CODE 4 GovTech

UCI Web Channel

Project Proposal

By

Neelesh Sharma

Index

Summary

Currently, the project is in the very initial stages where we can only send messages to initialize the chatbot. But we are not storing anything anywhere. We have to implement the authentication so that users can signup for their account using credentials (phone no. or email). After signup, users will be interacting with the chatbots to fulfill their tasks. I will be working on the features to enhance and upgrade the custom web channel for UCI, from a PoC to being production-ready.

I will be making the user interaction with the bot easier and more convenient by adding chips as buttons for quick replies. File/media sharing along with location fetching and sharing will also be implemented. The other feature that I will be working on is a dropdown list with a multi-selection option. All the features are explained in detail ahead with along with my implementation approach.

It's been more than 1 month since I first started exploring, and working on the vast codebase of the UCI web channel. I need no more time to explore the codebase and the workflow of the project, from solving beginner issues to designing the interfaces of the final output of the project, this shows that I have explored the project pretty well.

I have the required skill set for this project and I’m looking forward to making myself equipped by working with the mentors in this C4GT.

Relevant Skill & Motivation

I'm in the 3rd year currently pursuing a Bachelor of Technology in Computer Science & Engineering. I got introduced to the world of open-source in my second year and since then I have been involved in various local open-source groups present on and around my college campus. Open-source programs really fascinate me because this is the way through which we can give something back to the community rather than sitting back and mugging up things. Since the past year, I was learning new techs and now I feel it's a great opportunity to use my skills where I can benefit the community. Code4GovTech is a government program and India's first initiative toward open-source programs and whatever we will build is going to affect the lives of citizens of India or maybe other countries too. Doing something for our country is another motivation for me.

Relevant Skills

- Familiarity with **javascript, Reactjs, Typescript, Websockets**
- Familiarity with frontend designing and UI/UX
- Understanding of Linux
- Able to communicate complex ideas to non-specialists

Previous Contributions

Pull request opened:

[#6] Disabled sending empty messages and added toast error alert for the same. (Merged)

[#24] Created the Figma designs and added them to the Readme.MD . (Merged)

Issues Participated/Resolved

[!3] Able to send empty text messages. (Resolved)

[!8] User is not clear about the first/starting message. (Opened)

[!12] OTP-based login Screens. (Working)

Project Details

Organization: Samagra-comms

Project: UCI Web Channel

Mentors: Chakshu Gautam, Shruti Agarwal

This project is about enhancing a configurable platform, UCI, to enable personalized chatbot communication across channels such as WhatsApp, Telegram, SMS, email, and more. UCI is an open-source communication platform that is powering the national DIKSHA WhatsApp bot. The aim of this project is to target the mass users so that the government can send information to the citizens whether it is an update on the processing of an application for a driving license, updating parents on their child's performance at school, or issuing directives across government levels say from the state to the district and block-level.

Project Overview

Problem 1:

There is no authentication or we can say the user cannot create an account on the UCI web channel.

Solution:

We can achieve authentication with the same model as WhatsApp.

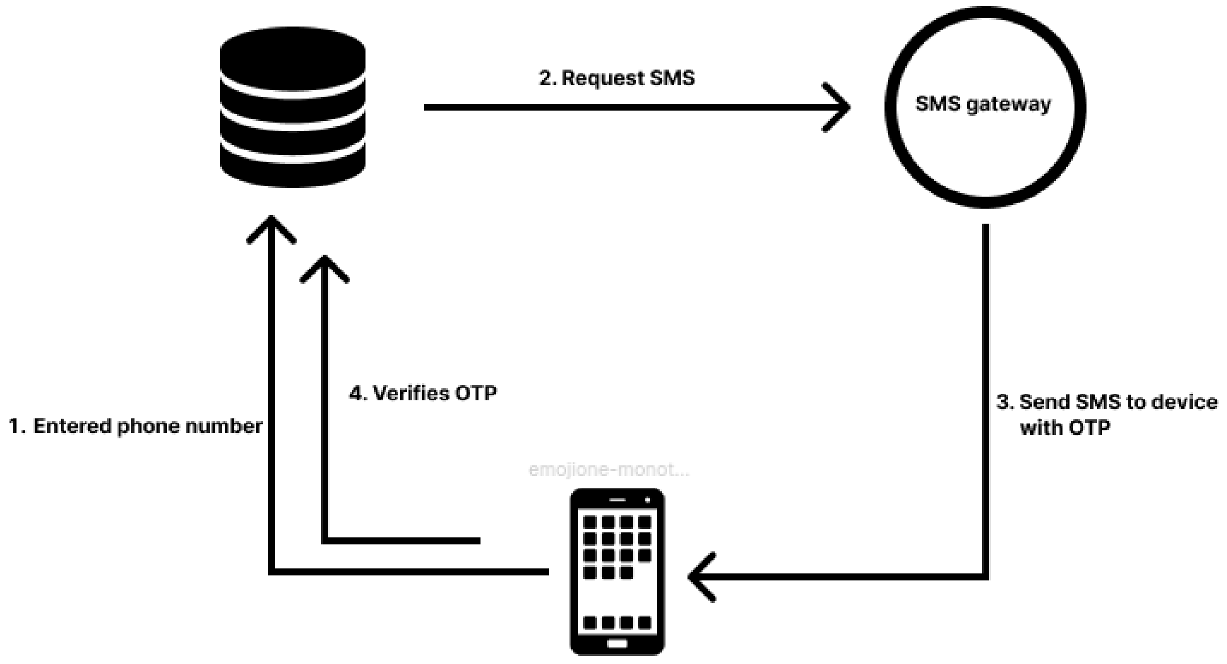
Generally, we use Email and passwords for authentication but the problem with that is users need to remember many passwords nowadays, and to remember passwords easily they choose dangerously common passwords which are easy to crack so the alternative we can use is phone number verification. Authentication based on mobile phone number verification is an ideal replacement for passwords. Let's see how we are going to do it:

Implementation Approach

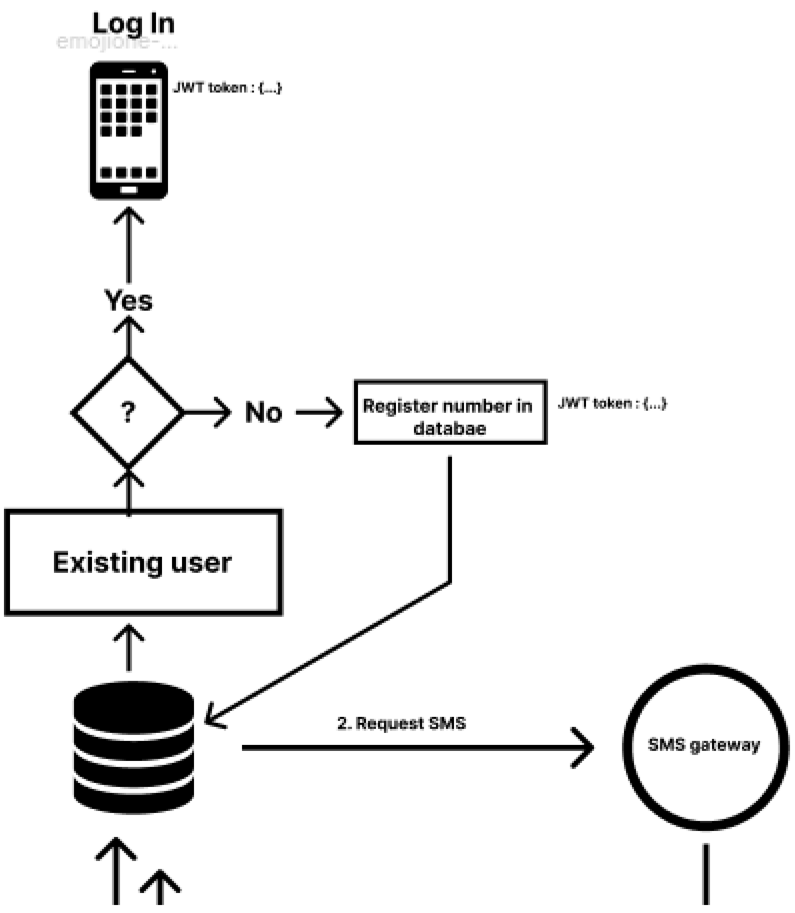
We are going to add a signup/Login feature for the UCI bot which will help the new users to easily sign up on the UCI channel and easily login through a managed user login service with enabled authentication.

We will be needing 2 APIs:

1. For initiating the send OTP process for verifying the authenticity of the user

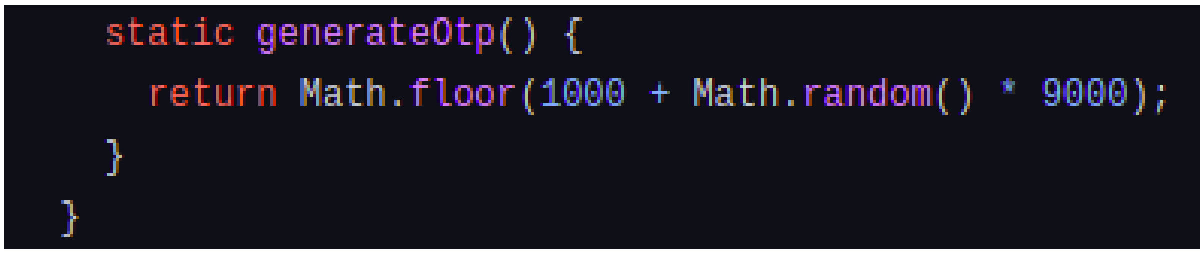


2. A register/login API would verify the OTP, if the OTP is verified and is an existing user, it would log in and return JWT tokens for the user and if it is a new user it will register the user and then return a JWT token in the response.



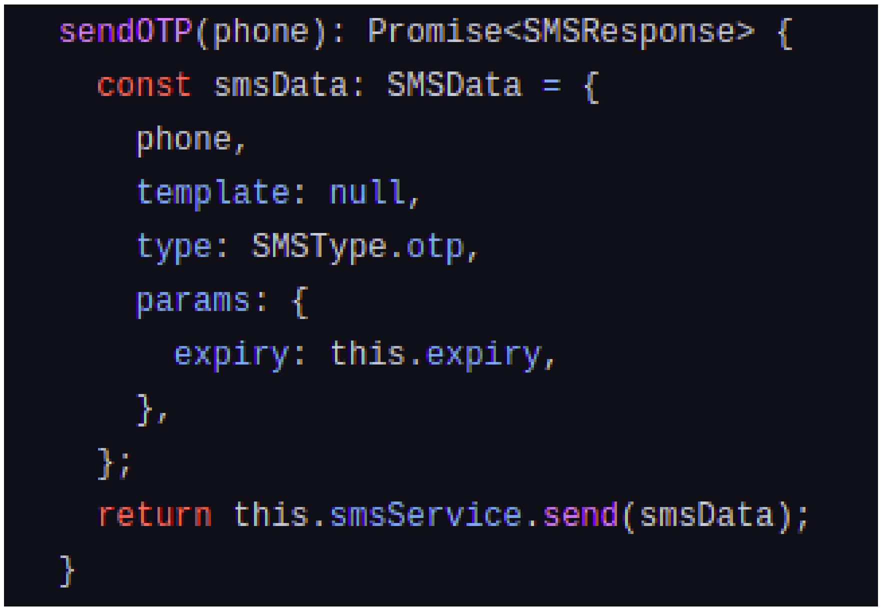
We will be using fusion auth typescript SDK for API calls. Following are the representations of some of the functions which we will be using :

Generating 4-digit random OTP



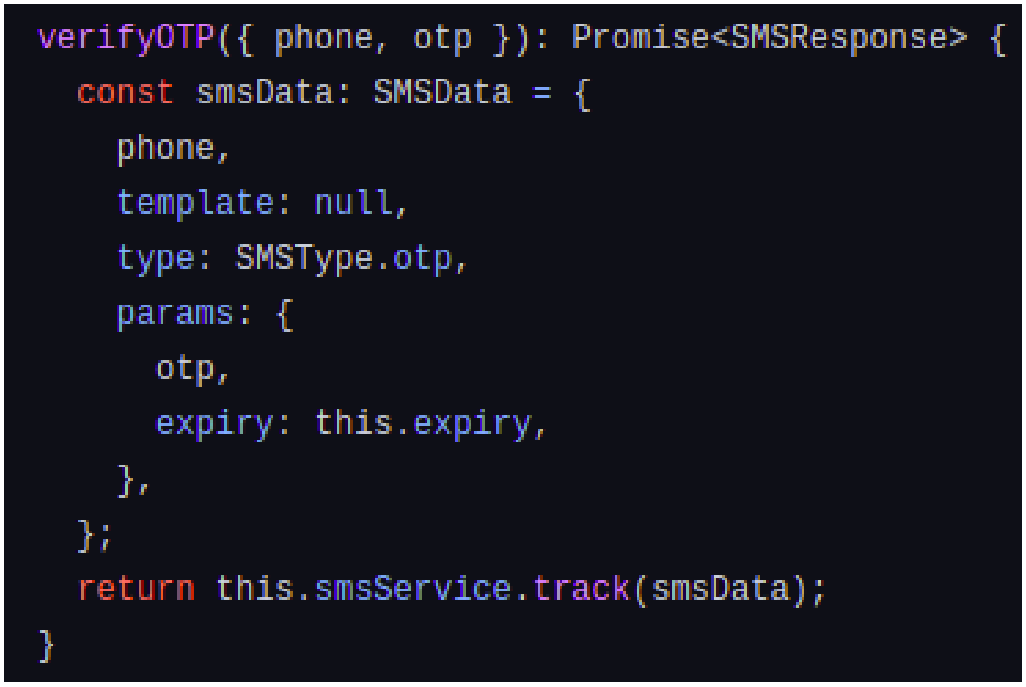
```
    static generateOtp() {
      return Math.floor(1000 + Math.random() * 9000);
    }
  }
```

Sending OTP



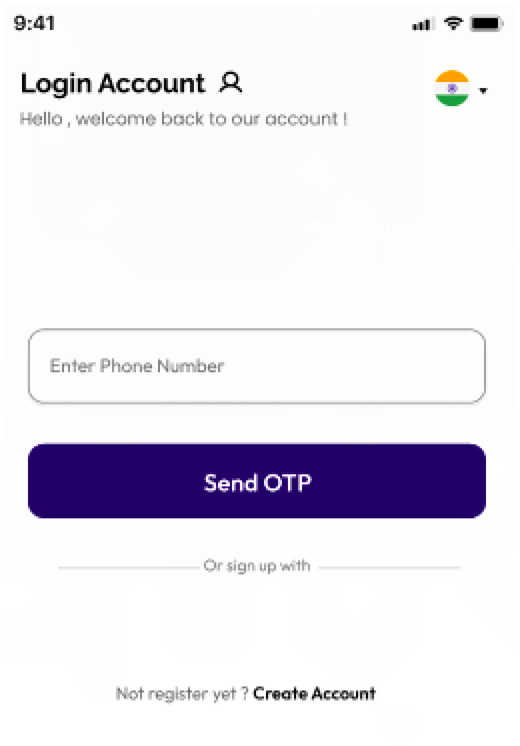
```
sendOTP(phone): Promise<SMSResponse> {
  const smsData: SMSData = {
    phone,
    template: null,
    type: SMSType.otp,
    params: {
      expiry: this.expiry,
    },
  };
  return this.smsService.send(smsData);
}
```

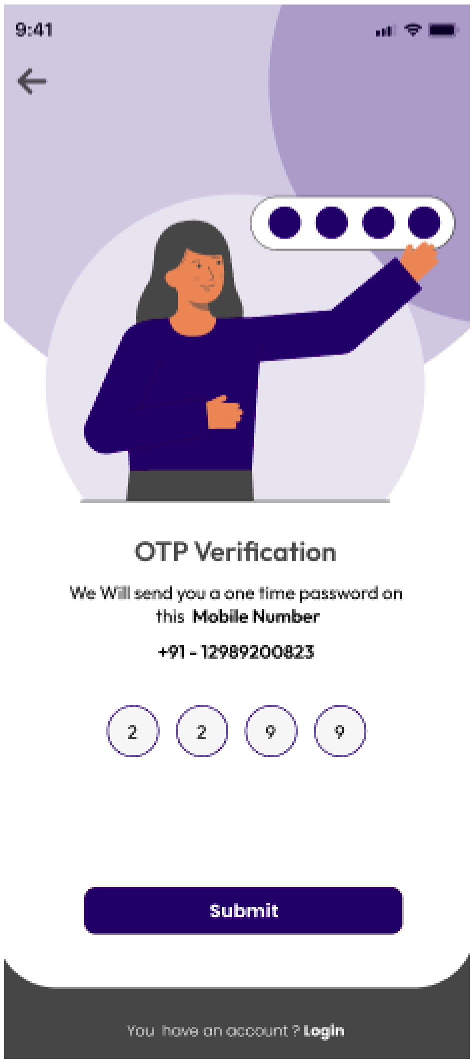
Verifying OTP



```
verifyOTP({ phone, otp }): Promise<SMSResponse> {
  const smsData: SMSData = {
    phone,
    template: null,
    type: SMSType.otp,
    params: {
      otp,
      expiry: this.expiry,
    },
  };
  return this.smsService.track(smsData);
}
```

How Final Screens will look like


9:41
Login Account
Hello , welcome back to our account !
Enter Phone Number
Send OTP
Or sign up with
Not register yet ? Create Account


9:41
OTP Verification
We Will send you a one time password on this Mobile Number
+91 - 12989200823
2
2
9
9
Submit
You have an account ? Login

Problem 2:

We cannot fetch the older messages as they are not getting stored anywhere right now and that's why sessions getting created are not persistent.

Solution:

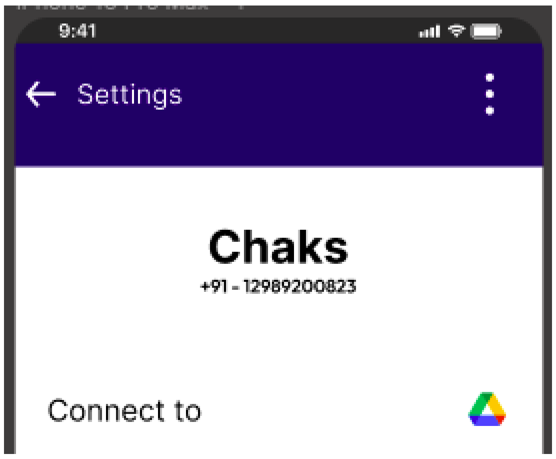
To overcome this problem we can use the local storage of our browser to store the messages.

Since the UCI web channel is in its very early stages, currently there is no way to store the messages. But once the authentication will be done we can store the messages locally using the browser storage of the user.

When the local storage will get filled, we can move the message backup to my other preferable location of the user, let's say google drive, just like the backup system of WhatsApp.

To use the local storage and make the session persistent, we have use-localstorage that replaces the useState hooks used previously to give persistence to user login. It allows us to use our local storage as a variable store in our application. Data that remains constant, such as username, a room to which the user is connected, etc. is stored in localStorage so that it persists between user sessions.

To shift the data from local storage to google drive, we need to access the google drive of the user for which we can give the option in th settings page.



We can set up the OAuth 2.0 to connect to the google drive using Google APIs. OAuth 2.0 allows users to share specific data with an application while keeping their usernames, passwords, and other information private.

Now let's find out answers to some questions on how are we dealing with the backup system.

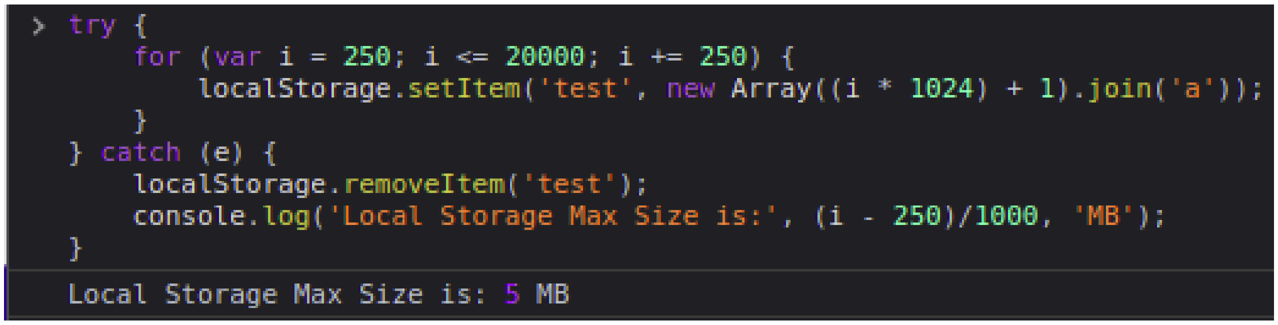
Question 1. How I will move this data to Google Drive when the local storage is at a certain limit?

Approach:

We can check the maximum size of the local storage of the browser which is 5 MB for most the browsers. Users can increase this in the settings.

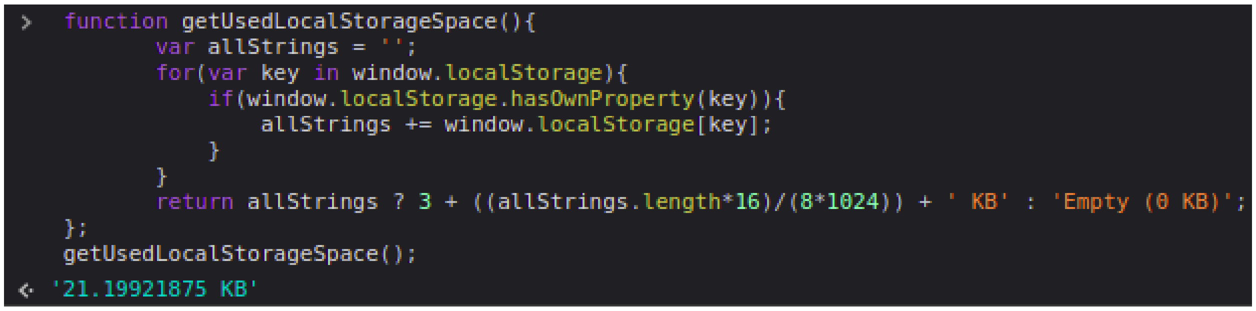
Now we can keep track of the local storage and we can predefine some value, let's say 80%. When local storage crosses the predefined limit we move everything to the google drive and delete the data from the local storage.

To get the maximum size of the local storage:



```
> try {
      for (var i = 250; i <= 20000; i += 250) {
          localStorage.setItem('test', new Array((i * 1024) + 1).join('a'));
      }
  } catch (e) {
      localStorage.removeItem('test');
      console.log('Local Storage Max Size is:', (i - 250)/1000, 'MB');
  }
  Local Storage Max Size is: 5 MB
```

To get the storage used after storing messages to the local storage:



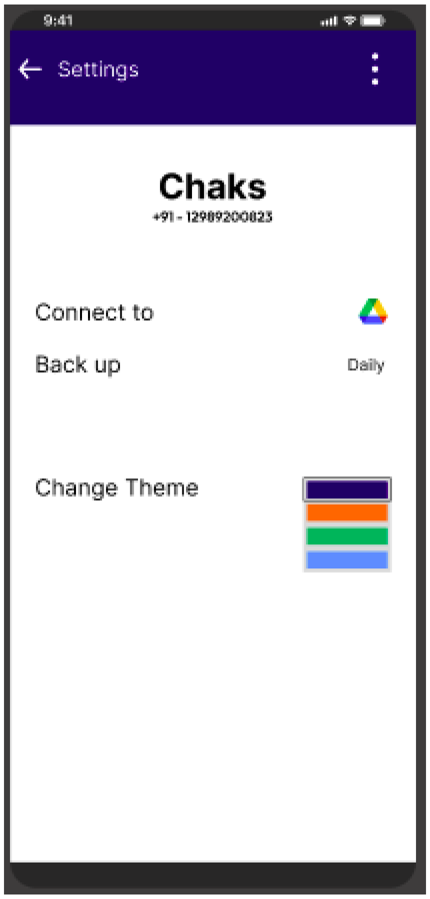
```
> function getUsedLocalStorageSpace(){
          var allStrings = '';
          for(var key in window.localStorage){
              if(window.localStorage.hasOwnProperty(key)){
                  allStrings += window.localStorage[key];
              }
          }
          return allStrings ? 3 + ((allStrings.length*16)/(8*1024)) + ' KB' : 'Empty (0 KB)';
  };
  getUsedLocalStorageSpace();
< '21.19921875 KB'
```

Question 2. How will the user configure this on the frontend?

Approach:

We can create a settings page where we can ask the user to update the backup time and frequency (daily/weekly/monthly). We can select and set some default values to these settings.

Settings page



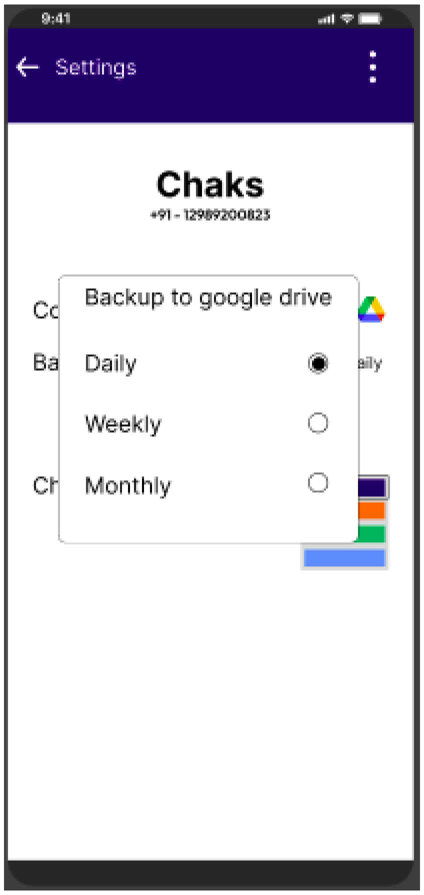
Question 3. When will the data get transferred to Google Drive - cron style at night?

Approach:

Data can be transferred to google drive in 2 scenarios:

a) When the local storage will cross the set limit. After transferring the data, it can easily be deleted from the local storage.
b) Daily/weekly/monthly selected by the user in the settings. At the selected time backup cron will trigger the backup script. This process will be automated. If the user will select weekly then at the end of every week cron will trigger the backup script.

Options of backup



Backup to google drive

Daily [✓]

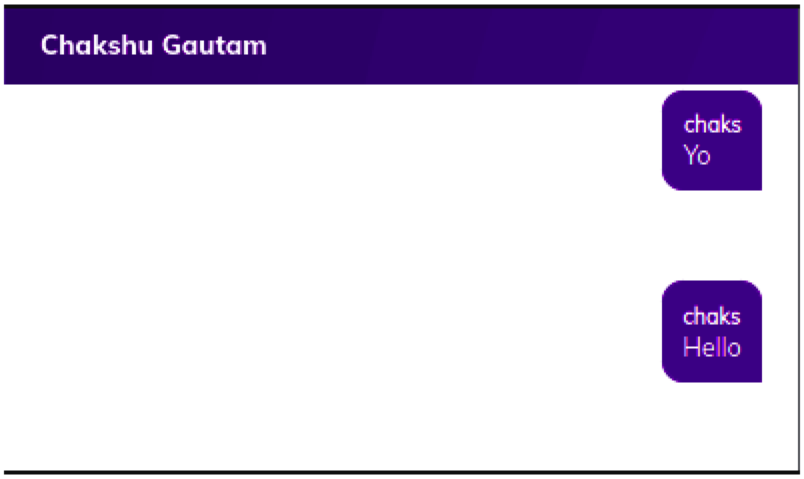
Weekly

Monthly

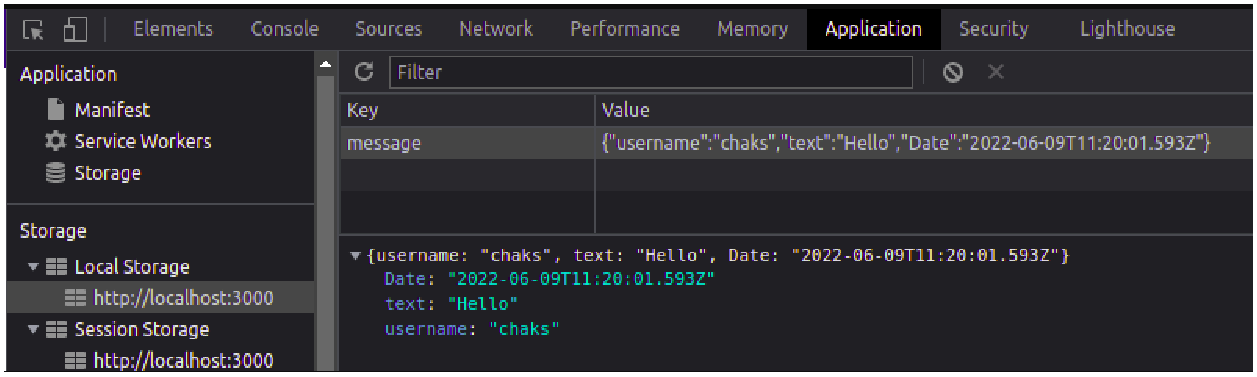
Question 4. How will I keep track of what is pushed to Google Drive and what is not?

Approach:

This problem can easily be solved by adding the time stamp to the backed-up files. We can then read the last backed-up file and everything on and after that timestamp is yet to be pushed and need to be pushed to the google drive.



Messages getting stored in local storage with date and time



Key	Value
message	{"username":"chaks","text":"Hello","Date":"2022-06-09T11:20:01.593Z"}

```
▼{username: "chaks", text: "Hello", Date: "2022-06-09T11:20:01.593Z"}
   Date: "2022-06-09T11:20:01.593Z"
   text: "Hello"
   username: "chaks"
```

Question 5. When logging in again on a new device, the user should be able to download the messages from Google Drive and view them although this should not impact sending new messages.

Approach:

When the user will log in to a new device, we can validate the user. If the user validates then we can check if there are any existing backup files in the linked account or not. If yes, we can restore the messages to the new device.

Problem 3:

We need to improve the UI of the web bot by adding some new features like location sharing, a recent chats section, quick replies, and a cascaded,multi-selection dropdown list.

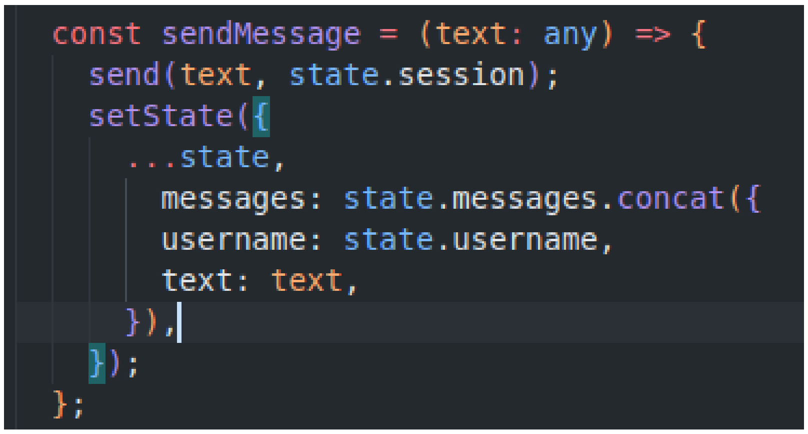
Solution:

1. **Sidebar with recent chats similar to WhatsApp**

Since we are using arrays to store the messages, it will be very easy to sort the messages according to the most recent message sent With the other attributes present in the message array we can store another “Date” attribute.

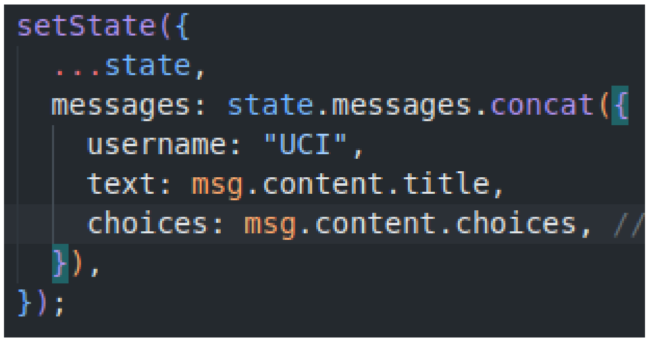
Current Message Structure

1. **Message sent by user**



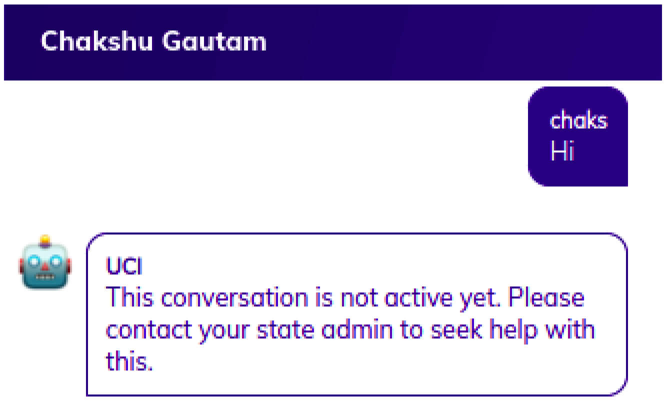
```
const sendMessage = (text: any) => {
  send(text, state.session);
  setState({
    ...state,
     messages: state.messages.concat({
     username: state.username,
     text: text,
    }),
  });
};
```

2. **Message received by user**



```
setState({
  ...state,
  messages: state.messages.concat({
    username: "UCI",
    text: msg.content.title,
    choices: msg.content.choices, //
  }),
});
```

Example:



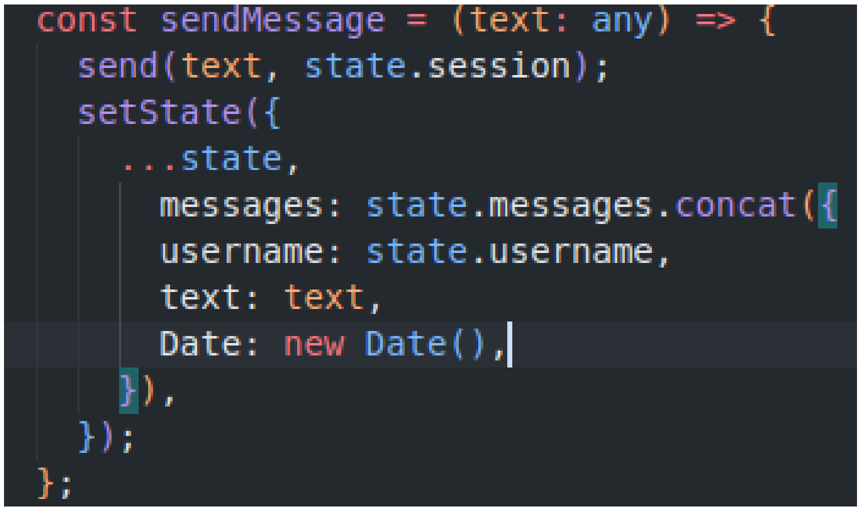
Chakshu Gautam

chaks
Hi

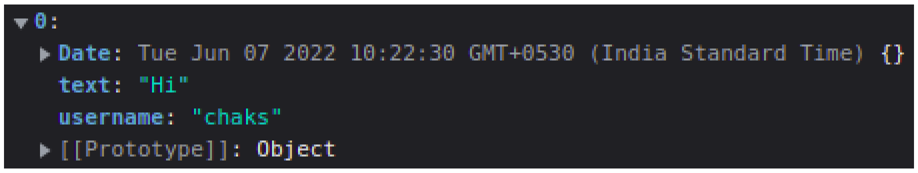
UCI
This conversation is not active yet. Please contact your state admin to seek help with this.

Proposed Message Structure

1. **For messages sent by user**

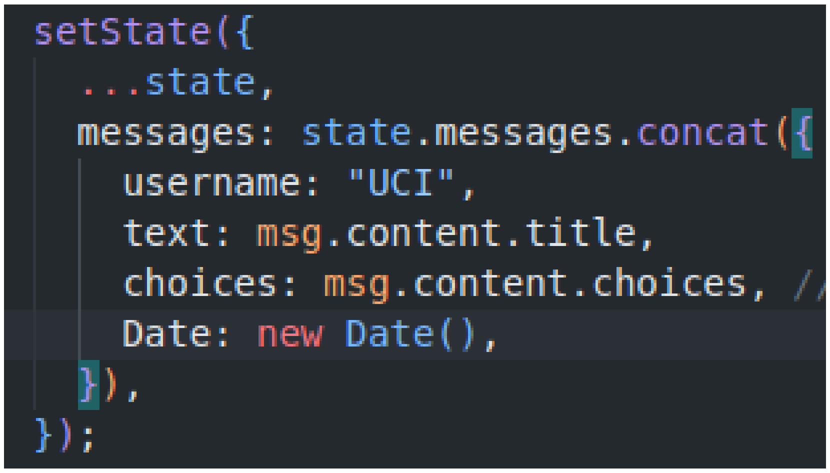


```
const sendMessage = (text: any) => {
  send(text, state.session);
  setState({
    ...state,
      messages: state.messages.concat({
      username: state.username,
      text: text,
      Date: new Date(),
    }),
  });
};
```

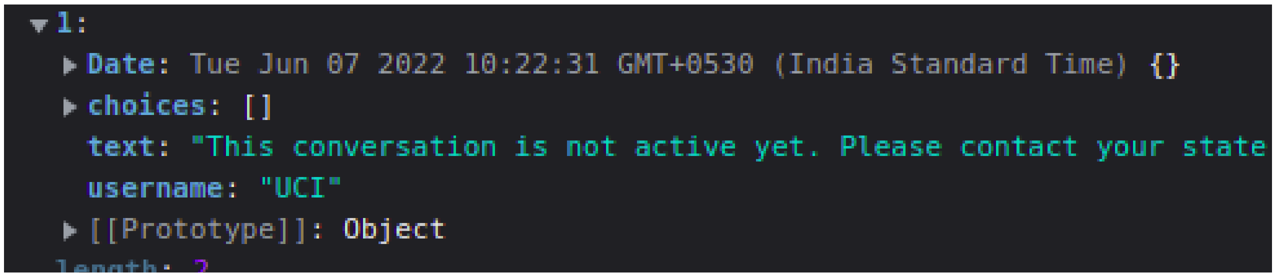


```
▼0:
  ▶Date: Tue Jun 07 2022 10:22:30 GMT+0530 (India Standard Time) {}
   text: "Hi"
   username: "chaks"
  ▶[[Prototype]]: Object
```

2. **Message received by user**

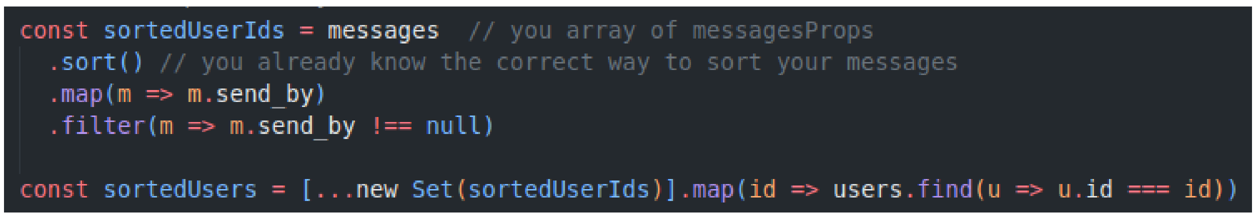


```
setState({
  ...state,
  messages: state.messages.concat({
    username: "UCI",
    text: msg.content.title,
    choices: msg.content.choices, //
    Date: new Date(),
  }),
});
```



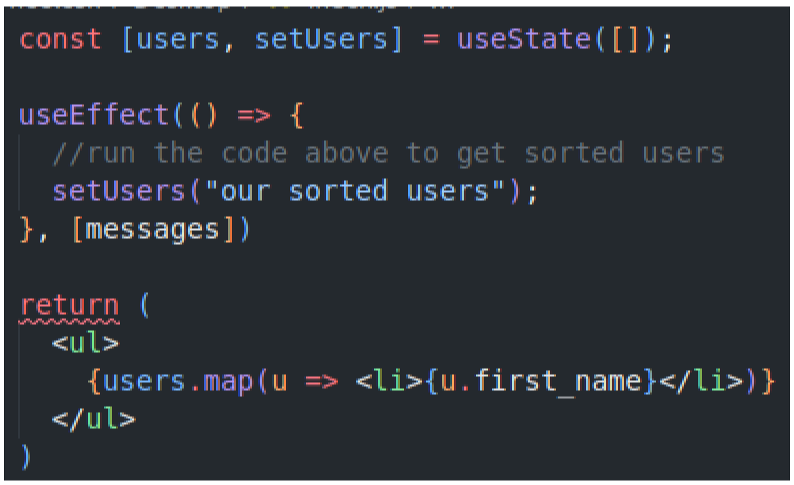
```
▼1:
  ▶Date: Tue Jun 07 2022 10:22:31 GMT+0530 (India Standard Time) {}
  ▶choices: []
   text: "This conversation is not active yet. Please contact your state
   username: "UCI"
  ▶[[Prototype]]: Object
 length: 2
```

This will simply return the date with the time of every message sent and received as we can see. With time we can easily identify the last message sent in a chat section. Then we can compare the last message of every chat and sort them accordingly.



```
const sortedUserIds = messages  // you array of messagesProps
  .sort() // you already know the correct way to sort your messages
  .map(m => m.send_by)
  .filter(m => m.send_by !== null)

const sortedUsers = [...new Set(sortedUserIds)].map(id => users.find(u => u.id === id))
```

Finally, we can return the sorted users as a list item to show them on the screen.

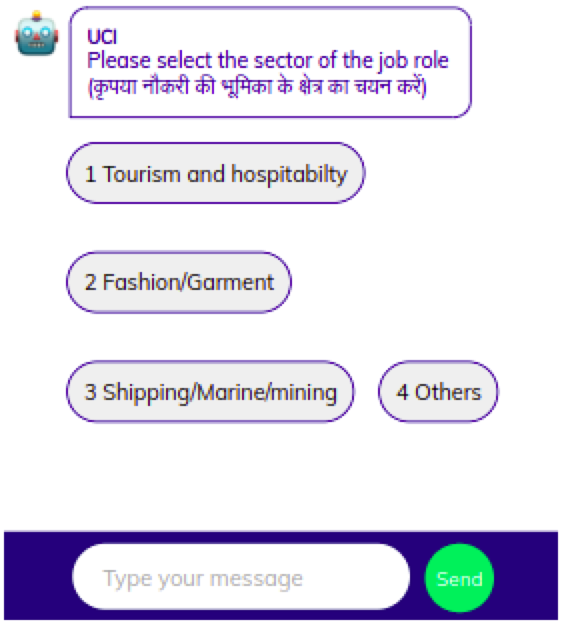


```
const [users, setUsers] = useState([]);

useEffect(() => {
  //run the code above to get sorted users
  setUsers("our sorted users");
}, [messages])

return (
  <ul>
    {users.map(u => <li>{u.first_name}</li>)}
  </ul>
)
```

2. Chips as the button too quick reply

Quick replies (or chips as Google calls them) are pre-defined responses that chatbots offer to their users. They are displayed as bubbles next to the message typing area, and users can click one of the replies instead of typing it in. We are going to implement quick replies functionality in the web bot to reduce the chances of typing errors and make the user experience smooth.

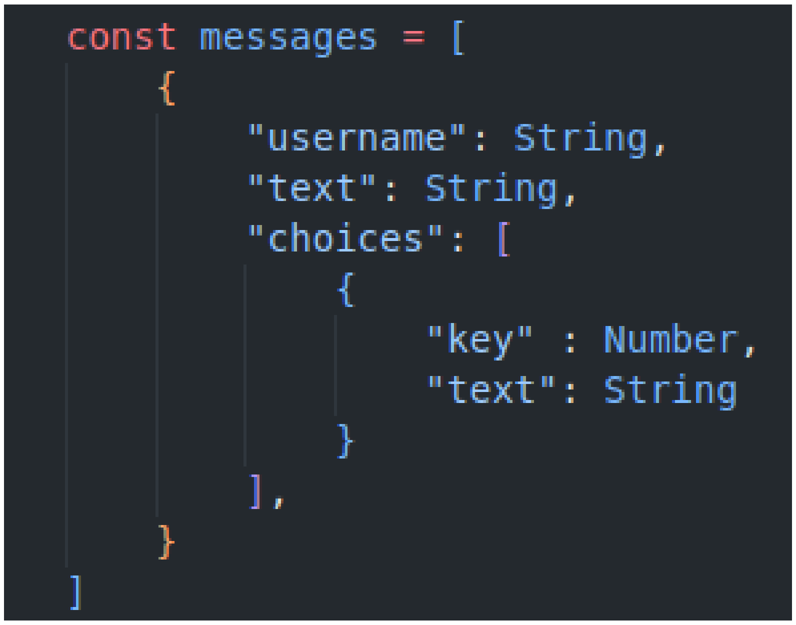
The following image shows how we are storing the quick replies and making them available to the user.



UCI
Please select the sector of the job role
(कृपया नौकरी की भूमिका के क्षेत्र का चयन करें)

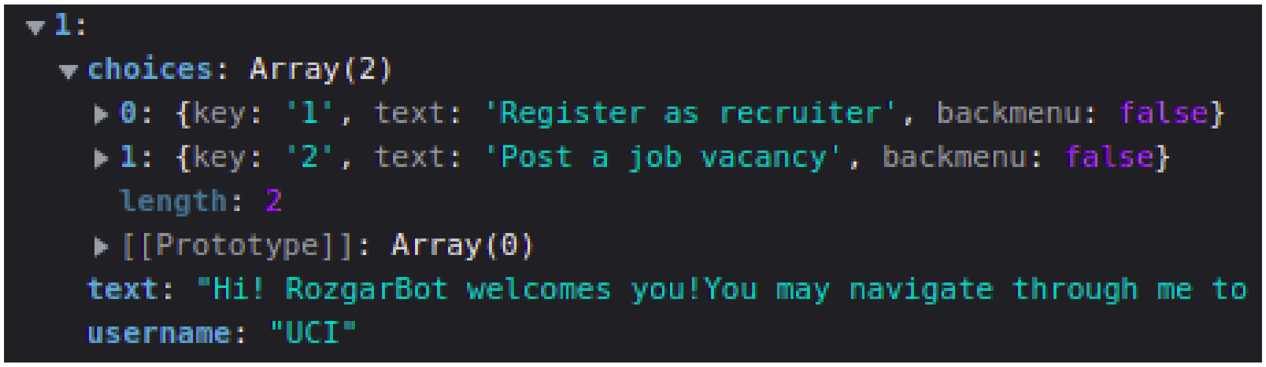
Users can directly choose an option by clicking one of the chips or they can enter their own answer by choosing others.

We can store the choices (options that we are giving in form of chips) in an array of let's say strings which will further be stored in the message array with other fields.



```
const messages = [
    {
        "username": String,
        "text": String,
        "choices": [
            {
                "key" : Number,
                "text": String
            }
        ],
    }
]
```

This will be our schema for the message



```
▼1:
  ▼choices: Array(2)
    ▶0: {key: '1', text: 'Register as recruiter', backmenu: false}
    ▶1: {key: '2', text: 'Post a job vacancy', backmenu: false}
     length: 2
    ▶[[Prototype]]: Array(0)
   text: "Hi! RozgarBot welcomes you!You may navigate through me to
   username: "UCI"
```

A practical view of the schema of how it will look in the console and get stored

We can map the number of options available and can generate the same number of buttons and can pass the choices as data when the user will click the choice button.

But, showing so many choices to the user in the form of chips can make the interface messy and it can reduce its readability. So, I am planning to use chips as quick replies if the number of choices is up to 5.

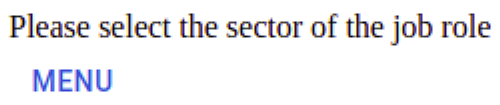
Many fields require more than 5 options to choose from and for that case, we are not using quick replies. Then what can we use? Solution to this problem is that we can use a cascaded list/menu and show it to the user.

3. List (cascaded and multi-select) as a dropdown

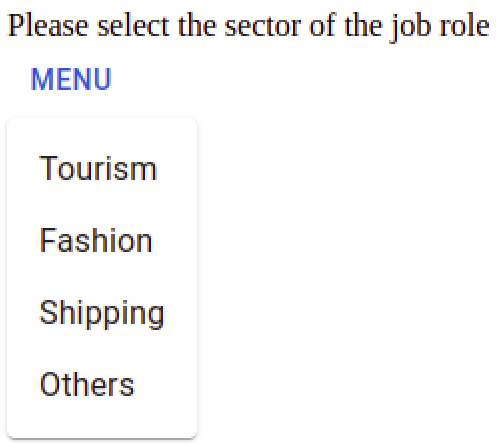
A cascading drop-down list is a series of dependent DropDownList controls in which one DropDownList control depends on the parent or previous DropDownList controls. The items in the DropDownList control are populated based on an item that is selected by the user from another DropDownList control.

If choices exceed 5 then rather than showing so many chip replies we can represent the choices in the form of a menu and show it to the user.

This is how the output will look like


Please select the sector of the job role

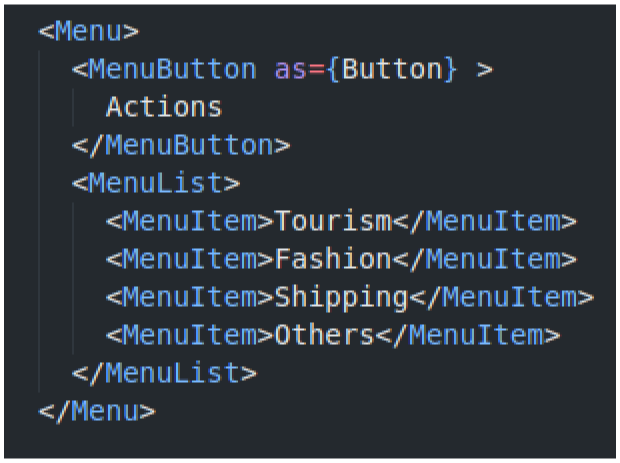
1. Before clicking the menu button



2. After clicking the menu button

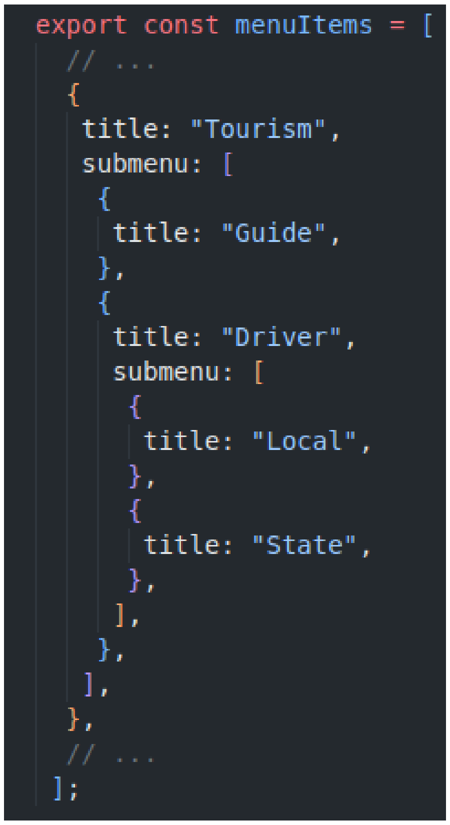
Users can choose any of the menu items by clicking on them. Further options will depend on what users will choose from the present list. This is called cascading.

How we are going to implement menuList?



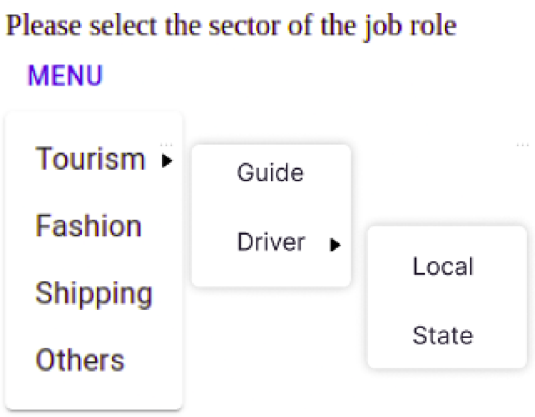
```
<Menu>
  <MenuButton as={Button} >
    Actions
  </MenuButton>
  <MenuList>
    <MenuItem>Tourism</MenuItem>
    <MenuItem>Fashion</MenuItem>
    <MenuItem>Shipping</MenuItem>
    <MenuItem>Others</MenuItem>
  </MenuList>
</Menu>
```

This is a simple Menu with single choices. If there will choices within a choice, we can create an array of objects for leveling the choices as follows.



```
export const menuItems = [
  // ...
  {
   title: "Tourism",
   submenu: [
    {
     title: "Guide",
    },
    {
     title: "Driver",
     submenu: [
      {
       title: "Local",
      },
      {
       title: "State",
      },
     ],
    },
   ],
  },
  // ...
 ];
```

This will result in the cascaded list.



4. Location Fetch and Sharing

To fetch the user’s live location, we can use Geolocation APIs. The Geolocation API allows the user to provide their location to web applications if they so desire. For privacy reasons, the user is asked for permission to report location information.

We can access the Geolocation API by calling the `navigator.geolocation;` this will cause the user's browser to ask them for permission to access their location data. If they accept, then the browser will use the best available functionality on the device to access this information (for example, GPS).

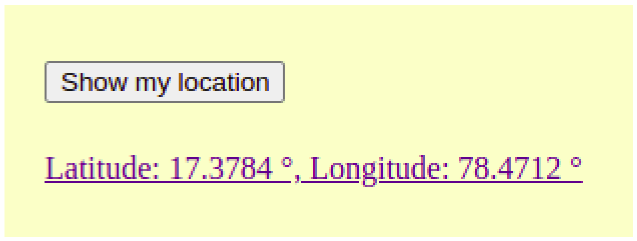
We can access the user’s location in two different ways:-

1. `Geolocation.getCurrentPosition()`: Retrieves the device's current location.
2. `Geolocation.watchPosition()`: Registers a handler function that will be called automatically each time the position of the device changes, returning the updated location.

Getting the Current location of the user

```
navigator.geolocation.getCurrentPosition((position) => {
    doSomething(position.coords.latitude, position.coords.longitude);
 });
```

Output:



Latitude: 17.3784 °, Longitude: 78.4712 °

For further enhancements, we can pass these coordinates to map services and can send the URL in the form of a message.

Milestones

- **Milestone 1 (Week 1 and 2)**

First milestone will be to enhance the UI of the UCI web bot by implementing the following features:

1. Sidebar recent chats
2. Chips as buttons for quick replies
3. Cascaded dropdown list
4. Location fetching and sharing

I will be utilizing the first 2 weeks to implement these features.

- **Milestone 2 (Week 3 and 4)**

After completing the UI part, I will be moving toward the authentication part. I have planned to utilize the next 1 to 2 weeks to implement the authentication part as I will be studying about now Fusion Auth works and then implementing it.

- **Milestone 3 (Week 5 and 6)**

After implementing the authentication, In the remaining time I will be working on the backup of messages and making the user session persistent.

Timeline

All the research-related work will be done in the community bonding period. As I'm already familiar with the codebase of the project, it will be easy for me to add new functionalities to the UCI. I will utilize this time to brush up on my web development skills and learn the new required skills by watching and following some relevant tutorials.

The evaluation period will be used to cope with backlogs and will be used to implement the requested changes mentioned in any PRs.

Week #No	Task
Week #0 (10 June - 19 June)	**Community Bonding Period.**
Week #1(20 June - 27 June)	Work with mentors to get the review and formalize the workflow and also review the design templates on Figma. Getting Started with implementing UI features. Start working on creating sidebar and location sharing
Week #2 (28 June - 04 July)	Polishing the above features and implementing the quick replies functionality including the cascaded list/menu.
Week #3 (05 July - 11 July)	Reading and understanding the workflow of authentication and getting familiar with FusionAuth and other requirements for authentication. Getting started with implementing the authentication and making signup/login pages

Week#4 (12 July - 18 July)	Finishing up the authentication part including the frontend pages and starting researching the backup system of WhatsApp in depth.
Week #5 (19 July - 25 July)	Working on the backup system of the web bot and implementing the functionality to store the messages locally in the browser.
Week #6 (26 July - 31 July)	Working on making the user session persistent and finalizing all the changes.

Future Development

These are some features on which we can work in the future to make the UCI bot more efficient and user-friendly

- Add more color themes.
- Handle case when there has been a socket error.
- Link Parser-Youtube link (embedded video should appear), link to an image (embedded image should appear), and all other links should appear as anchors.
- Auto-scroll to the bottom main chat area after scrolling up.
- Turn on or off notifications.

Availability

Since my summer vacations are going on I will be devoting **35-40 hours a week.** My college will reopen on 11th July so after that, I will be devoting around 30 hours because my placements will be starting after the summer vacations. If something comes up in between I will make sure to keep the mentors in the loop and try to finish off the work in advance :)

Personal Background

Hello, I am **Neelesh Sharma** currently doing my B.Tech from **Vellore Institute of Technology(VIT)** in **Computer Science and Engineering.**

I completed my schooling at Guru Harkrishan Public School, New Delhi with a **9.8 CGPA** in 10th and **88%** in 12th under the CBSE board.

Currently, my location is **New Delhi, India (GMT+5:30)**

These are my personal details:

- Github: Neelesh
- Email Address: neeleshsharma2512@gmail.com
- Mobile Number: +91 9625954572
- LinkedIn: Neelesh

Thanking Note

This is my final proposal for the project “UCI web channel” under Code4GovTech. I want to thank you C4GT for giving us this opportunity and organizing India's first open-source program. This will give confidence to the open-source community of India. Further. I want to Thank You my mentors who helped me in clarifying all my doubts regarding the project which helped me in completing this proposal successfully.

Hoping to work on the UCI web channel this summer ;)

Thank you once again, it couldn’t have been possible without your help and support :D