Title: QuickBook

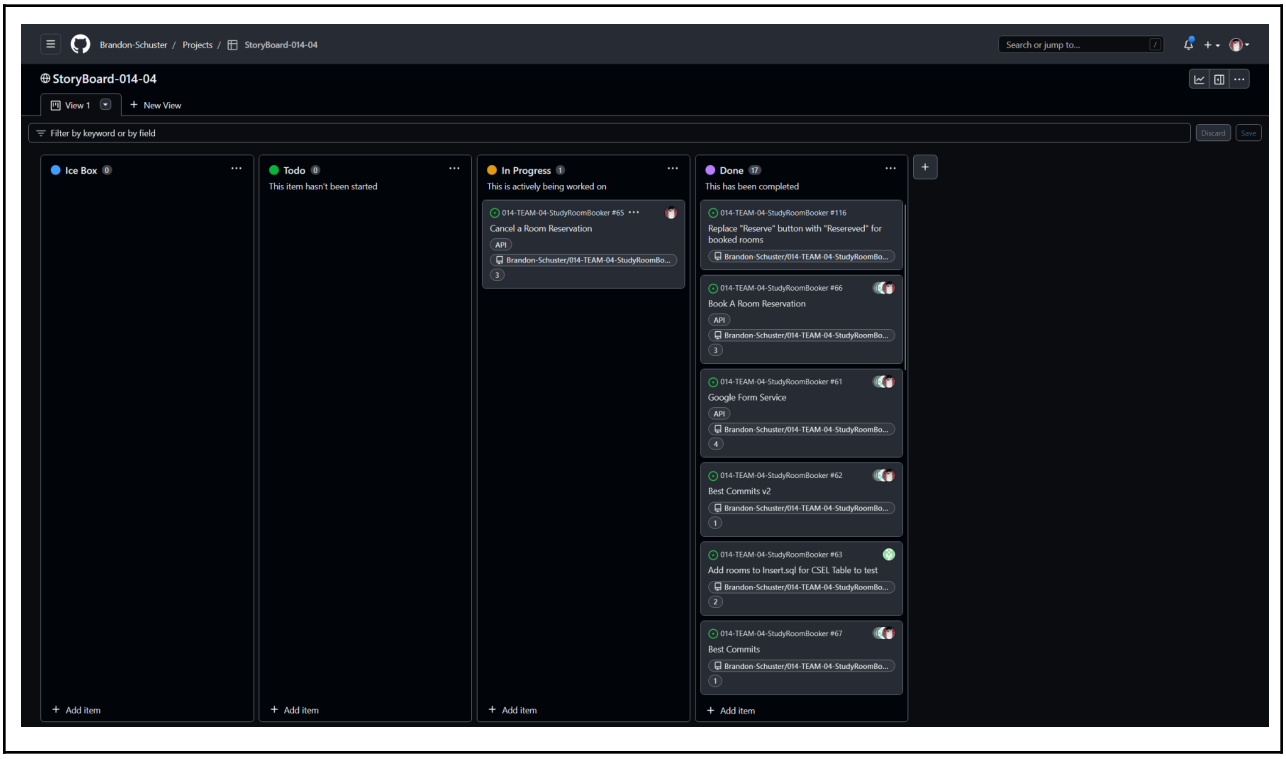
Who: Daniel Bonnecaze, Noah Garrett, James Gashi, Brandon Schuster, Jeremy Tang

Project Description:

QuickBook is a website dedicated to reserving study rooms in CSEL in an easy, efficient manner. After a user logs in they are directed to a table that contains the current date and various hour-long time slots. Every time slot has a button for each available room to book, so the user can easily choose which time slot they would like to book a room, and then choose a specific room. Once users click the reservation button they will be redirected to a Google form to fill out all relevant information and confirm their reservation. Another important feature of Quickbook is the profile page. The profile page serves various purposes. Aside from displaying your user information, you can update your password, delete your account, and view your current reservations. When viewing their current reservations users can see an organized list of their reservations. Another important purpose of the profile page is the ability to cancel reservations. Other features include a login, registration, and logout page. A common issue that Computer Science students face is being able to book study rooms in CSEL in advance instead of hoping that they will be available when they arrive. With QuickBook they do not have to worry about that anymore.

Project Tracker

- GitHub Project-Board



Video

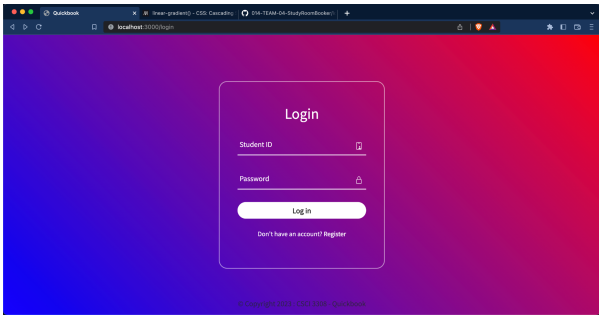
- Youtube Demo Video
- Repo Demo Video

VCS

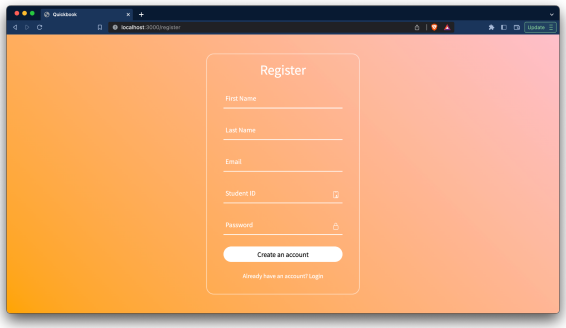
- GitHub Repo

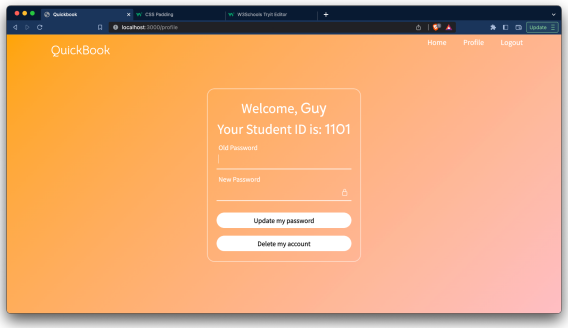
Contributions:
A brief (not more than 100 words) from each team member about their contributions.
This should include the technologies worked on
Features that have contributed to
You can also include:
A screenshot of the project Board
A screenshot of the contributions on GitHub

Jeremy's Contribution: Although my individual contributions began with a rocky start, most of my efforts went towards designing the UI and the front end. The majority of my contributions went towards ensuring a beautified, clean-looking application. Over the course of the project, I also acted as the scribe for all TA meetings. I also briefly worked on API setup as well as one of the POST api routes.

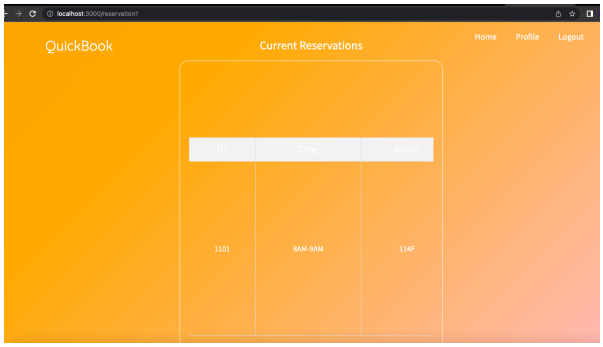



```
app.post('/register', async (req, res) => {
  //hash the password using bcrypt library
  const hash = await bcrypt.hash(req.body.password, 10);

  const info = `insert into students (first_name, last_name, email, StudentID, pwd) values ($1, $2, $3, $4, $5);`;

  db.any(info, [req.body.first_name, req.body.last_name, req.body.email, req.body.StudentID, hash])
    .then((data) => {
      user.first_name = req.body.first_name;
      user.email = req.body.email;
      user.last_name = req.body.last_name;
      user.studentid = req.body.StudentID;
      req.session.user = user;
      req.session.save();
      res.redirect('/home')
    })
    .catch((error) => {
      console.log(error);
      res.redirect(404, "/register");
    })
});
```





James's Contribution: I worked all over the project, including the preliminary database design, the reservations page, as well as the home page. Behind the scenes I also worked on the NodeJS file by adding multiple GET/POST routes and also contributing to many others. I mostly worked on the reservations page as well as the home page and node file with end routes. Below are a few screengrabs of some of my contributions.

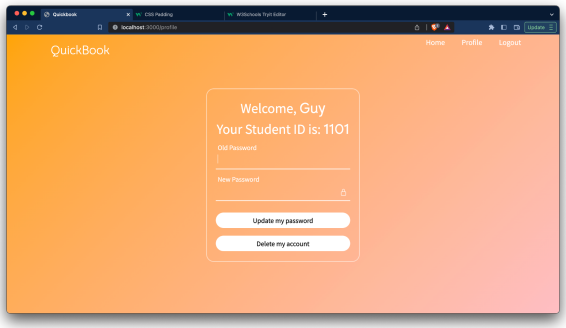



```
app.get("/reservation", (req, res) =>{
  let nStudentID = req.session.user.studentid;
 let Query = `SELECT * FROM bookings WHERE username = ${nStudentID};`;
 db.any(Query)
 .then((results) => {
  console.log(results[Object.keys(results)[Object.keys(results).length - 1]].chosentime);
  for(let i = 0; i < Object.keys(results).length; i++){
    results[Object.keys(results)[i]].chosentime = numbertoTime(results[Object.keys(results)[i]].chosentime);
    results[Object.keys(results)[i]].chosenroom = numbertoRoom(results[Object.keys(results)[i]].chosenroom);
  }

  res.render('pages/reservation', { data: results });
 })
 .catch((error) =>{
  console.log("error")
 });

})
```

Brandon Schuster's Contribution: During my first week, I set up the repo and all of the skeleton code required for our project to get on its feet. Created the primary views for our site and set up the test cases. I fixed our external google API and set up our .js file so that it would insert the data from the google form to our local PostgreSQL DB. I reworked most of our SQL database to accommodate for this. From there I worked to make our project look as professional as possible by adding the necessary stylization in preparation for our presentation. Once done, I deployed our web server using Microsoft Azure. Reference image below.

```
    chosenTime = timeToNumber(chosenTime);

   // console.log(timeStamp, chosenDate, chosenRoom, chosenTime, username, notes);
    // Insert the data into the database.
    const insertQuery = `
      INSERT INTO bookings (timeStamp, chosenDate, chosenRoom, chosenTime, username, notes)
      VALUES ($1, $2, $3, $4, $5, $6);
    `;

    // Use an array to hold the values corresponding to the placeholders in the query.
    const values = [timeStamp, chosenDate, chosenRoom, chosenTime, username, notes];

    // Execute the query and pass the values array.
    await db.none(insertQuery, values);
  }

} else {
  console.log("No data found.");
}
```

Noah Garrett's Contribution: At the beginning of the first week I helped fix some issues with the repo not functioning properly because we forgot to add dependencies to the repository. After that I created the login page and the registration page, using the PostgreSQL DB to store the information below which is in the students table.
After that I tried to get a calendar page to work that we would use to display rooms that the users would then be able to book. Due to difficulty getting this to work we ended up using buttons that the user can click to allow them to book the room insead. We made this look pretty using css. After that I ended contributing to the home page by making it so that when you click the buttons to reserve a room they change color and redirect to a cancellation form instead of allowing you to reserve the room. I ended up modifying the post request that you can see below to update the booking status of the rooms when you book them. I also contributed to the project by creating a profile page that allowed the user to delete their account and change their password.



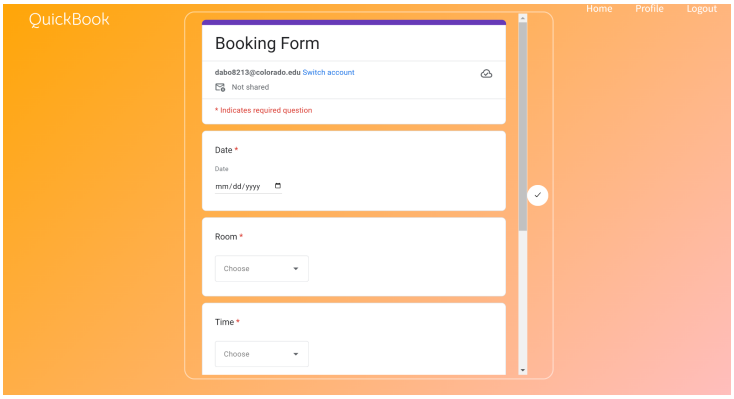

```
app.post("/tableBook", async (req, res) => {


  console.log('in post request' ,tableidl);
  const query1 = `update tableid_to_booked set bookedstatus = true where tableid = ${tableidl} returning * ;`;

  db.one(query1)
  .then(function (results){
    const query1 = `select * from tableid_to_booked ORDER BY tableid ASC;`;
    db.any(query1)
     .then(function (result){
       console.log('!!!!! RESERVE:', result)
       res.render("pages/home", {
         StudentID: req.session.user.studentid,
         first_name: req.session.user.first_name,
         last_name: req.session.user.last_name,
         email: req.session.user.email,
         bookedinfo: result
        // formid: req.body.formid,
```

Daniel Bonnecaze's Contribution:
My contributions were a mix of the front-end UI and the back-end implementation. My first task was creating the registration page. I provided both the ejs file for the front-end and the API endpoints to get to the page and submit the new account into our database. From there I started implementing that Google Forms API into the code. I created the endpoints to redirect to the form and submit the responses. I also worked on fixing the reservations listing page to properly insert room numbers and time slots. Finally, I contributed to fixing the SQL database alongside other group members.

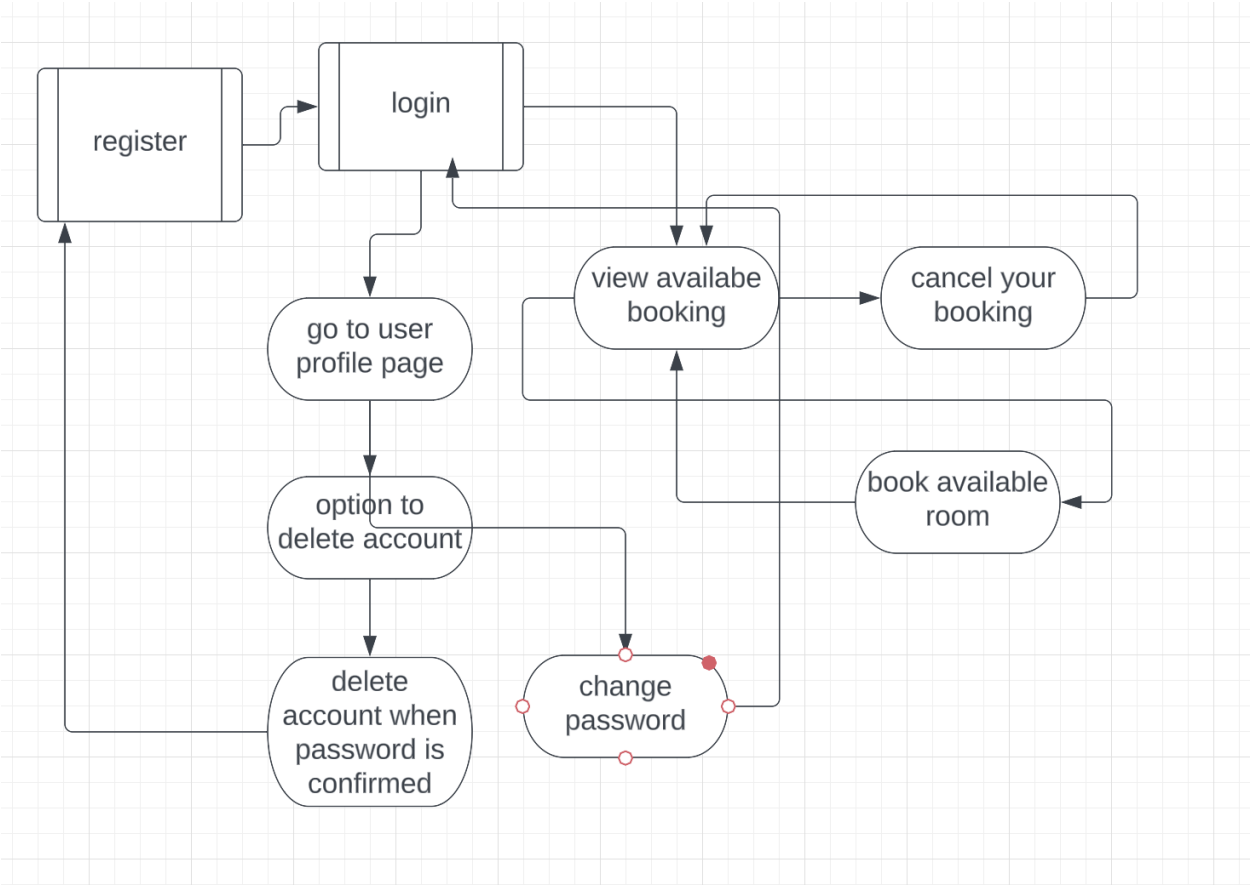
```
axios({
  url: `https://docs.google.com/forms/d/e/1FAIpQLSeFQu96i8thKDPh6chmpaRUTuFvAZkUBRhwTlhWmPOA0pC4iw/viewform`,
  method: 'GET',
  dataType: 'json',
  headers: {
    'Accept-Encoding': 'application/json',
  },
  params: {
    apikey: process.env.API_KEY,
    size: 1,
  },
})
.then(results => {
  //console.log(results.data); // the results will be displayed on the terminal if the docker containers are running //
  res.render("pages/tableBook");
})
.catch(error => {
  // Handle errors
  res.render("pages/tableBook", {
    results: [],
    error: true,
    message: error.message,
  });
});
// }
// else{
//   res.redirect('/cancelBooking');
// }
})
```

```
app.get("/register", (req, res) => {
  console.log('register called')
  res.render('pages/register');
});

// CREATE A POST ROUTE HERE TO CREATE A NEW ACC
app.post('/register', async (req, res) => {
  //hash the password using bcrypt library
  const hash = await bcrypt.hash(req.body.password, 10);

  const info = `insert into students (first_name, last_name, email, StudentID, pwd) values ($1, $2, $3, $4, $5);`;

  db.any(info, [req.body.first_name, req.body.last_name, req.body.email, req.body.StudentID, hash])
    .then((data) => {
      user.first_name = req.body.first_name;
      user.email = req.body.email;
      user.last_name = req.body.last_name;
      user.studentid = req.body.StudentID;
      req.session.user = user;
      req.session.save();
      res.redirect('/home')
    })
    .catch((error) => {
      console.log(error);
      res.redirect(404, "/register");
    })
});
```

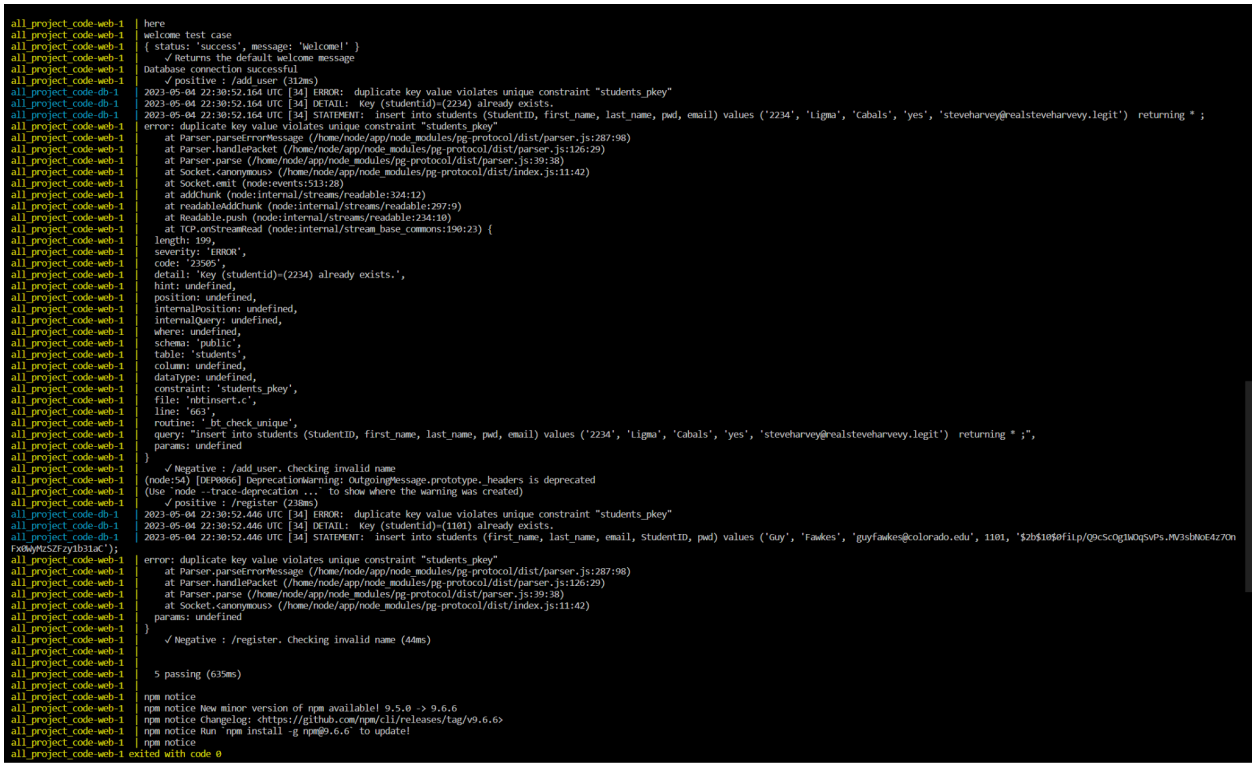
```
<div class="form-box" id="register">
    <div class="form-value">
        <form action="/register" method="POST">
            <h2>Register</h2>
            <div class="inputbox">
                <input type="text" id="first_name" name="first_name" required/>
                <label for="first_name">First Name</label>
            </div>
            <div class="inputbox">
                <input type="text" id="last_name" name="last_name" required/>
                <label for="last_name">Last Name</label>
            </div>
            <div class="inputbox">
                <input type="text" id="email" name="email" pattern="^(.+)@(.+)$" required/>
                <label for="email">Email</label>
            </div>
            <div class="inputbox">
                <ion-icon name="id-card-outline"></ion-icon>
                <input type="text" id="StudentID" name="StudentID" pattern ="[0-9]{1,50}" title = "student id is only allowed
                <label for="StudentID">Student ID</label>
            </div>
            <div class="inputbox">
                <ion-icon name="lock-closed-outline"></ion-icon>
                <input type="text" id="password" name="password" required/>
                <label for="password">Password</label>
            </div>
            <button type="submit">Create an account</button>
            <div class="login">
                <p>Already have an account? <a href="/login">Login</a></p>
            </div>
        </form>
```



Use Case Diagram:



Test results:



Test Result Observations:

Use Case 1: **Basic Booking Form Testing**
What are the users doing?
The user is filling out a Google Form with relevant information and submitting it.

What is the user's reasoning for their actions?
The user wants to book a specific room.

Is their behavior consistent with the use case?
Yes. In order to book a specific room, the user must fill out a Google form.

Did you use that to make changes to your application? If so, what changes did you make?
The google form will appear after the user clicks on their specified room.

Deployment:

- **Local Deployment** Instructions can be found in the ReadMe.md
- **Public Deployment** was a success. When operating we use the following link to host our web server online. http://recitation-014-team-04.eastus.cloudapp.azure.com:3000/