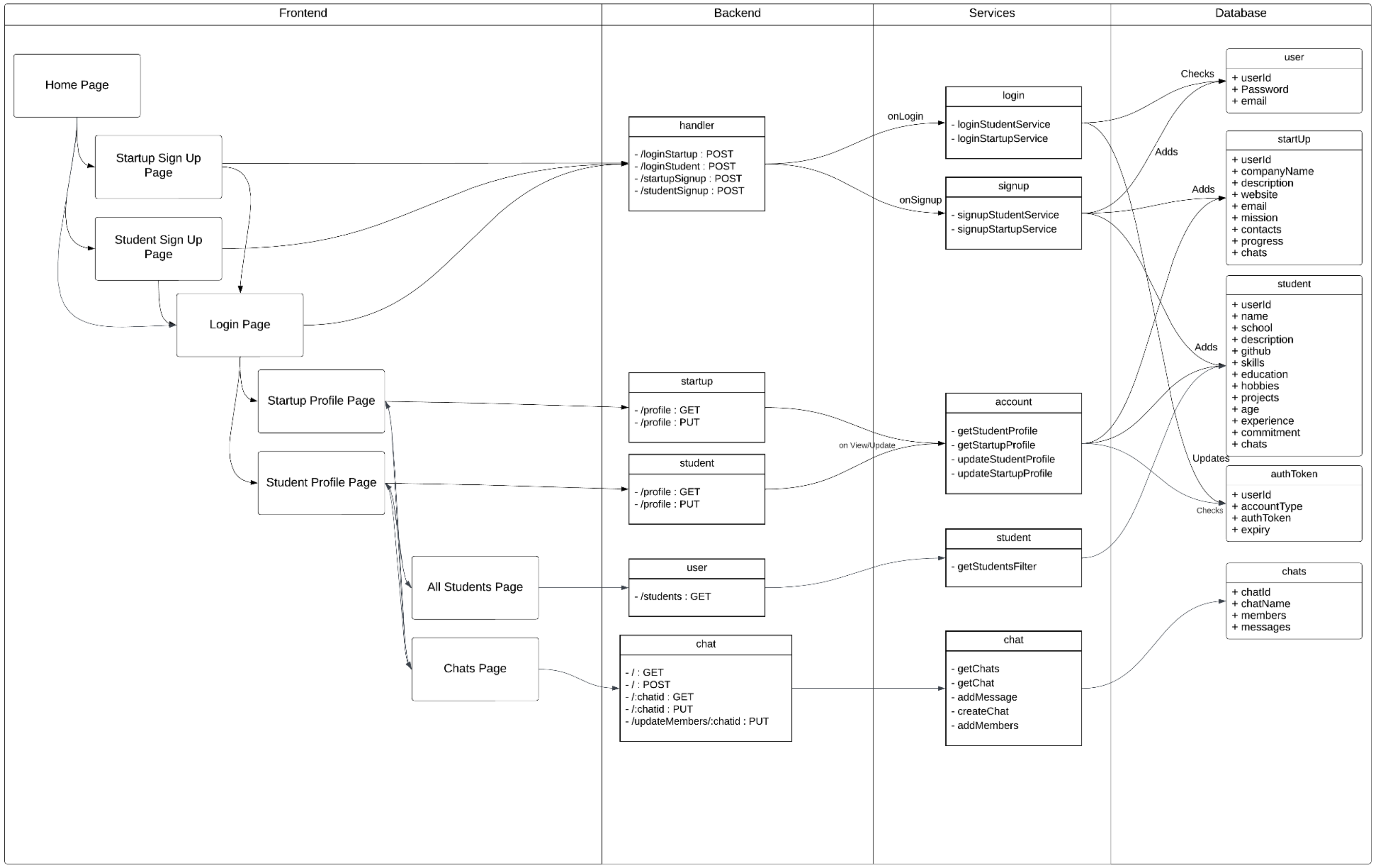
The app being built is called HYPE, it will initially start as a web application, that will focus on connecting students to startups where they can intern. The main means that students will be able to connect with these start-ups are via internship opportunity posts as well as possible competitions that the startups can hold to help filter candidates. Students will also have a profile where they can share a bit about themselves, through text but also with a short video resume. Similarly, startups will have a similar profile where they can share more about what they do, they will also have a video on their profile. For the competition, the students will be able to team up with each other to create innovative applications to submit allowing startups to find teams of people that work well together.

SYSTEM ARCHITECTURE:

For the application, the frontend is built as a React app, and the backend is a nodeJS app that runs on expressJS to create a server. The Database will be a MongoDB database hosted on MongoDB Atlas cloud servers. The app will be a three-tier architecture, where the frontend will call the backend for all data requests and the backend will access and store things in the database. The following is a diagram that shows the architecture in more detail. Subject to change as more features are added.



[Academy, K. (2021). MERN Stack Diagram. Kenzie Academy. Retrieved June 16, 2023, from https://kenzie.snhu.edu/blog/what-is-mern-stack/.]


Frontend
Backend
Services
Database
Home Page
Startup Sign Up Page
Student Sign Up Page
Login Page
Startup Profile Page
Student Profile Page
All Students Page
Chats Page
handler
- /loginStartup : POST
- /loginStudent : POST
- /startupSignup : POST
- /studentSignup : POST
startup
- /profile : GET
- /profile : PUT
student
- /profile : GET
- /profile : PUT
user
- /students : GET
chat
- / : GET
- / : POST
- /:chatid : GET
- /:chatid : PUT
- /updateMembers/:chatid : PUT
onLogin
onSignup
on View/Update
login
- loginStudentService
- loginStartupService
signup
- signupStudentService
- signupStartupService
account
- getStudentProfile
- getStartupProfile
- updateStudentProfile
- updateStartupProfile
student
- getStudentsFilter
chat
- getChats
- getChat
- addMessage
- createChat
- addMembers
Checks
Adds
Adds
Adds
Updates
Checks
user
+ userId
+ Password
+ email
startUp
+ userId
+ companyName
+ description
+ website
+ email
+ mission
+ contacts
+ progress
+ chats
student
+ userId
+ name
+ school
+ description
+ github
+ skills
+ education
+ hobbies
+ projects
+ age
+ experience
+ commitment
+ chats
authToken
+ userId
+ accountType
+ authToken
+ expiry
chats
+ chatId
+ chatName
+ members
+ messages

FRONTEND DESIGN:

The organization of the frontend code is the following:

- The components folder will contain all the code for each component used in the application.
- The context folder will store all the code for any context in the application.
- The pages folder will contain all the code for all the assessable pages in the application, and it will be built by using components.
- The routes folder will store any route specific code, i.e., like code to verify is the route assess requires authentication and authorization.
- The sagas folder will hold code to fetch different endpoints.

- The whole app is enclosed in a global context to store states that are required in the whole app, like User data etc.
- The whole app is enclosed in a saga context to give access to all the sagas from any part of the app.

- The following libraries were added to help development:
  - styled-components

BACKEND DESIGN:

The organization of the backend code is the following:

- The models folder will contain all different models for the database schema.
- The routes folder will store all the code for each route, the routes will be relative to their location in the folder.
- The services folder will store all the code that will interact with the actual database.
- The server.js file is the main file to launch the server.

- The following libraries were added to help development:
  - bcryptjs
  - body-parser
  - cors
  - dotenv
  - express
  - mongodb
  - mongoose
  - uuid

DB DESIGN:

The following is the information stored in the data base and its schemas:

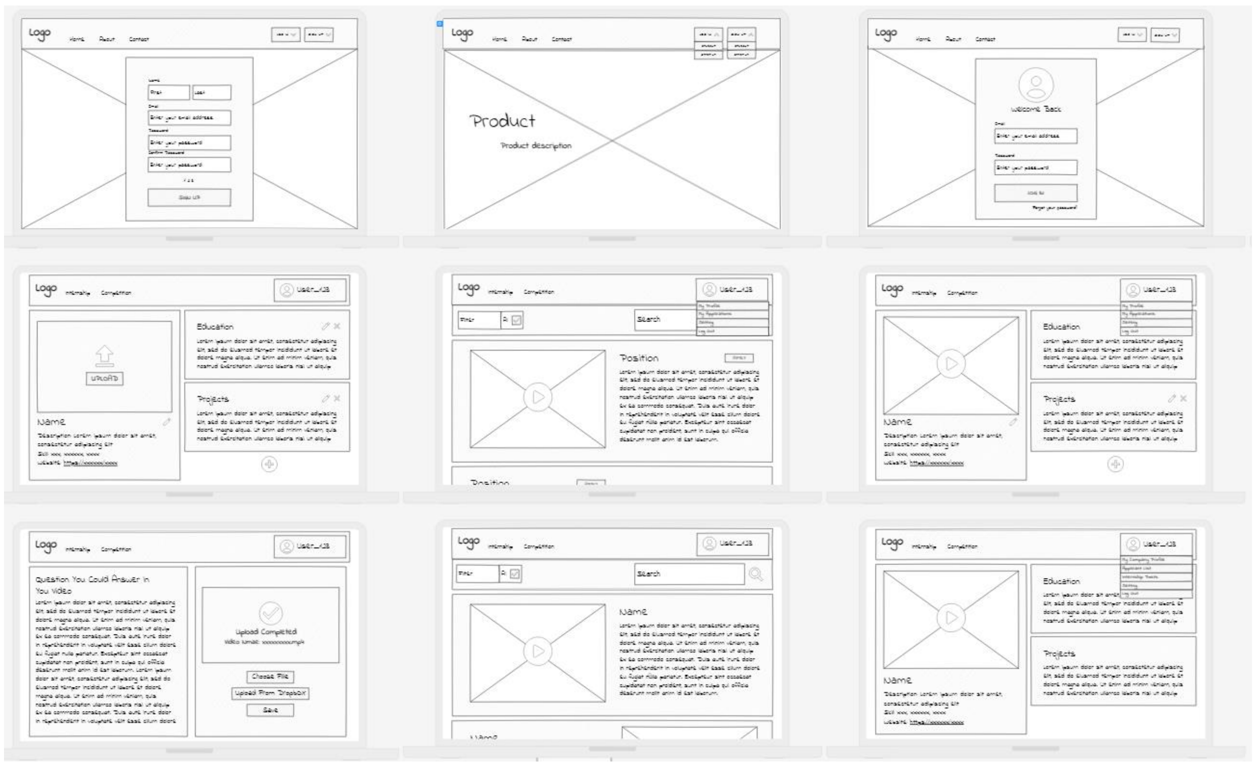
Collection Name	Schema
users	email password
students	user name school description github skills education hobbies projects
startups	user companyname description website email mission contacts progress
authTokens	user accounttype authtoken expiry

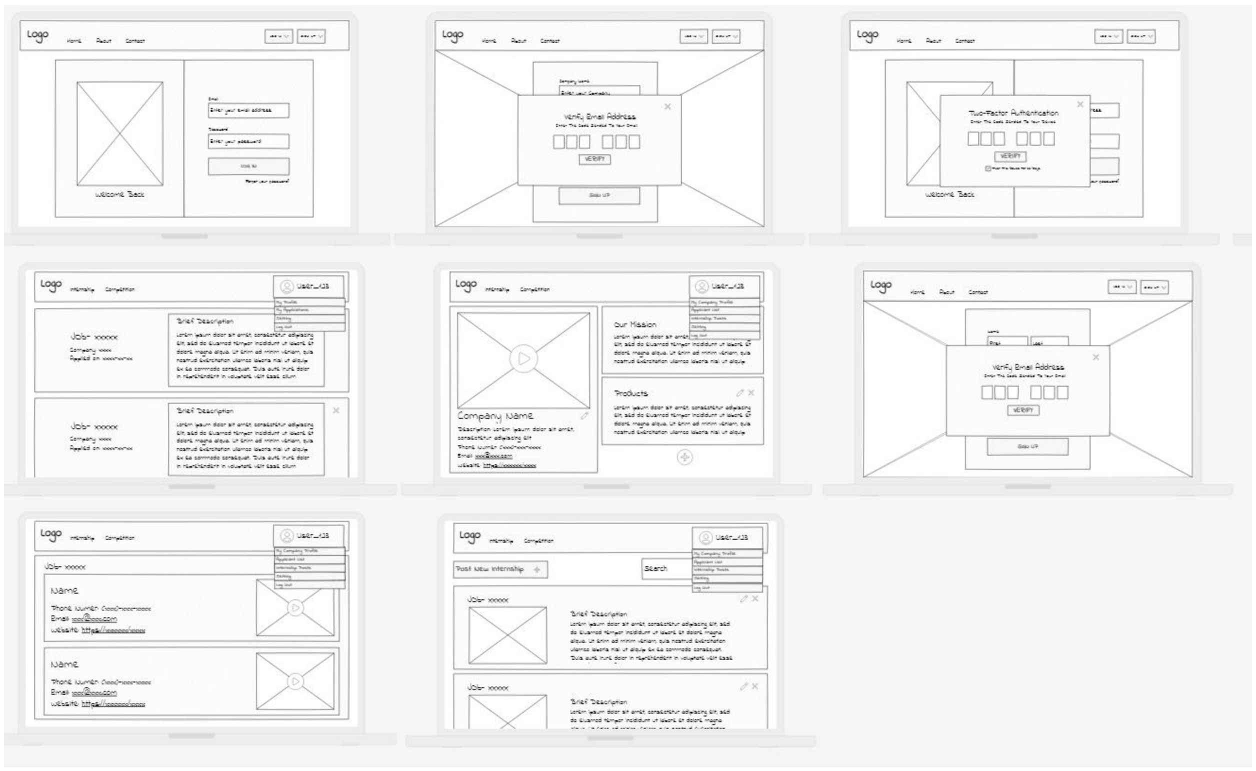
ERROR AND EXCEPTION HANDLEING:

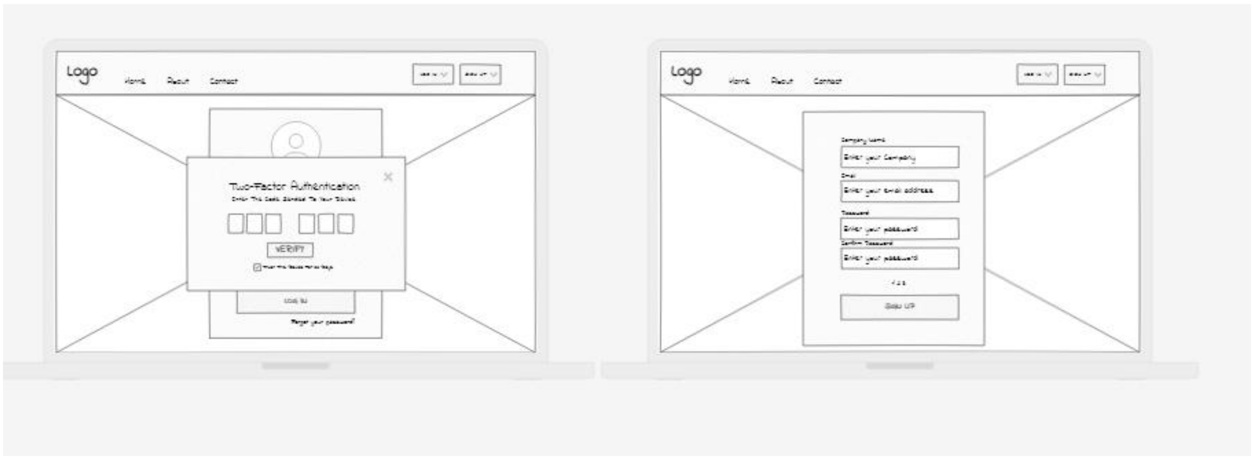
All error should be handled gracefully. This will be done by adhering to the following strategies:

- All API calls, and Database calls should be wrapped in a try catch block to ensure proper error handling.
- If the error is a frontend error, it should be handled in the react app and show a informative error message to the user. No pop-up alerts should be used. The error message should also be logged as a console error message to help make debugging easier in the future.
- If the error happens in the backend, it should be communicated to the frontend in an informative way, and it should then be handled the same way as mentioned above.
- If the error is a database error it should be handled on a case-by-case basis by the backend and if necessary, communicated to the frontend, and handled as stated above.
- Exceptions should be handled in the same way.
- ALL SPECIAL CASE SHOULD BE DOCUMENTED BELOW:
    - NONE AT THE MOMENT

UI/UX WIREFRAME:







SUPPORTED PLATFORMS:

Name	Version
Chrome	114+
Firefox	114+
Edge	114+
Safari	16.5+