Tab 1

EPICS Service Learning

Jacob Shank

Will Denning

Nathan Szadowski

Rachel Tebbe

Fall 2024

Immigrant Welcome Center

Table of Contents

1. Summary

2. Introduction

3. Requirement Specifications

4. Architecture

5. Design

6. Implementation

7. Project Organization & Management

8. Future Work

9. Acknowledgements

10. Sources

11. Appendix

**Summary**

We worked with the Immigrant Welcome Center in Indianapolis to assist with their English Language Learners through their Pathway to Literacy. Through this program, they assist immigrants as they transition into America. They do this through formal classes of learning English. The Immigrant Welcome Center wanted us to transfer the assessments they provide to their learners to an online e-assessment that could keep track of scores. Our goal was to provide an e-assessment that would best suit the learners in a simple and understandable format.

**Introduction**

**Our Client**

The Immigrant Welcome Center is a nonprofit organization that provides support to non-English speaking immigrants with the goal of integrating them into the Indianapolis community. IWC was founded in 2006 and has grown from just 12 volunteers to a much larger organization that helps and supports hundreds of people each year. We worked with the English language learning assessment team, tasked with helping non-English-speaking immigrants learn English. They accomplish this through assessments focusing on assisting them with reading, writing, speaking, and understanding different elements of American culture.

**Problem Statement and Objectives**

Immigrant Welcome Center’s Pathway to Literacy Program works with adult English language learners. They do this through class for adults who have less than 6 years of formal education. In the program, learners are assessed based on their level, whether level A, B, C, or D. Each of these levels are broken down into different categories such as Oral, Reading, Writing, Phonics, and Numeracy. Currently, these assessments are all administered on paper and require more hands-on from the teachers than what could be.

Our team was tasked with creating an application to administer assessments to adult English language learners. The application must have a simple login functionality for both teachers and students. Teachers should be able to view the results from a student's test including their score and which questions were answered incorrectly. The assessment has 5 levels each with multiple tasks. The application should be built in a manner that allows inclusion of more levels and tasks in the future. The primary focus was implementing the level B assessment. The application must have a very simple UI and not utilize any textual instruction. Both of these features help non-English speakers navigate the assessment. The client also wants the application to be accessible and free to use for other organizations around the world.

**Motivation and Rationale**

Being able to work with The Immigrant Welcome Center was something as a group that we thought to be impactful. The Immigrant Welcome Center's overall mission is something that we found to be important as they were helping immigrants directly in our community of Indianapolis. Not only that, but they also want to make an impact beyond just our community which is something that we wanted to be a part of making possible. On top of that, we as a group felt that this project would be challenging and provide us with meaningful skills.

**Approach**

Our team went about this project with the Immigrant Welcome Center’s expectations at the forefront. The semester started with the team gaining a full understanding of the realm of the project and all the Immigrant Welcome Center can provide for immigrants. From there our team kept our options open on what the best way to go about the project was so that we would be able to best fulfill the Immigrant Welcome Center’s needs. As the semester progressed there was a plethora of communication with the Immigrant Welcome Center to ensure that our team’s understanding was correct and that our application would be suited for the learners. This meant verifying that the technology that our team was using would perform correctly as well as give us the end result that the Immigrant Welcome Center asked for.

**Glossary and Terminology**

**Node.js**: Open-source JavaScript runtime environment for running web applications outside the browser.

**NPM (Node Package Manager)**: A package manager for JavaScript applications, included with Node.js.

**Git**: A source code management tool for tracking code changes, essential for collaborative projects.

**AWS (Amazon Web Services)**: Cloud hosting service offering tools like EC2, RDS, Aurora, and S3.

**Aurora**: AWS database hosting service compatible with MySQL and PostgreSQL, offering faster performance.

**EC2 (Elastic Compute Cloud)**: AWS virtual computer service used for hosting databases and application code.

**RDS (Relational Database Service)**: AWS service for managing SQL databases.

**PostgreSQL**: Open-source SQL database system with extensive compatibility and support.

**React**: JavaScript library for building user interfaces and connecting front-end to back-end via APIs.

**API (Application Programming Interface)**: A set of rules enabling communication between software applications.

**S3 (Simple Storage Service)**: AWS storage solution for files like images and audio.

For more information on all of these, please view the installation and information guide in the Github.

**Organization of the Report**

Throughout this report, you will find an overview of the project for the Immigrant Welcome Center. There will be detailed explanations about the expectations and requirements of the application. On top of that, there will be background on what tools were used for development and why such tools were used. It will also go into detail about the current application's features and how to use it. Lastly, it will give some possible recommendations to future groups on how they can continue to improve the application for the Immigrant Welcome Center.

**Requirements Specifications**

**Functional and Non-Functional Requirements**

At the beginning of the semester, the Immigrant Welcome Center’s Assessment Team laid out to us in detail what they envisioned for the final application to be. The biggest thing that they wished for was a platform that would allow the learners to be able to take their assessments online. For this semester, they decided to make things simpler and for us to solely focus on their Level B assessment. To them, this would ideally be a website that would be easy to get to and be available on any type of device. With that being said, these learners are still going through the process of becoming fluent in English and there was a significant need for the application to be very simplistic. There was also a need for each of the instructions for the different test tasks to be spoken in the learner's native language. This meant that when on the application, it would allow an easy student login that would then lead straight to the test. Each student would also be able to pick from a themes list so their questions would relate to that theme. From there, they wanted each test question page to be very minimal with just the images or words and the button for the instructions to play. Regarding how the test would flow, they expressed that a student is expected to get a certain number of questions from each category correct to move on to a different category.

They wanted not only a student side of the test but also a teacher side of the application. In this, they wanted each teacher to be able to see all of their students, look at their test scores, and see what they got right or wrong.

Finally, they wanted this application to be accessible beyond the Immigrant Welcome Center here in Indianapolis. They hope it can be used nationwide with other programs that help immigrants and their immersion into the United States.

**Assumptions and Constraints**

Some of the assumptions that the team held was that we were going to do our best to provide the Immigrant Welcome Center with an application that met their expectations. Going into this, our team knew that it would not be an easy feat, but we were willing to take on the challenge and at least set a solid foundation for future groups to build off of. Another assumption is that we need to make this application as accessible as possible. This is due to the fact of language barriers of the learners and being able to make the application as simple and straightforward as possible. Doing this will only help to enhance the overall experience for the test taker. There is also the assumption that the ultimate goal is to be able to use this application nationwide and for other programs to be able to access and use the platform. Lastly, there was the assumption that the Immigrant Welcome Center would provide us with all of the proper voice recordings in various languages, pictures for questions, and finally the sample bank for the test. All of this the Immigrant Welcome Center was able to provide us with to be able to properly develop the application.

There were several different constraints that our group was faced with. The biggest one was a time constraint. Due to other demands of classes, there were times when it was challenging to find extra time to work on the application. Finally, each team member had different schedules which led to difficulties in always being able to meet up. Not only was time an issue, but there was a steep learning curve for our group. The applications and languages that were used in this project were new to our team. This meant that time needed to be dedicated to learning them before being able to implement them. Not only this, but it presented difficulties when errors were thrown due to unfamiliarity with the application. Despite these challenges, our team was able to work together to take on what seemed at times a daunting task.

**Architecture**

From an architectural perspective, the application conforms to a three-tier model. The front end (presentation tier) executes in the browser, the backend (logic tier) runs on Node.js/Express servers, and the database (data tier) resides in PostgreSQL. The tiers communicate over HTTP(S) using JSON payloads. JWTs are used for stateless authentication, where the front end stores the token (often in memory or secure HTTP-only cookies) and includes it in the Authorization header for each API call. The backend validates the token on each request, enabling scalable authentication flows without maintaining a server-side session state.

**Backend**

The backend is built on Node.js and Express, with PostgreSQL serving as the data store. At its core, the database schema is designed around two primary domains: user management and a testing system. The user management domain encompasses tables such as “users” (containing common user attributes, role identifiers, and authentication credentials), “teachers” (storing extended attributes specific to teacher roles), and “students” (storing language preferences and proficiency levels, categorized A-D for adaptive learning scenarios). The testing domain includes “tests” (defining metadata and parameters for assessments created by teachers), “questions” (supporting various formats, including multiple-choice, picture-based vocabulary questions, sequence ordering, fill-in-the-blank, and audio-based listening questions), “test\_assignments” (managing test distribution and scheduling for student cohorts), “student\_tests” (tracking individual student attempts, scores, and attempt timestamps), and “answers” (capturing raw student responses for evaluation and analytics).

On a technical level, the database layer uses PostgreSQL features such as foreign keys, ENUM types, and JSONB columns. Foreign keys maintain relational integrity across dependent entities, ENUMs enforce controlled vocabularies for fields like user roles or question types, and JSONB fields store semi-structured data (e.g., question configuration parameters) that may evolve over time without requiring complex schema migrations. Indexing strategies (e.g., B-tree indexes on user IDs, composite indexes for test-query patterns) improve query performance, while primary keys and uniqueness constraints ensure data consistency. The application uses node-postgres, with connection pooling managed by a pool instance that handles concurrent requests efficiently. Parameterized queries are employed to prevent SQL injection attacks, and transactions ensure atomicity of multi-step operations. For example, when a teacher creates a new test with associated questions, the process runs within a database transaction so that any failure when creating related entities rolls back the entire operation, preserving data consistency.

The Express.js backend employs a structured approach to route organization, separating endpoint definitions by domain. For instance, “auth.js” manages authentication flows (registration, login via JWT issuance, token refresh, logout), “tests.js” handles CRUD operations for tests and their associated questions, “assignments.js” manages linking tests to students, “students.js” provides endpoints for retrieving and updating student records, and “media.js” supports file uploads (images, audio) and secure retrieval of media content. Middleware layers enforce JWT-based authorization and role-based access control, ensuring that teachers cannot access student-only endpoints and vice versa. Errors are consistently handled through a centralized error-handling middleware that captures database constraint violations, missing parameters, or authentication errors, returning standardized JSON error responses and appropriate HTTP status codes.

From a configuration standpoint, the backend leverages environment variables for database credentials, JWT secrets, and port assignments. The “.env” file sets variables for DB\_USER, DB\_PASSWORD, DB\_HOST, DB\_PORT, DB\_DATABASE, and JWT\_SECRET. This approach supports environment-specific configurations (development, staging, production) without requiring code changes. Logging is handled through console output in development mode, and it can be extended using libraries like Winston or Bunyan in production to capture structured logs and integrate them with external log management systems.

**Frontend**

The front end is implemented as a Single-Page Application (SPA) using React (with TypeScript for type safety and improved developer tooling). Vite serves as the build and development environment, providing fast incremental builds and hot module replacement. The front-end application architecture employs React Router for client-side routing, allowing protected routes that enforce role-based redirects. For example, teacher-only pages (like test creation screens and analytics dashboards) require the user to be authenticated as a teacher. React Context and custom hooks manage global application state, particularly user authentication state and role information. Axios is used as an HTTP client for API requests, providing standardized request/response interceptors for error handling, token injection, and automatic retry logic on certain status codes if required.

The frontend codebase follows a clear directory structure:

* “components” contains reusable UI elements (e.g., buttons, form inputs, modal dialogs) that can be composed into larger views.
* “pages” corresponds to top-level screens, such as a TeacherDashboard or StudentTestView. Each page integrates multiple components and coordinates data fetching, state management, and navigation flows.
* “api” centralizes endpoint definitions and API call functions to maintain a clean separation between presentation logic and data fetching logic. Here, TypeScript types define the shape of payloads and responses, ensuring that data handling is type-safe and reduces runtime errors.
* “context” includes global states such as authentication context, which stores user tokens, role data, and refresh logic. Hooks like “useAuth” provide a consistent interface for components that need user credentials or login status.
* “types” defines shared TypeScript interfaces, enums, and type aliases, enabling type consistency across the entire application.

Styling and layout follow a modular approach using CSS modules or styled components, ensuring that class names and styles do not leak or collide. Responsive design principles are applied, allowing the application to adapt to various screen sizes and ensuring a usable experience on tablets, laptops, and desktop monitors. Accessibility considerations include proper ARIA attributes, keyboard navigability, and descriptive alt text for images.

**Design Details**

The system is designed with scalability and maintainability in mind. Horizontal scaling on the backend can be achieved by running multiple Node.js processes behind a load balancer, as sessions are stateless and tokens are self-contained. Connection pooling to PostgreSQL ensures efficient resource usage under higher concurrency. The frontend build output (optimized JavaScript, CSS, and media assets) can be served via a content delivery network (CDN) for performance and global distribution.

A detailed user guide is provided to help developers set up and run the application. Prerequisites include Node.js, PostgreSQL, and Git. After cloning the repository, developers create and initialize the PostgreSQL database, then populate environment variables in the backend’s “.env” file. The backend is started with “npm start” (using nodemon for hot reloading in development), and the frontend is launched with “npm run dev” in the frontend directory. Developers can confirm that the system is operational by hitting a health check endpoint on the backend (e.g., GET /api/health) and verifying that the frontend’s login page renders correctly. Common tasks like adding dependencies, running linters, building production bundles, and troubleshooting database connection or port conflicts are documented for convenience.

Overall, the backend’s transaction-driven, parameterized query model ensures data integrity and security, while the frontend’s typed React codebase and well-structured directories facilitate maintainability and scalability. Together, these details equip developers with the knowledge needed to understand, modify, and extend the system as it evolves.

Our team wanted to ensure that there was a sense of neatness and clarity in the work of the project. We wanted to ensure that everything was not only clear for our team and Immigrant Welcome, but more importantly for future groups. We used a shared drive so everyone in the team had access to important documents about the project. A GitHub was also used to ensure a central location for all of the code. The biggest source of team organization was communication among members to ensure that everyone was on the same page and understanding of the current state of the project.

**Project Organization and Management**

**Team Organization**

Due to the fact that this would be a steep learning curve, our team all collaborated to help develop this application. Each of the team came in with limited knowledge and experience in the languages and technologies that were used. Ultimately, we were able to use various resources to gain an understanding of what was being used. Jacob took on the role of team leader and ensured that everyone was aware of their responsibilities and the overall plan of the project. He also ensured that we had a client relationship through maintaining communication with the Immigrant Welcome Center. Nathan took on the role of a developer and really stepped up and took the lead with this. He was able to grasp new concepts and tools quickly and was able to help the other team members along the way. Will was able to work on documentation of the project to ensure that the Immigrant Welcome Center and future groups would be able to understand what we had done and also made a guide on how to install everything we used and information about important applications. Rachel was able to use her knowledge and assist where needed in the development of the application.

**Contributions of Each Team Member**

**Jacob**: Took on the role of team leader as well as the client liaison. This included managing meetings and communication with the client. He also ensured the correct layout of the database. He played a role in deployment of the application on AWS Amplify before the team took the project in a different direction.

**Nathan**: Took on the role of primary developer and contributed extensive amounts of time and effort working to learn and develop the entire application.

**Will**: Took on the role of documentation, crafted installation and informational guides on applications/technologies used, and ensured all of the code was understandable for future users.

**Rachel**: Took on the role of ensuring that things were correctly written up for future groups to have a solid understanding of what the state of the application is.

**Division of Work**

This project has been collaborative, as each team member is learning as we go. Some members came into the semester with more background knowledge than others, which allowed those with more experience to teach the other members along the way.

Once we decided on our method, development began until there was a bump in the road. At this point, it was late in the semester, and it was discovered that the tools we were using were not the best. After Nathan changed the entire tech stack in two days, things were able to get back up and running and even make more progress than what was had with the original tools.

**Project Management Process**

As a team, there were many ways that we managed the project’s process. First off, we met on Tuesdays and Thursdays during class times for 75 minutes. There was also constant communication among members when there was a need for clarification or assistance. All team members also regularly attended class and any meetings that were set up outside of class time. Along with team communication, there was an emphasis on client communication. For us, this meant meeting with the Immigrant Welcome Center every other Thursday on Zoom during class time. This allowed us to present to them where we were at in the project as well as being able to ask or answer any questions. Outside of this, our team used a shared Google Drive for important documents and collaboration on weekly deliverables. There was also Slack which we used for communication among members and holding important access information. Along with that Github was used to properly hold code.

**Future Work**

**Explain work that needs be accomplished in the future**

Moving forward different aspects of this application will need more attention. The biggest area of work would be working to continue to develop the front end of the application.

Future groups will also need to work on implementing the audio and images into the questions. This is crucial to making the questions as accessible as possible to the learners. Beyond this being able to add in all of the questions so the Level B assessment is complete. From there teams will be able to easily add in all of the other levels of assessments.

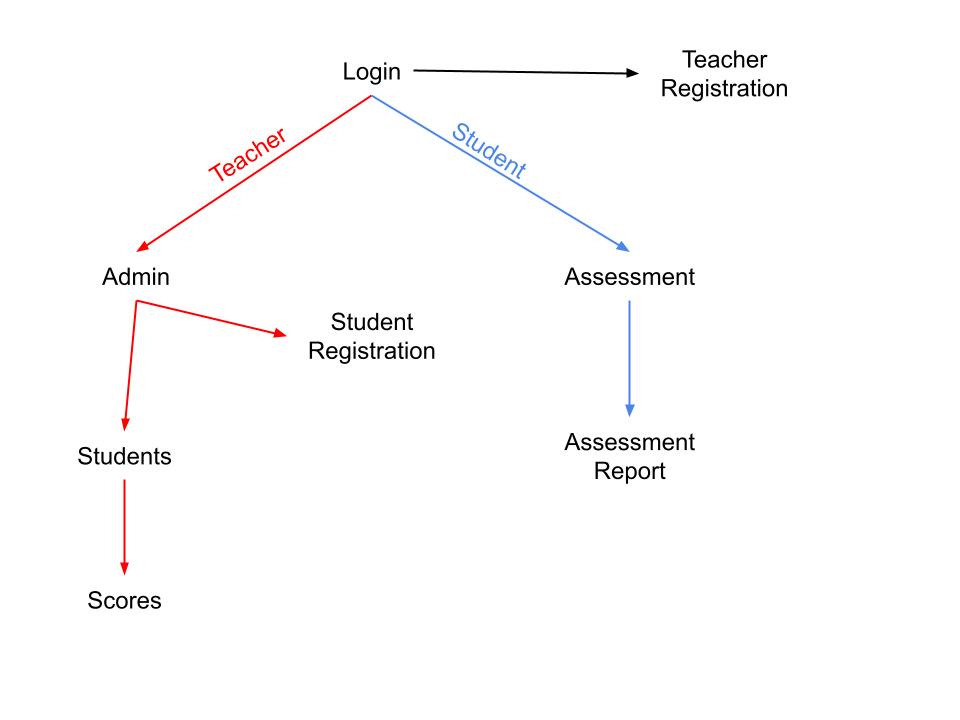
In the future, groups will need to focus the application. This means tailoring the application more specifically to the needs of the client. Due to last-minute complications, the final deliverable is an application that is fairly general or standard. This means that the application may need to be pruned in some areas, or expanded in others.

**Include any additional functionality or requirements that can be implemented by the next team**

The application needs to create more assessments from the front end for certain users/user types. The application’s ability to take the assessment is underway but needs some tweaking. The UI is currently very standard but IWC may want some parts of the application (specifically for students) to be simpler. For example, the registration process has a flow and a number of fields that the client may want changed.

Further functionality should also be a top priority. The application currently cannot create or administer a test. The initial goal was to store enough information about a test question (like level and task) to discern the order of questions from the data their record holds to build a test. When considering the application at scale, the application needs a more in-depth system for assigning a student to a teacher. This could be done by re-structuring the registration system such that a teacher registers a student and then that student is automatically assigned to the teacher.

The registration process currently has a flow that is not desired by the clients. Currently anyone with access to the website can register as a student or a teacher. The registration should flow more like the diagram below, where teachers register students, and students are assigned specifically to that teacher.



**Sources**

<https://immigrantwelcomecenter.org/>

<https://immigrantwelcomecenter.org/what-we-do/english-language-learning/>

**Appendix**

**WEEKLY STATUS REPORT (WSR)**

September 24, 2024

**TO:** Dr. Haque, Will Denning, Rachel Tebbe, Nathan Szadowski

**FROM:** Jacob Shank,

**SUBJECT:** 9/17 - 9/24

1. **RED FLAGS:** Previous groups used Xcode as their Development Environment but we do not all develop on macOS. We don't know what environment to continue in.
2. **ISSUES:** Using Swift on non-macOS devices
3. **ACCOMPLISHMENTS (dates):**

9/17/24

* Finalized group for IWC
* Made team group chat

9/19/24

* Reached out to Ginger with IWC to set up a meeting
* Decided team member tentative roles
* Set up a team Trello Board and Slack Server
* Acquired the previous team’s GitHub

1. **ACTION ITEMS FOR FOLLOWING WEEK (dates):**

* Meet with IWC on Thursday (9/26) to go over the project and their expectations
* Meet with Previous EPICS team (have not set time/date)
* Finalize development platforms (all team members)

**WEEKLY STATUS REPORT (WSR)**

October 8th, 2024

**TO:** Dr. Haque, Will Denning, Rachel Tebbe, Nathan Szadowski

**FROM:** Jacob Shank

**SUBJECT:** 9/26 - 10/8

1. **RED FLAGS:** Starting from scratch, nothing to work off of
2. **ISSUES:** Lack of experience with AWS
3. **ACCOMPLISHMENTS (dates):**

9/26/24

* Met with the client and discussed expectations
* Decided to go in a different direction than the previous group
* Agreed with IWC to use AWS to make a website
* Set up bi-weekly meeting times

10/1/24

* Presented our work so far to the class
* Rated presentations of other groups

10/3/24

* Met with advisors who told us about Butler’s personal domain
* We were advised not to worry about AWS at the moment

10/4/24

* Created prototype website hosted on Nathan’s personal AWS account and Github account
* Created mock login and register page

1. **ACTION ITEMS FOR FOLLOWING WEEK (dates):**

* Gain an understanding of Butler’s personal domain that students can access
* Get in contact with EPICS consultants for additional assistance
* Get a single question to appear on our server
* Explore backend technologies
* Meet with IWC on 10/10

**WEEKLY STATUS REPORT (WSR)**

October 22nd, 2024

**TO:** Dr. Haque, Will Denning, Rachel Tebbe, Nathan Szadowski

**FROM:** Jacob Shank

**SUBJECT:** 10/8 - 10/22

1. **RED FLAGS:** We may need a lot of different classes in Java for each part of the assessment
2. **ISSUES:** Implementing Java classes in JavaScript, deciding who should focus on specific parts of development. Unsure how to proceed in connecting cloud-based databases to amplify applications.
3. **ACCOMPLISHMENTS (dates):**

10/8/24

* Talked with Dr. Haque about the feasibility of AWS
* Ultimately decided to attempt to learn how to use
* Reviewed feedback from other students on our presentation

10/10/24

* Met with IWC on Zoom to discuss current progress
* Gained access to the slideshow they use for assessments
* Came up with ideas for how to implement a database for the assessment
* Discussed how we could use Java classes with javascript
* Reached out to an advisor to discuss how we can implement a database in AWS

10/14/24

* Met with Nate Partenheimer to discuss AmazonRDS
* The team learned how to configure the database to communicate without amplified application
* Created an ERD to begin work in developing the database
* Presented our progress so far to the class

10/20/24

* Wrote DDL for database

1. **ACTION ITEMS FOR FOLLOWING WEEK (dates):**

* Start on back-end development
* Make classes on Java for each part of the assessment
* Create an Amazon RDS server running MySQL
  + Configure settings to allow communication between RDS server and Amplify application
  + Implement DDL
  + Begin early database class development

**WEEKLY STATUS REPORT (WSR)**

October 29th, 2024

**TO:** Dr. Haque, Will Denning, Rachel Tebbe, Nathan Szadowski

**FROM:** Jacob Shank

**SUBJECT:** 10/22 - 10/29

1. **RED FLAGS:** Unable to have free AmazonRDS Database for testing purposes without paying
2. **ISSUES:** Having Java classes transfer to javascript. Working on getting the website to function with SQL rather than json.
3. **ACCOMPLISHMENTS (dates):**

10/22/24

* Found possible documentation tool (compodoc)

10/24/24

* Met with IWC to explain our progress
* Installed Angular on everyone's computers
* Installed necessary Angular package managers

1. **ACTION ITEMS FOR FOLLOWING WEEK (dates):**

* Begin back-end development on everyone’s computers
  + Continue
* Ensure that compodoc documentation can work on our code
* Document process for installing all packages and programs to run the current project on localhost
* Get a single question to appear
* Find a way to transfer Java classes to javascript

**WEEKLY STATUS REPORT (WSR)**

October 29th, 2024

**TO:** Dr. Haque, Will Denning, Rachel Tebbe, Nathan Szadowski

**FROM:** Jacob Shank

**SUBJECT:** 10/29 - 11/4

1. **RED FLAGS:**
2. **ISSUES:** Having Java classes transfer to JavaScript. Working on getting the website to function with SQL rather than json.
3. **ACCOMPLISHMENTS (dates):**

10/29/24

* Installed and ran Compodoc documentation tool successfully
* Began writing a step-by-step guide to get everything needed for angular installed for future groups

10/31/24

* Successfully got the application to work with SQL

1. **ACTION ITEMS FOR FOLLOWING WEEK (dates):**

* Implement ddl and begin work on backend programming
* Continue Documenting process for installing all packages and programs to run the current project on localhost
* Get a single question to appear
* Meet with IWC and get our questions answered
  + Questions about test structure

**WEEKLY STATUS REPORT (WSR)**

November 19th, 2024

**TO:** Dr. Haque, Will Denning, Rachel Tebbe, Nathan Szadowski

**FROM:** Jacob Shank

**SUBJECT:** 11/6 - 11/19

1. **RED FLAGS:** Unsure how to record session data (username, user language, user type, etc.)
2. **ISSUES:** MySQL not working on a mac
3. **ACCOMPLISHMENTS (dates):**

11/06/24

* Met with IWC to discuss progress so far
* Got answers on writing questions, user access keys, and question format

11/12/24

* Presented our progress leading up to Spring 2 and received feedback
* Wrote Java files that are capable of registering a user into an sql database

11/14/24

* Installed and configured MySQL with the team’s Windows computers
* Documented how to install and run angular and spring-boot for future teams

1. **ACTION ITEMS FOR FOLLOWING WEEK (dates):**

* Begin in-depth development of front end
* Connect front end to the backend
* Finalize backend
* Find documentation tool for SQL

**WEEKLY STATUS REPORT (WSR)**

December 10th, 2024

**TO:** Dr. Haque, Will Denning, Rachel Tebbe, Nathan Szadowski

**FROM:** Jacob Shank

**SUBJECT:** 12/5-12/10

1. **RED FLAGS:** Tech stack in process of being changed
2. **ISSUES:** Having an issue with deployment that is yet to be resolved
3. **ACCOMPLISHMENTS (dates):**

12/10/24

* Met with IWC Assessment team and created AWS account
* Added Nathan’s personal account as a member of the IWC AWS organization
* Set up PostgreSQL server and React front end hosted on Vercel (temporary cloud alternative to aws). Tech stack is now likely PERN.

12/11/24

* Emailed IWC Assessment team a summary of thursday’s meeting
* Decided to create GitHub account ourselves for IWC

1. **ACTION ITEMS FOR FOLLOWING WEEK (dates):**

* Work out exact cost for all AWS services (Rachel, 11/12)
* Create and send video instructions on settings AWS access level (Nathan/Will, 11/12)
* Deploy React test repository on Amplify (Jacob, 11/12)