Farhikhta Farzan, Alyan Irdam, Nickolas Wang, Michael Zheng
CSCI355 Gray Paper

1.

```
//creating session collection
const store = new MongoDBSession({
  uri: process.env.MONGO_URL,
  collection: 'session',
});

//session setup
app.use(
  session({
    secret: process.env.SESSION_KEY,
    resave: false,
    saveUninitialized: false,
    store: store,
  })
);
```

One of the challenges I faced was to set up user session. Each user has their own session where userid and email and other session information is stored. The issue I was facing was that the session wasn't being stored in the database. But after checking everything, i figured out there was an issue in connecting client side with server side.

2.

```
//registering user and storing to db
app.post('/register', async (req, res) => {
  try {
    const data = {
      firstName: req.body.firstname,
      lastName: req.body.lastname,
      userName: req.body.username,
      email: req.body.email,
      password: req.body.password,
    };

    // Insert the new user into the database
    await userModel.create(data);
    res.status(200);
    res.redirect('/login');
  } catch (error) {
    console.error('Error registering user:', error);
    res.status(500).send('An error occurred while registering the user');
  }
});
```

This is the server side of the user registration. The user details are sent to the server. The server parses the details being sent and uses it to create user in the database. After registration is done, the user is sent to login, to get authenticated and log in.

3.

```
//function to request server to login
async function handleLogin(email, password) {
  try {
    const response = await fetch("/login", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ email: email, password: password }),
    });

    if (response.ok) {
      window.location.href = "/";
      console.log("logged in");
    } else {
      console.error("Login request failed:", response.statusText);
    }
  } catch (error) {
    console.error("Error during login:", error);
  }
}

submitBtn.addEventListener("click", async (event) => {
  event.preventDefault();
  handleLogin(email.value, password.value);
});
```

```
//authetication and logging in user
app.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await userModel.findOne({ email: email }); // check if user exists in the database

    // If no user found, return error
    if (!user) {
      return res.status(404).send('User not found');
    }

    // Check if the provided password matches the one stored in the database
    if (password !== user.password) {
      return res.status(401).send('Incorrect password');
    }

    //starting a session
    req.session.user = {
      id: user.id,
      username: user.userName,
      email: user.email,
    };
    res.status(200).send('Login successful');
  } catch (error) {
    console.error('Error logging in user:', error);
    res.status(500).send('Error logging in user');
  }
});
```

This is the code for user login. From the client side, user email and password are being sent to server side for authentication purposes. The server looks up the email address in the database. If user with that email exists, the server checks the password information. If everything matches the user gets logged in.

4.

```
app.post('/generate', (req, res) => {
  const { role, userMessage } = req.body;
  console.log(userMessage);
  async function run() {
    try {
      const model = genAI.getGenerativeModel({ model: 'gemini-pro' });
      const chat = model.startChat({
        history: [],
        generationConfig: {
          maxOutputTokens: 500,
        },
      });
      const message = userMessage;
      const result = await chat.sendMessage(message);
      const response = await result.response;
      const text = response.text();
      return res.status(200).json({ responseText: text });
    } catch (err) {
      console.error('Error in generating response:', err);
      return res.status(500).json({ error: 'Error generating response' });
    }
  }
  run();
});
```

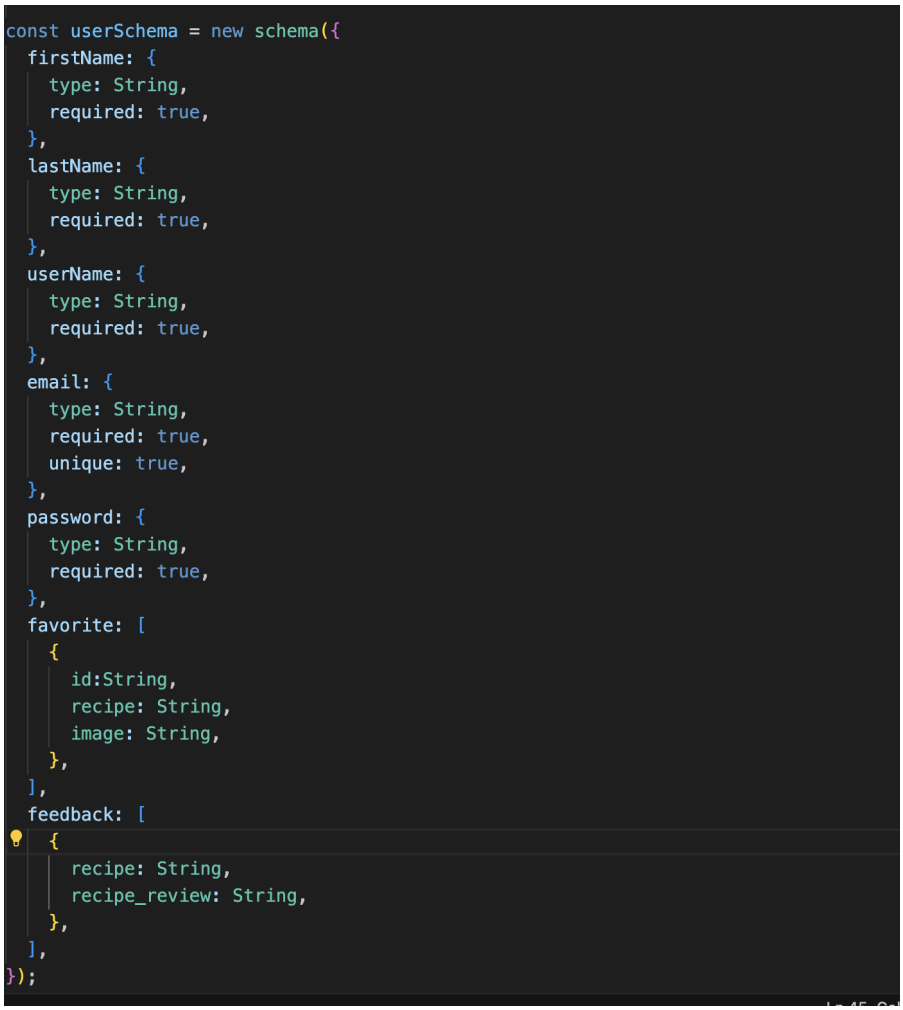
This is the code for chatbot. The chatbot uses Gemini api. The client side sends the user message to the server side. The server side parses that and uses that message to generate query to the bot. The history is stored and token count is set. The response generated by the bot is sent to client side to append to the chat.

5.

```
//function to request server to logout
async function handleLogout() {
  try {
    const response = await fetch("/logout", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
    });
    if (response.ok) {
      window.location.href = "/";
    } else {
      console.error("Logout failed!");
    }
  } catch (error) {
    console.error("Error during logout:", error);
  }
}
```

```
//Logout functionality
app.post('/logout', (req, res) => {
  //destroy the user session
  req.session.destroy((err) => {
    if (err) {
      console.error(err);
      return res.status(500).send('Logout error!');
    }
    res.status(200).send('Logout successful');
    // res.redirect('/'); // redirect to home page
  });
});
```

The above code shows the logout functionality. The client side requests for logout. The server side destroys the current user session and ultimately logs the user out.

6. 

```
const userSchema = new schema({
  firstName: {
    type: String,
    required: true,
  },
  lastName: {
    type: String,
    required: true,
  },
  userName: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
  favorite: [
    {
      id:String,
      recipe: String,
      image: String,
    },
  ],
  feedback: [
    {
      recipe: String,
      recipe_review: String,
    },
  ],
});
```

   This is the database. The db contains user information which includes first name, last name, email, username, password, favorites, feedback etc.
7. One of the biggest challenges encountered in the project is the positioning of elements, mainly images and formatting of said images and positions using CSS.
8. One of the difficult parts of this project was creating the vertical slider. The issue with it was due to the fact it was something I haven't done before. Took a good amount of time, but most of the help came from the Udemy video.
9. Another challenge was figuring out the Java Script code to utilize the search bar, so when you type, it will start to pull recipes that's on the TheMealDB.com api. It wasn't too bad once I figured it out
10. Since I was mostly working on Home page, pulling from the api was easy, but pulling from it on the home page and then clicking on it to redirect the user to the recipe page, that was sort of difficult. I had to figure out the JS code to make it so when the user clicks on the picture on the home page, it will take you to the recipe page, which on arrival will pull the recipe instructions and display it.
11. The most challenging part in this project for me, was figuring out MongoDB. It took me almost a whole day to get it working on my system.

12. The last challenge for me in this project has to be the css styling for the home page, due to having to constantly refresh the page to check the positioning, making it very time consuming. I personally think I spent most of my time figuring out the styling to make everything organized.