Time: Aug 10 @ 8-8:25pm

Demo - max 5 mins

Problem

- Students
    - Want hands-on experience in the field & explore different career options
    - Want to collaborate with like-minded individuals & expand their network
- Startups
    - Want innovative, driven, capable and creative people to help realize their ambitions
    - Tight budget & small team with heavy workload
- LinkedIn & co-op positions
    - Unable to help form this personalized connection between students & startups, and provide opportunities to gain experience that is not restricted to certain qualifications, cities or regions

Hype

- Web application that connects students with startups
- Platform for students & startups to:
    - Communicate
    - Collaborate
    - Innovate
- Opportunities:
    - Competitions
    - Internships

Target users

- Students
    - Undergraduate (including recent graduates)
    - Innovative & highly motivated
    - Team player
    - Curiosity in exploring different career options & desire to learn
- Startups
    - Good team dynamics (collaboration focused)
    - Flexible structure
    - Passion for innovation & technological advancement
    - Has experienced individuals in multiple different fields

- Hype attempts to connect innovative students with each other and with startups who need them
- while internships can be eventual goals, and we do have a place for them in the app, they are not the first priority
    - competitions offer a more results-oriented way to showcase individual students' aptitude and ingenuity

- Our app features accounts for both students and startups,
    - With the ability to save and view information that shows what they're each capable of and what their goals are
    - With the ability for students to search for, filter through, and communicate with each other, alone and in groups
    - With the ability for startups to create competitions which students can view and sign up for

- Together this helps facilitate making teams of innovative creators and connecting them with the ideas they can make real

Process Discussion - max 3 mins

1. Describe your team's workflow
    - Including:
        - How did you communicate

          We communicated in multiple ways. We communicated in-person, through meetings on Tuesdays after the lecture, and meetings on Saturdays at either 11 AM or 1 PM. Though, after the first few meetings on Saturdays we included an online option for the meeting, wherein group members could meet in person, or attend an online discord call which would be started by someone who attended in person.
          Outside of these meetings, we mainly used Discord for general communication, while reserving Slack for important project related details.We used discord for general communication due to it being a very popular communication method among students, and all of us were familiar with it, and would respond quicker with Discord compared to other apps. In addition some members don't install Slack on their phones, thus making it more inconvenient for general group discussion and response times. We did use Slack for more official project details though, we discussed important information, meeting notes, and standups on Slack. .
          We also have a Whatsapp group which included a founder, Veb. which we would use to discuss developments regarding the app with him. He would also join in on some Saturday meetings, either in person, or through a zoom call.

        - How did you use Git/GitHub

          In general we created branches for each task/Jira story that we assigned ourselves with the branch name starting with the story ID. For example, Jira Story C01-50 would have a branch on github starting with C01-50. When we finish a story, we would make a pull request on that branch to dev, wait for another member to approve the pull request and then merge

to dev. Starting sprint 3, we made a rule that we would only mark a Jira ticket as done after the related branch was merged to dev. We would branch off dev to implement new features, and push back to dev when those features were completed. When the end of the sprint comes we merge dev to main, for submission for that sprint.

- How did you divide tasks

  We divided tasks in two different ways between sprint 1, and sprints 2-4. During sprint 1, we did not assign story points to any of the Jira stories. During this sprint, after creating the Jira stories and selecting 11 stories for the sprint, we simply assigned ourselves to at least 1 story each, with some people choosing more unassigned stories afterwards.

  Starting from sprint 2, we assigned story points to the Jira stories, usually having around 40 story points worth of stories per sprint (though we managed to move up to 48 story points during sprint 4), and each member tried to assign themselves to near the amount of average story points per member to reach the total which in these cases we simply did a division of total story points by 7 members, which is usually around 40/7 equals rounded up to 6 story points. So each member would try to take on around 6 story points worth of tasks, and so on depending on how many total story points we start a sprint with. Of course, this was just a general way of how we split tasks, there were some situations with exceptions where people took on more than the average amount, or less since other stories were already assigned. Such as in the case where a story was incomplete and thus moved to the next sprint, in which case the person assigned to that task sometimes takes on more than the average amount of story points when continuing that task alongside others. In general we used story points to allow tasks to be divided more fairly.

- Audience should be able to:
  - Understand how you worked together as a team

    This is how we communicated, used github, and divided tasks. In addition, when it comes to other tasks outside of the Jira stories, such as deliverables for each sprint, we simply discussed them either during the Saturday meetings or on Discord and people volunteered/assigned themselves to do them, trying to aim for everyone to participate or volunteer for some deliverable. Such as when it came to this presentation, we discussed the requirements, and how to split this task fairly, and after deciding the available roles we each volunteered for a role.

2. Highlight significant decisions (good/bad) that you've made along the way & explain rationale behind them.

   Good Decisions:
   - Creating Figma prototypes of what we want to make before starting implementation. This allowed us to save a lot of time when implementing with the prototypes as a guideline, while having a consistent theme rather than each developer imagining their own design for each story they work on, which could have led to even related stories looking dramatically different in the end.
   - Making Jira stories smaller and more fine-grained, allowing for better division of tasks and more manageable stories. We did this after learning from the bad decision of tasks that were too large in sprint 1. This allowed us to have higher rates of completed stories out of committed stories per sprint, and workloads were more manageable for team members.
   - We changed the definition of done for Jira stories to include merging our code to dev, and integrating it with the other code there. This improved our situation of having to merge many feature branches, which may conflict, all around the time of the sprint ending. It also allowed for people working on stories that need to interact to integrate it on time.

   Bad Decisions:
   - During the first sprint we made tasks too big, they were a lot of work for individual team member's to do, and some tasks also depended on other tasks in the same sprint to be completed first. This lead to some incomplete tasks.

Technical Discussion - max 3 mins

Describe 1 - 3 technical items that came up while working on your project

- Technical item can be:
  - An interesting bug
  - Design change
  - Lesson you learned from using a certain technology/framework
  - Etc.
- Be specific & include useful artifacts (code snippets, screenshots, etc.)

- Technical Item: Adding chatId in create chat API POST Response
  - Issue: the absence of chatId in the response of the Create Chat POST request.
    - Initially, the response provided only a message indicating the success of the chat creation process but lacked the necessary "chatId" field.
    - This omission hindered the smooth integration of the chat creation process with other functionalities, such as displaying chat messages that are sent through a popup window.

        - chatId is a mandatory body in update chat message put request so that we could update the corresponding message array in the database.
    - Solution:
        - Backend: chatId was added to the response of the POST request.
        - Frontend:
            - Use createChatUpdate() to save the chatId from the API response in the saga file.
            - Add the function useCreateChatContext() to get the chatId in the popup window frontend javascript file, so that when the response is returned, we could use this chatId in the update chat message request.

Initially:

```
import { POST, chatEndpoint } from "./endpoints"
import { fetchCall } from "./fetchCalls"


export const createChatSaga = async (chatname, othersUserId, authToken, onSuccess, onFailure) => {
    const response = await fetchCall(chatEndpoint, POST, { chatname, othersUserId, authToken });

    const data = await response.json();

    if (data.success) {
        onSuccess(data);
    } else {
        onFailure(data);
    }
}
```

```

 router.post('/', async (req, res) => {
    try {
        const userId = req.userId;
        const { chatname, othersUserId } = req.body;
        const {success, message} = await createChat(chatname, userId, othersUserId);


        if (success) {
            res.status(200).json({
                success: true,
                message: message,
            });
        } else {
            res.status(400).json({
                success: false,
                message: message,
            });
        }

    } catch (err) {
        console.log(err);
        res.status(500).json({
            success: false,
            message: "Internal server error",
        });
    }
 });

```

After modification:

frontend > src > sagas > JS createChatSaga.js > ...

```
1  import { POST, chatEndpoint } from "./endpoints"
2  import { fetchCall } from "./fetchCalls"
3
4
5  export const createChatSaga = async (chatname, othersUserId, authToken, createChatUpdate, onSuccess, onFailure) => {
6      const response = await fetchCall(chatEndpoint, POST, { chatname, othersUserId, authToken });
7
8      const data = await response.json();
9
10     if (data.success) {
11         createChatUpdate(data.chatId);
12         onSuccess(data);
13     } else {
14         onFailure(data);
15     }
16 }
17
```

```
router.post('/', async (req, res) => {
    try {
        const userId = req.userId;
        const { chatname, othersUserId } = req.body;
        const {success, message, chatId} = await createChat(chatname, userId, othersUserId);

        if (success) {
            res.status(200).json({
                success: true,
                message: message,
                chatId: chatId,
            });
        } else {
            res.status(400).json({
                success: false,
                message: message,
            });
        }

    } catch (err) {
        console.log(err);
        res.status(500).json({
            success: false,
            message: "Internal server error",
        });
    }
});
```

- Technical Item: Store type of user account locally.
    - Issue: Navigation bar required an account type after login
    - Attempted solution: Assign types as props to Nav component
    - Challenge: Shared web pages between student and startup users; prop addition insufficient
    - Alternative solution: Use localStorage to share type data across pages
    - Rationale: User type data is non-sensitive and simple
    - Implementation: Added code after successful user login
    - Result: User type stored in localStorage for later use in Nav component

```
const loggedIn = useAuthTokenContext();
useEffect(() => {
    if (loggedIn) {
        if (props.isStudent) {
            const NavInfo = { storeType: "student"};
            localStorage.setItem("NavType", JSON.stringify(NavInfo));
            history.push("/student-profile"); //change to where you want to go after login
        } else {
            const NavInfo = { storeType: "startup"};
            localStorage.setItem("NavType", JSON.stringify(NavInfo));
            history.push("/startup-profile"); //change to where you want to go after login
        }
    }
}, [loggedIn]);
```

```
if (localStorage.getItem("NavType")) {
  type = JSON.parse(localStorage.getItem("NavType")).storeType;
}
```

Software Architecture - max 3 mins

1. Run us through the way your software is built
    - What are its main components
    - What are their responsibilities
    - How do they interact with one another
2. Tell us about the technologies & tools that you used
3. Mention interesting technical challenges you faced and/or interesting software techniques that you used

[Frontend]
Main Components:

- **Components Folder**: Houses individual UI components.
- **Context Folder**: Manages global state and shared data.
- **Pages Folder**: Represents the different views or screens users interact with.
- **Sagas Folder**: Centralizes our API calls, ensuring consistency and reusability.

Interactions:

- Each page is constructed using multiple components from the different folders.
- Pages access global data via contexts, like user authentication tokens.
- For data fetching, pages and components utilize sagas to interact with our backend.

Technologies:

- We've styled our UI using **styled-components**.
- API calls are made using **axios**.
- For rapid UI development, we've integrated **bootstrap** and **mui**.

Challenges/techniques:

- Achieving a responsive design was initially challenging, especially for mobile views.
- Managing global state using contexts required careful planning to avoid over-complication.

[Backend]
Main Components:

- **Models Folder**: Defines our data structure and interacts with the database.
- **Routes Folder**: Specifies the available API endpoints.
- **Services Folder**: Contains the core logic for each endpoint.
- **Utils Folder**: Offers utility functions that enhance code reusability and efficiency.

Interactions:

- Incoming requests are directed by routes to the appropriate service.
- Services, when needing data operations, interact with models.
- Utility functions from the Utils folder are used to help with various tasks.

Technologies:

- Our backend runs on **Express.js**, a framework for Node.js.
- For database interactions, we've employed **Mongoose** and the native **MongoDB driver**.
- During development, **Nodemon** was for auto-restarting our server on code changes.

Challenges/techniques:

- Large queries with useless data: As our user base expanded, we refined some of our database queries ensuring we only fetched necessary data.
- We implemented middleware in Express, for tasks like authentication and request parsing.

Individual Contribution - max 1 min per member

1. At end of your presentation, each member will briefly describe their main individual contributions and/or lessons learned

Alex:

- Setup sagas (for frontend / backend communication)

- Helped integrate some early solutions which ran into merge conflicts
- Allowed editing of profile page for both students and startups (minus profile image)
- Fixed formatting and display bugs

Pan:
- Parts of user logins and hashing passwords.
- Profile icons (editable and random default icons on creation)
- Student List with filtering

Katy:
- Home Page
- Navigation Bar
- Message display and chat delete

Denise:
- Design/layout - student profile, all students page
- Cue card versions of student profiles
- Customized scrollbar
- Search ability in messages sidebar
- Display different icons for direct messages vs. group chats

Giselle:
- Backend for startup user login and student user signup
- Popup window for sending a message
- Delete messages
- Competition signup
- Registered student list for competition

Jawad:
- Frontend for the login/sign up form
- Backend for some of the chats feature
- Frontend for the chat create/edit group
- Frontend for the competition and all competition page
- Backend for the competition and the all competition page

Sanjay:
- User logout functionality
- Backend for student user edit profile
- Backend for startup user edit profile
- Backend for startup user create competition
- Frontend for startup user create competition
- Design improvements, and error fixes

Questions - 5 to 10 mins

1. Will be asked questions on how you worked together as a team
2. Will be asked technical questions about your software

Script

Intro

<click>

Good evening everyone! My name is Denise and **<click>** today our **<click>** team Git Hype would like to present to you **<click>** Hype! A web application that provides a platform for both students and startups to communicate, collaborate, and innovate.
<click>

Demo
Denise:

- Problem
  <click>
  - Now we've all been there as students where we want to gain hands-on experience in a field that we're interested in, explore different career options, and collaborate with like-minded individuals, thus expanding our network.
  - On the other hand, startups usually consist of a small team who want to look for innovative & capable individuals to join their team. However, they face a tight budget and may not be able to hire a full time employee.
  - While LinkedIn and co-op positions provide experience for students and allows companies to recruit talent, they are unable to help form a personalized connection and provide opportunities that aren't restricted to certain qualifications or locations.
- Hype
  <click>
  <click>
  - This is where Hype comes in. On our web app, students are able to connect with startups and other students that have similar visions and goals. With the help of this interactive platform, users are able to communicate through messages, and collaborate and innovate through competitions. Current opportunities allow students to participate in competitions created by startups based on their ongoing projects. Future opportunities will allow students to sign up for internships offered by startups.
- Target users
  <click>
  - Our target users consist of students and startups. We are targeting undergraduates including recent graduates, who are innovative and highly

motivated, a team player, and have this curiosity and desire to learn and explore different career options. In terms of startups, we are targeting ones that have good team dynamics with a focus on collaboration, has a flexible structure, passionate for innovation and technology, and most importantly has experienced individuals who can become mentors for our students.
    - Now Alex will present to you our app and its features.

Alex:

- Demo

<click>

Process Discussion

Sanjay:

Here is how we worked together as a team.

<click>

We had quite a few methods of communication, mainly using in person meetings, discord and Slack for important project related details. We had in-person meetings on Tuesdays after the lecture and on Saturdays at either 11 AM, or 1 PM, where we would discuss which time to meet beforehand. During these meetings we would discuss the details of what we need to do for each sprint, do some of the deliverables as a group, create Jira stories and assign ourselves to them and whatever else needs to be discussed or brought up in these meetings. We later included the option to attend these meetings virtually, through a discord call. Discord was our main method of general communication due to popularity and familiarity for all group members, allowing for easy and fast communication. We used Slack for project details, important information, meeting notes and standups. In addition, we had a Whatsapp group which included a founder, Veb, using which we discussed project developments with him. He would sometimes attend our Saturday meetings either in-person or virtually through a zoom call.

<click>

Our utilization of github was through using dev as our developmental branch, where for every Jira story we would branch off dev, with the branch name starting with the story ID, so story C01-50 would have a branch starting with C01-50 and so on. When we finish implementing a story, we would make a pull request on that branch to dev, wait for another member to approve the pull request, and then merge and integrate that branch into dev. Starting sprint 3, we changed our definition of done so that we would only mark a Jira ticket as done after the related branch was merged and integrated to dev. At the end of the sprint, we merge dev to main.

<click>

We divided our tasks in different ways in Sprint 1 as compared to sprints 2-4. During sprint 1, we didn't assign any story points, so after selecting priority tasks, we simply assigned ourselves to at least 1 task each, with unassigned tasks being picked up later on.

In sprints 2-4 we assigned story points to all of the stories, generally taking around 40 or more story points worth of stories per sprint. By dividing the total story points by the number of members, we tried to generally assign ourselves to stories worth the average number of points per member to reach the goal, which tended to be around 6 story points per member. Sometimes the assignment values changed due to situations like unfinished stories being brought over from a previous sprint.

<click>

We made many significant decisions through this project, but here are some of them. Starting with the bad decision, as this was one of our first decisions. For sprint 1, we created Jira tasks that were too big, they were a lot of work for individual group members, and some even relied on others being completed first. This led to some unfinished tasks.

Now, for the good significant decisions we made. We created Figma prototypes of what we want before starting implementation, allowing for saved time by having a guideline to follow and for us to have a consistent design theme. As compared to each developer implementing their own design, leading to even related pages looking like different websites. Next, we made Jira stories smaller and more fine-grained, making them more easily divisible, manageable and improving completion rates. One of the reasons for this change was from learning from our bad decision with large tasks. Lastly, the changing of our definition of done that I mentioned earlier when discussing github was a good significant change. It minimized conflicts and allowed us to be able to easily merge to main and release our features at the end of each sprint.

<click>

Technical Discussion
<click>

Giselle:

- During the development of our project, one of the design changes we had is related to optimizing API response for better performance.
- The main issue is the absence of chatId in the response to the Create Chat POST request. Initially, the response provided only a message indicating the success of the chat creation process but lacked the necessary "chatId" field. This is because Create Chat POST request is used when creating a group chat. This omission hindered the smooth integration of the chat creation process with other functionalities, such as displaying chat messages that are sent through a popup window. chatId is a mandatory body in updating chat message PUT requests so that we could update the corresponding message array in the database.

<click>

- The solution involved both the frontend and backend components of the application.
- In the backend, chatId was added to the response of the POST request. In the frontend, we use createChatUpdate function to save the chatId from the API response in the saga file. In addition, we add the function useCreateChatContext() to get the chatId in the popup window javascript file, so that when the response is returned, we could use this chatId in the update chat message request.

<click>

Katy:

- In our pursuit to enhance user experience during sprint 4, the absence of a navigation bar for startup users emerged as a significant issue. Consequently, our initial focus lies in attributing specific types to the navigation bar. As we delved deeper into this predicament, we formulated a potential solution: assigning types as props to the Nav component. However, this approach did not yield the anticipated outcomes. One primary limitation resulted from certain web pages being shared between student and startup users; the simple addition of a prop proved inadequate.
- Consequently, we arrived at an alternative strategy: the elimination of the type prop in favor of using localStorage to transmit type data across various pages. Given the non-sensitive and straightforward nature of user type information, this seemed like a fitting approach. Subsequently, we incorporated a few lines of code after user logged in successfully. These lines facilitate the local storage of the user's type, allowing access to this information from the Nav component at a later juncture.

<click>

Software Architecture

Jawad:
We used a 3 tier architecture, separated into the frontend, backend and the database..

For the frontend it's structured around four main components:

1. The Components Folder is where we store individual UI elements.
2. The Context Folder is used for managing global state and shared data.
3. Our Pages Folder represents the different views or screens that users see and interact with.
4. And the Sagas Folder centralizes our API calls.

<click>

In terms of how these components come together:

- Each of our pages is constructed using multiple components.

- These pages, when they need to access global data, use the contexts.
- And when it comes to fetching data or interacting with our backend, both pages and components rely on sagas.

And the technologies used in the frontend were styled components to help with style of the UI, bootstrap and mui was used to speed up ui and component development, and finally axios was used to actually make api calls.

Pan:
<click>

Our backend is structured around four main components:

1. The Models Folder, which defines our data structure and handles interactions with the database.
2. The Routes Folder, which lays out all the available API endpoints.
3. The Services Folder, where the core logic for each endpoint resides.
4. And the Utils Folder.

<click>

When a request hits our server, it's first directed by the routes to the appropriate service. These service functions, when they need to perform data operations, interact with the models. Throughout this process, utility functions from the Utils folder are called as needed.

In terms of the technologies we used:

- We've built it on Express.js, a framework for Node.js.
- For our database interactions, we've used mongoose alongside the native MongoDB driver.
- And to boost our development efficiency, we've integrated Nodemon, which auto-restarts our server with every code change.

Jawad:

<click>

One of the challenges we had was large queries with useless data: As our user base expanded, we refined some of our database queries ensuring we only fetched necessary data.

Another challenge was managing global state using contexts, which required careful planning to avoid over-complication.

Additionally, a technique we used was implementing middleware, for handling critical tasks such as request authentication and request parsing.

Contribution

Alex:

- Setup sagas (for frontend / backend communication). Helped integrate some early solutions which ran into merge conflicts. Allowed editing of profile page for both students and startups (minus profile image). Fixed formatting and display bugs.

Pan:

- For my contributions, I helped with implementing the user logins. One of the features I added was the hashing of passwords and storing the authtokens locally. I also worked on the profile icons, they are editable from the user's profile page, and on account creation they are randomly given a default icon. And lastly, I did the student list page, which dynamically updates the page to show all the students in the database with the option to filter them. Next, I'll hand it over to katy.

Katy:

- In this project, my primary achievement involved conceptualizing and constructing the Home page, showcasing comprehensive chat-related information. I also implemented the functionality to enable users to remove chats. Additionally I created and refined the navigation bar for enhanced user experience. Next, I'll hand it over to Denise.

Denise:

- For my part, I implemented the frontend for the student profile and all students page, where I integrated each student's profile into a smaller cue card version to be displayed when filtering through the students. I created a customized scrollbar, to ensure a user friendly and smooth experience, regardless of the browser used. Lastly, I implemented the ability for users to search in chats, as well as the functionality to display a different chat icon for group chats, to visually separate it from direct messages.
- Now I'll hand it over to Giselle.

Giselle:

- In this project, my main contribution is the backend for startup user login and student user signup. Also, I was responsible for the implementation for the popup window for sending a message, and functionality to delete messages. In addition, I implemented the competition related features such as competition signup and registered student list. Next, I'll hand it over to Jawad.

Jawad:

- For my part, I worked on the chats features backend as well as the ability to create/edit chat groups. I also created the components used for the login and sign up form. And lastly I worked on different parts of the Competition page, I worked on the page that list all the competitions, and the view that is shown when a user clicks on the competition listing. It was made for both the student and start-up side of things. Next I'll pass it on to Sanjay :)

Sanjay:

- For my part, I first implemented user logout functionality, and then I implemented the backend and connection from frontend to backend for startup user edit profile and student user edit profile. This refers to allowing startup and student users to edit and save the information on their profile pages. I implemented backend and frontend for startup user create competition, a page in which startup users can create the competitions they'll host. Other than these tasks, I also implemented design improvements and error fixes, such as showing an 'invalid credentials' message when logging in with invalid credentials, adding backgrounds to the dropdown menus and fixing the error on view profile in dev. Next, I'll pass it back to Denise.

<click>

End

Denise:

- Thank you everyone for listening to our presentation!
- Please feel free to let us know if you have any questions.