

CS 4850 – Senior Project Sections 02 & 03, Spring 2025

24-T1 OWL JUDGE

Team Members

Name

Lumiere Nathan Kue

Nathanael Fokou

Bill Roan Simo

Minh Quang Vu

Advisor / Instructor: Sharon Perry

Role: Facilitate project progress; advise on project planning and management.

April 28th, 2025

Website: <https://owl-judge-21-t1.github.io/project/>

GitHub: <https://github.com/OwlJudge-Mobile-App/OwlJudge-App>

STATS AND STATUS

| | |
|------------------|------|
| Lines of Code | 2500 |
| Components/Tools | 11 |
| Estimated Hours | 450 |
| Actual Hours | 500 |

Table of Contents

| | |
|---|-----------|
| I. Introduction | 3 |
| II. Requirements..... | 3 |
| 1. Functional Requirements: | 3 |
| 2. Non-Functional Requirements: | 4 |
| III. Analysis / Design | 4 |
| 1. Analysis..... | 4 |
| 2. Design..... | 7 |
| IV. Development..... | 18 |
| 1. Backend Development: | 18 |
| 2. Database Connection: | 18 |
| 3. Frontend Development: | 20 |
| 4. Security: | 20 |
| 5. Setting Up the Project | 20 |
| V. Test Plan and Test Report | 22 |
| VI. Version Control | 23 |
| VII. Conclusion | 23 |
| VIII. Appendix..... | 23 |
| 1. Project Plan: | 23 |
| 2. Training Verification (if applicable): | 26 |

I. Introduction

This document provides an overview of the development, design, and implementation of the **OwlJudge Mobile Application**, a system created for digital judging of projects or events. The system automates the traditionally manual process, improving efficiency, accuracy, and transparency. The app aims to monitor the judging process, improve accuracy, and offer real-time updates for judges, participants, and administrators. This app is developed using React Native for the mobile app, PostgreSQL for database and Node.js for the backend. The app ensures a responsive and secure experience across mobile platforms.

Goals and Objectives:

- Automate the event judging process to increase efficiency.
- Provide a user-friendly interface for judges and administrators.
- Ensure secure authentication and authorization through industry-standard protocols.
- Integrate PostgreSQL for reliable data storage and Redis for performance optimization.
- Enable real-time monitoring and analytics to track event progress.

Key Features:

- Secure login with role-based access control for Admins, and Judges.
- Event and judge management with customizable scoring criteria.
- Real-time score submission and aggregation.
- Real-time ranking and performance analytics for event administrators.

II. Requirements

1. Functional Requirements:

1. User Management & Authentication:

- Secure login for judges, admins, and participants.
- Role-based access control (Admin, Judge).
- Password recovery and secure password management.

2. Event Management:

- Admins can create, edit, and delete events.
- Bulk import of judge and project data.
- Event progress tracking for administrators.

3. Project & Judge Assignment:

- Automated/manual assignment of judges to projects.
- Notifications to judges upon assignment.

4. Digital Scoring System:

- Real-time score input and submission by judges.
- Auto-save scores periodically.
- Lock scores upon submission to prevent unauthorized changes.

5. Real-Time Score Aggregation & Ranking:

- Scores are aggregated in real-time as judges submit evaluations.
- Tie-breaker mechanisms for scoring conflicts.
- Real-time leaderboard and ranking for projects.

6. Reporting & Analytics:

- Real-time event analytics for administrators.
- Export of results in CSV/PDF format.

7. Security & Data Protection:

- Data encryption at rest and in transit.
- Audit logs for administrative actions and judge activity.
- Authentication using OAuth and JWT.

2. Non-Functional Requirements:

1. Performance:

- System must handle at least 100 concurrent users without degradation.
- Response times for score submission should be less than 2 seconds.

2. Scalability:

- multiple concurrent events without performance issues.

3. Usability:

- Mobile-first responsive design, ensuring the application works smoothly on every mobile device.

4. Reliability:

- System availability should be 99.9% during competition hours.
- Automated backups to prevent data loss.

5. Security:

- Compliant with FERPA and university IT security policies.

III. Analysis / Design

1. Analysis

The **OwlJudge Mobile App** is designed using a **microservices architecture**. It is divided into several modular components:

Security Design:

- **Authentication & Authorization:** Implemented using **OAuth 2.0** and **JWT** to ensure secure user login and role-based access control.
- **Encryption:** Data is encrypted at rest and in transit using **TLS 1.3**.

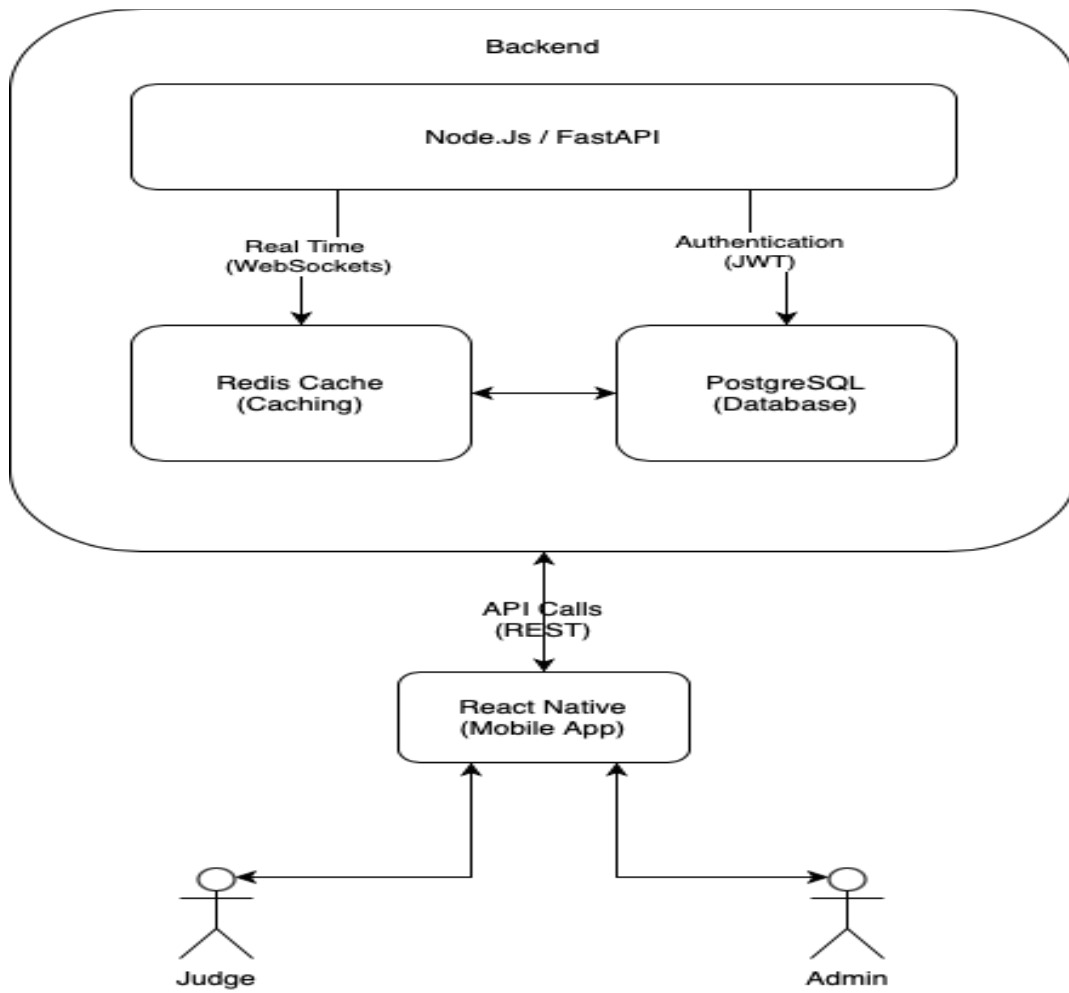
System Architecture:

- **Frontend:** Developed using **React Native**, ensuring a responsive, mobile-friendly interface for judges and administrators.
- **Backend:** Built using **Node.js** (Express) or **Python (FastAPI)** to provide high-speed asynchronous processing.
- **Database:** **PostgreSQL** for structured data storage (users, events, scores), **Redis** for caching frequently accessed data.

System Components:

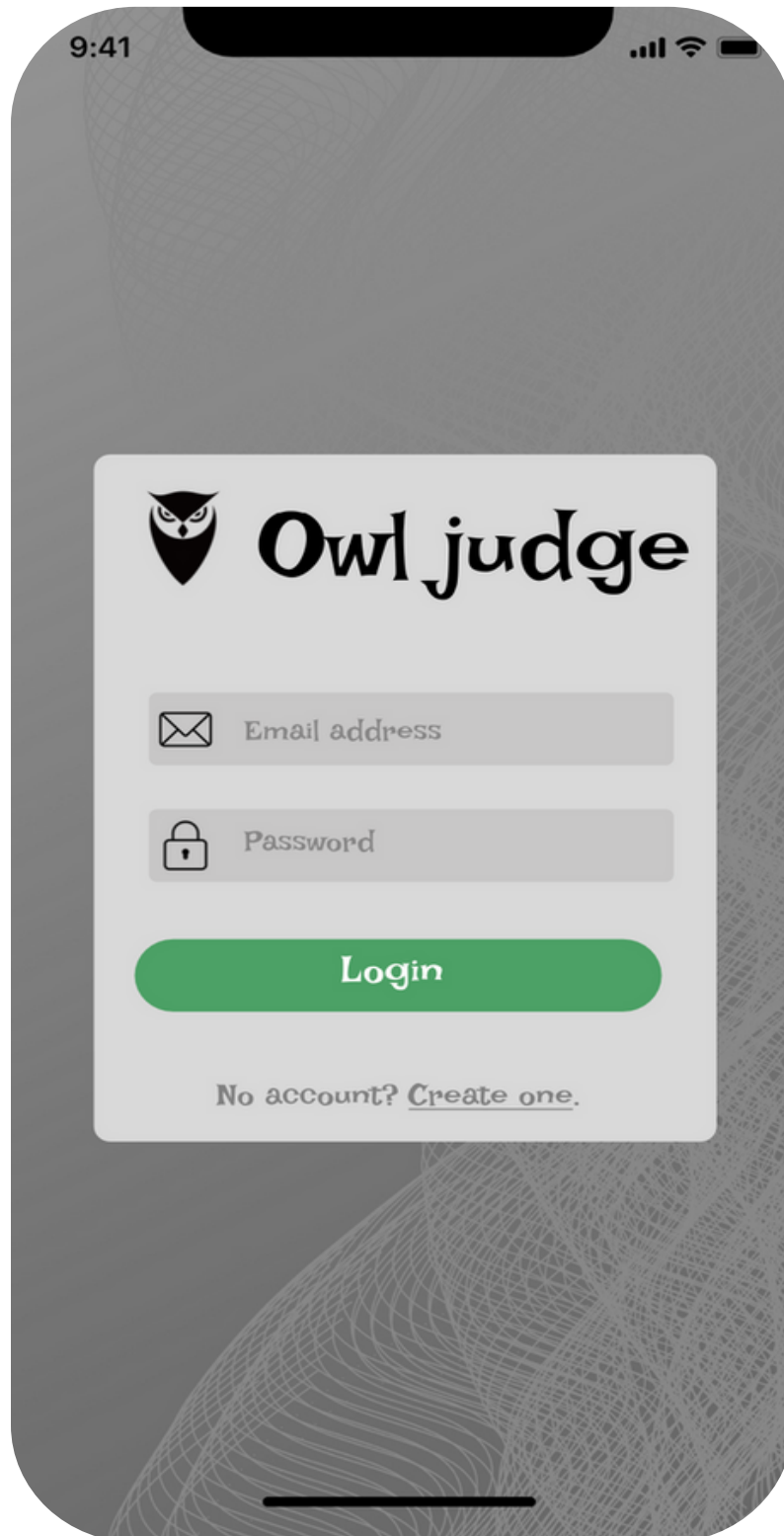
1. **User Management Service:** Handles login, registration, and user roles.
2. **Event Management Service:** Manages event creation, judge assignments, and event tracking.
3. **Project & Judge Assignment Service:** Ensures judges are assigned to projects in a fair and balanced manner.
4. **Scoring System:** Provides judges with an interface to input and submit scores.
5. **Reporting & Analytics Module:** Aggregates data and generates real-time reports for administrators.

System Architecture Diagram:




2. Design


- Login:




A mobile app login screen for 'Owl judge'. The background is a dark gray with a subtle, intricate line pattern. At the top, the status bar shows the time '9:41', signal strength, Wi-Fi, and battery. The login form is a light gray rounded rectangle in the center. It features the 'Owl judge' logo (an owl head icon) and the text 'Owl judge' in a serif font. Below the logo are two input fields: 'Email address' with an envelope icon and 'Password' with a lock icon. A green 'Login' button is positioned below the fields. At the bottom of the form, it says 'No account? [Create one.](#)'.

9:41

 Owl judge

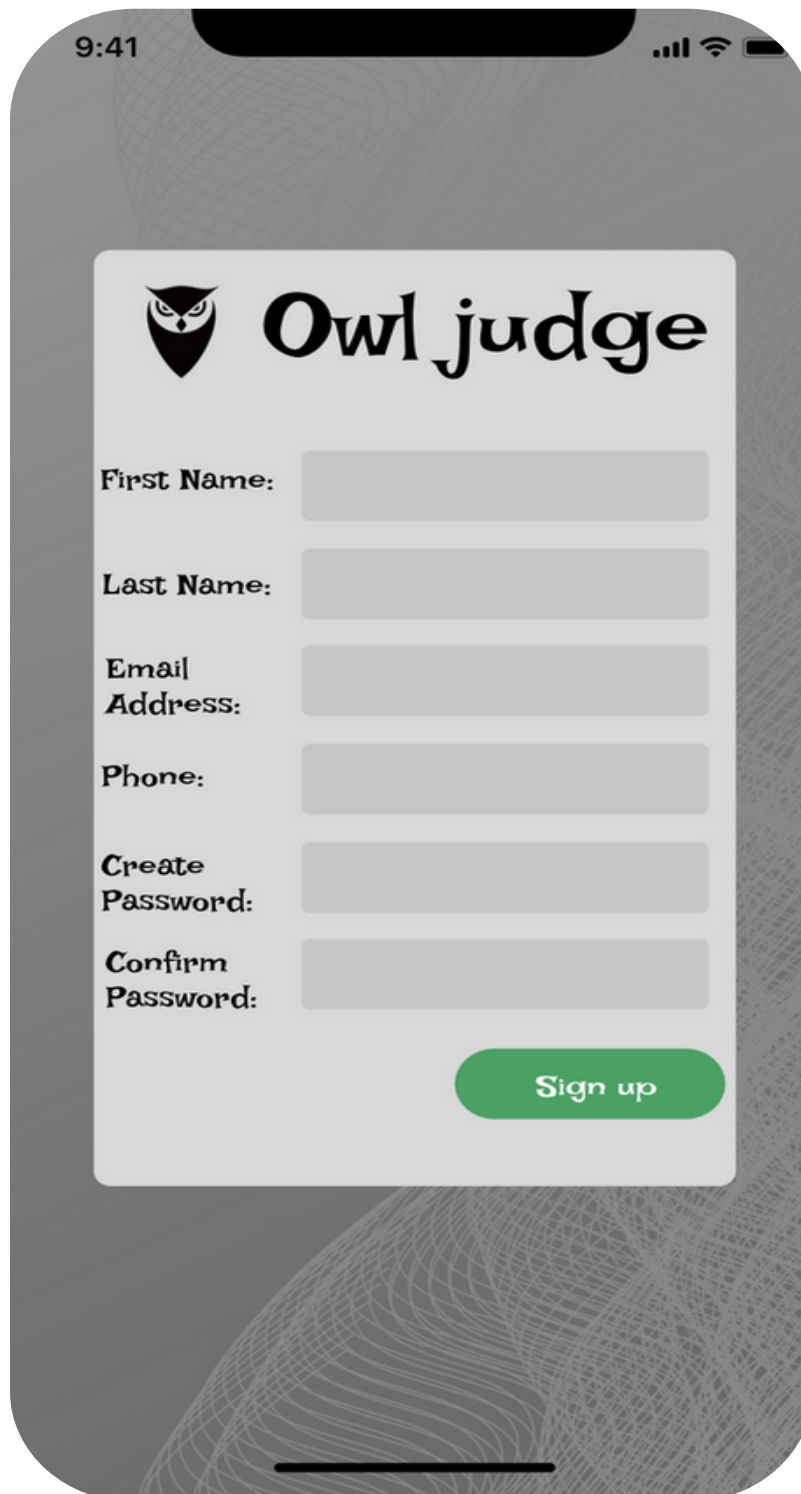
 Email address

 Password

Login


No account? [Create one.](#)

- Create an account:



A mobile app interface for 'Owl judge' showing a sign-up form. The background is a dark grey with a subtle, intricate line pattern. The form is a light grey rounded rectangle in the center. At the top left of the form is an owl logo, and to its right is the text 'Owl judge' in a serif font. Below the logo and text are six input fields, each with a label to its left: 'First Name:', 'Last Name:', 'Email Address:', 'Phone:', 'Create Password:', and 'Confirm Password:'. At the bottom right of the form is a green rounded button with the text 'Sign up' in white. The phone's status bar at the top shows the time '9:41', signal strength, Wi-Fi, and battery icons. A home indicator bar is at the very bottom.

9:41

 **Owl judge**

First Name:

Last Name:

Email Address:

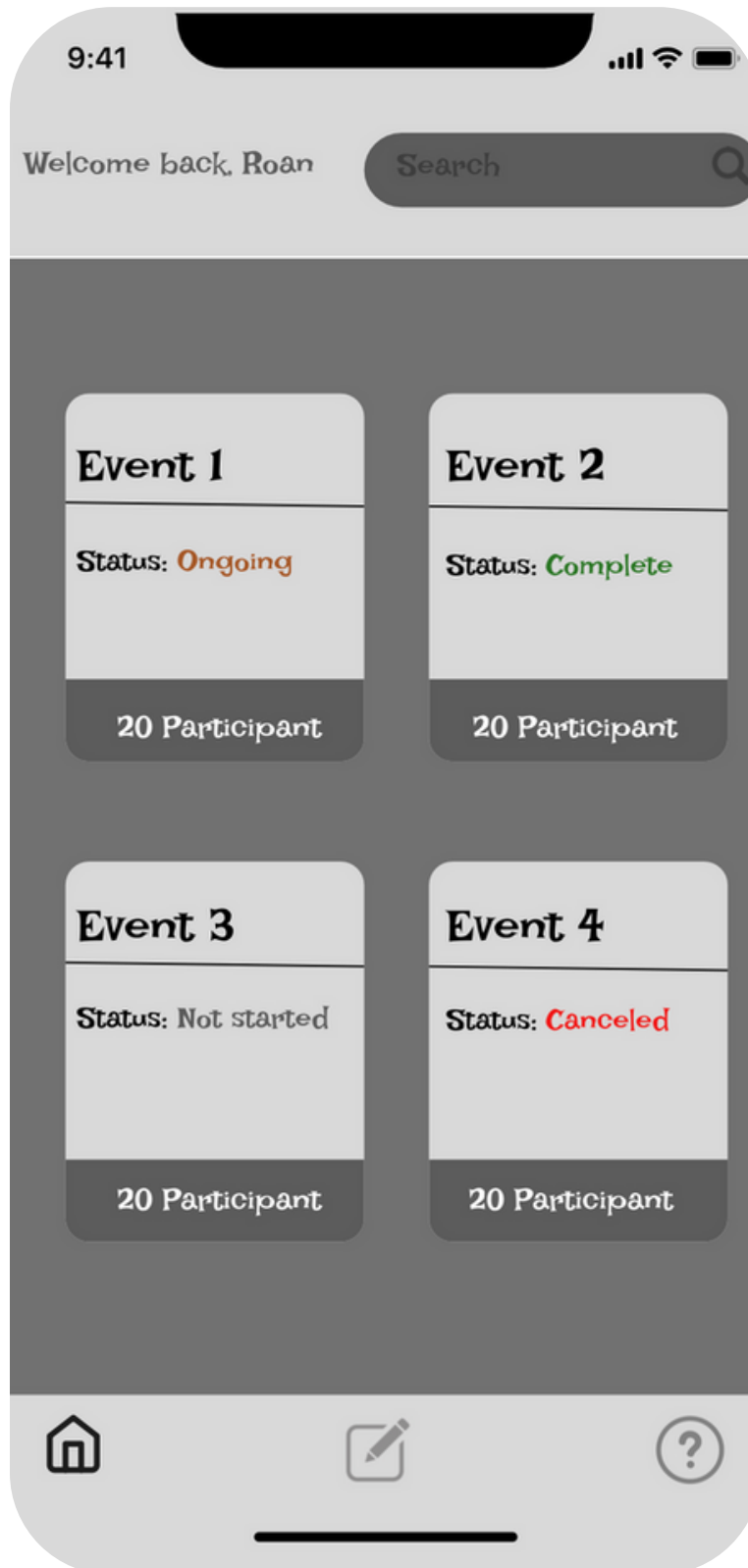
Phone:

Create Password:

Confirm Password:

Sign up

- Home (judge and admin):



- Event Informations (admin):

9:41

Event 1

From: 03/19/2025
- 03/24/2025

ID: Ev__0001

Judging Criteria

Search

| Project name | Grade | Published |
|------------------|-------------|-------------------------|
| <u>Project 1</u> | <div></div> | <div></div> <div></div> |
| <u>Project 2</u> | <div></div> | <div></div> <div></div> |
| <u>Project 3</u> | <div></div> | <div></div> <div></div> |
| <u>Project 4</u> | <div></div> | <div></div> <div></div> |
| <u>Project 5</u> | <div></div> | <div></div> <div></div> |
| <u>Project 6</u> | <div></div> | <div></div> <div></div> |
| <u>Project 7</u> | <div></div> | <div></div> <div></div> |
| <u>Project 8</u> | <div></div> | <div></div> <div></div> |
| <u>Project 9</u> | <div></div> | <div></div> <div></div> |

Add

- Project information(Admin):



The image shows a mobile application interface for managing project information. At the top, the status bar displays the time 9:41, signal strength, Wi-Fi, and battery icons. Below this, a header section contains the text "Event 1" and "ID: Ev__0001" on the left, and "From: 03/19/2025 - 03/24/2025" on the right. The main content area is a form with several input fields: "Project name:", "Category:", "Participant name:", "Telephone:", "Description:" (a larger text area), "Link:", and "Grade:" (a slider control). A green "Create" button is located at the bottom right of the form. The bottom of the screen features a navigation bar with three icons: a house (home), a person (profile), and a question mark (help).

9:41

Event 1

ID: Ev__0001

From: 03/19/2025
- 03/24/2025

Project name:

Category:

Participant name:

Telephone:

Description:

Link:

Grade:


Create

- Create event(Admin):


9:41

Create Event

Name:

Date: 




Location:

Number of projects: 

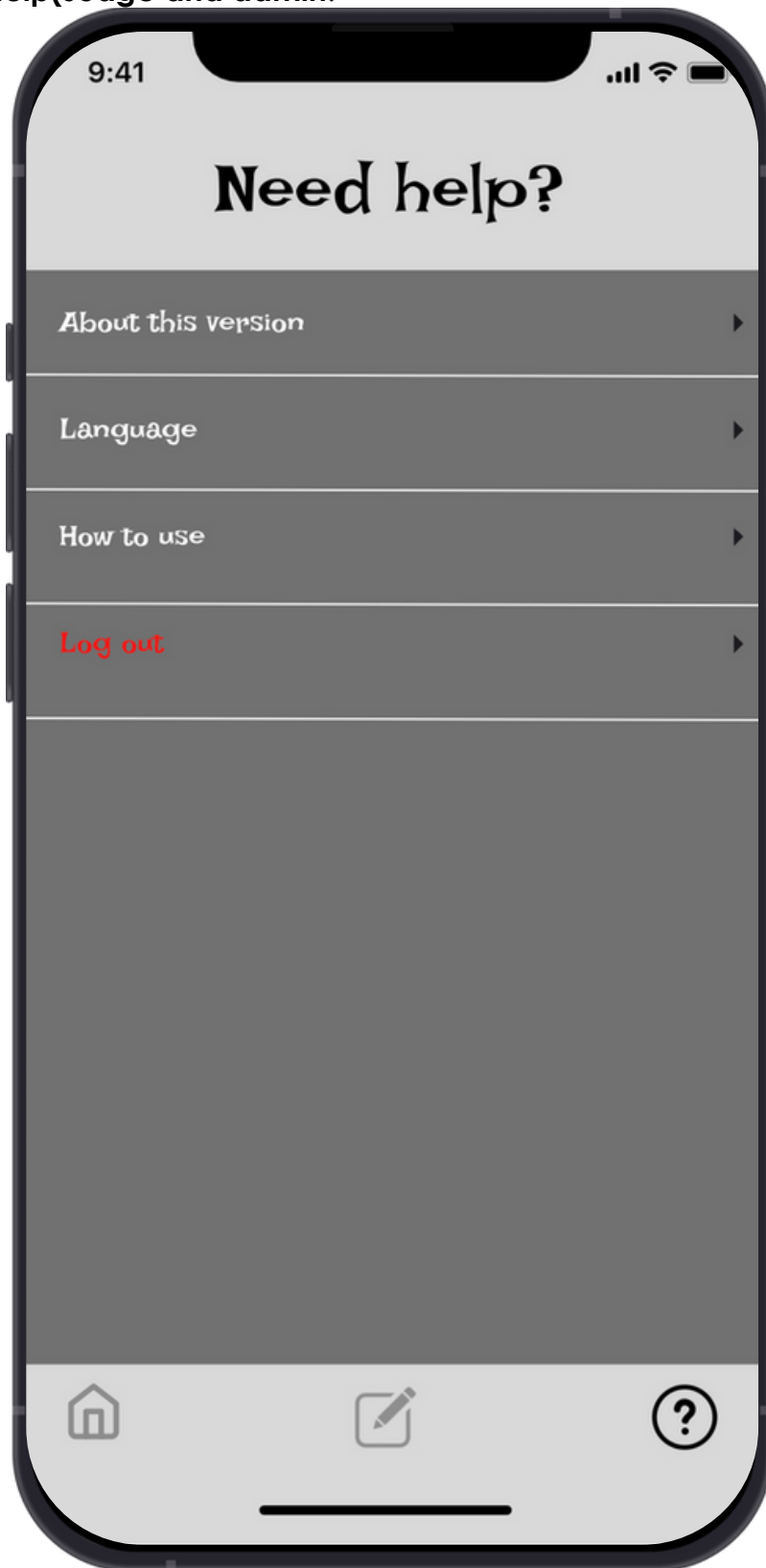
Judge(s):

Criteria(s):

Create

- Help(Judge and admin:



- Language / region(Judge and admin):

The image shows a mobile application interface on a smartphone. At the top, the status bar displays the time 9:41, signal strength, Wi-Fi, and battery icons. The app's title bar is light gray with the word "Language" in a large, black, serif font. Below the title bar is a dark gray background. In the center, there is a white rounded rectangle containing the text "Select the country/region and language." Below this text are two dropdown menus. The first dropdown is labeled "Country/Region" and the second is labeled "Language". Both dropdowns have a small downward arrow on the right side. Below the dropdowns is a green rounded button with the word "Submit" in white. At the bottom of the screen is a white bar with three icons: a house icon, a square with a pencil icon, and a circle with a question mark icon.

9:41

Language

Select the country/region and language.

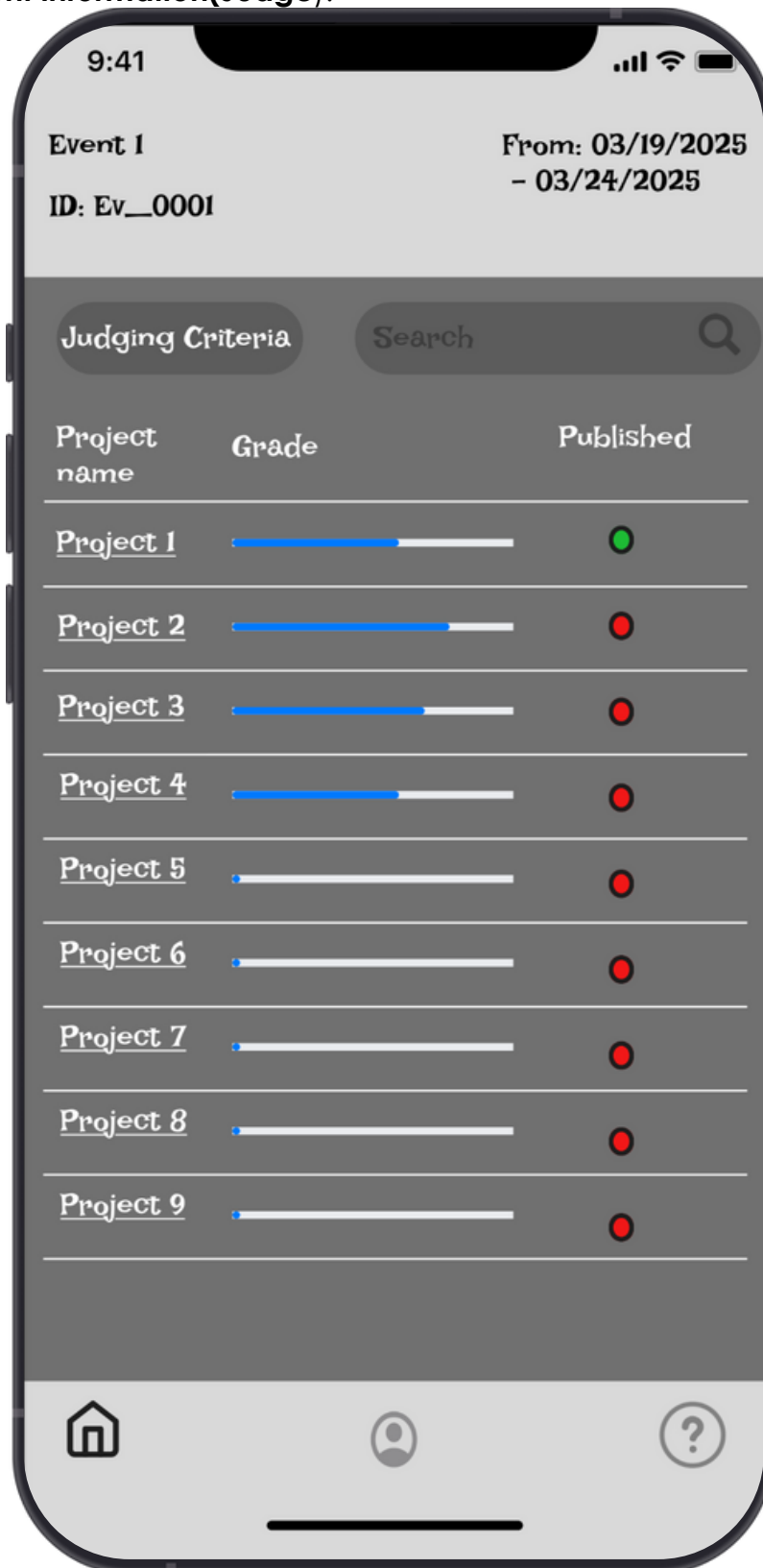
Country/Region ▼

Language ▼

Submit

Home Edit Help

- Event information(Judge):



- Project information(judge):

9:41

Event 1

From: 03/19/2025

ID: Ev__0001

- 03/24/2025

Project name: Owl judge

Category: Programming

Participant name: Roan Simo Masso

Telephone: 4044931078

Description: It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using "Content here, co ...
[Read More →](#)

Link: https://www.youtube.com/watch?v=jKl0a_yCZE

Grade:

Publish Submit

- View personal information(Judge):

The image shows a mobile application interface for viewing personal information. At the top, the status bar displays the time 9:41, signal strength, Wi-Fi, and battery icons. The main content area is a light gray card. At the top of the card is a circular profile picture placeholder with a camera icon and a green plus sign. Below this is a horizontal line. The form contains five labeled input fields: 'First Name:', 'Last Name:', 'Telephone:', 'Email:', and 'Role:'. The 'Role:' field is pre-filled with the text 'Judge'. At the bottom right of the card is a green button with the text 'Update?'. The bottom of the screen features a navigation bar with three icons: a house (Home), a person (Profile), and a question mark (Help/About).

9:41

First Name:

Last Name:

Telephone:

Email:

Role:

[Update?](#)

IV. Development

1. Backend Development:

The backend API is built using **Node.js** for handling HTTP requests. The backend APIs were built using **FastAPI** for asynchronous support and high performance. **JWT** is used for user authentication, ensuring that only authorized users can access sensitive data and submit scores.

2. Database Connection:

In the **OwlJudge Web Application**, the database connection is one of the core components for handling and storing data, such as user information (admins, judges, participants), events, projects, and scores. The system uses **PostgreSQL** as the relational database management system (RDBMS) and **Redis** as an in-memory cache to enhance performance for frequently accessed data.

PostgreSQL Database Connection

To connect to the **PostgreSQL** database, we use **SQLAlchemy** with **FastAPI**. This allows us to manage database sessions, perform queries, and interact with the data stored in the database.

1. Database Overview:

- **PostgreSQL** stores structured data like:
 - **Users:** Admins, Judges, and Participants
 - **Events:** Event details, dates, status
 - **Projects:** Project information, categories, participants
 - **Scores:** Scores and feedback from judges
- **Redis** is used for caching frequently queried data to reduce the load on the database and speed up access.

Setting Up PostgreSQL Connection with FastAPI

Here's an example of how the database connection works in the **OwlJudge** application.

Below are the code snippets for database connection:

Step 1: Install dependencies

```
pip install fastapi
pip install sqlalchemy
pip install psycopg2 # PostgreSQL driver
pip install redis
pip install uvicorn # For running the app
```

Step 2: Set up database connection

In your backend, create a file called `database.py` to manage the connection to the **PostgreSQL** database and the **Redis** cache.

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

DATABASE_URL = "postgresql://username:password@localhost/dbname"

# SQLAlchemy setup
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False,
                             bind=engine)
Base = declarative_base()

# Redis setup for caching
import redis
redis_client = redis.StrictRedis(host='localhost', port=6379, db=0)
```

- **DATABASE_URL:** This string connects to the **PostgreSQL** database. Replace "username", "password", "localhost", and "dbname" with the actual credentials and database name.
- **Redis** is initialized to cache frequently accessed data. Here, Redis is connected on the default port (6379), and we're using **database 0**.

Step 3: Create a database session

For each request that requires interacting with the database, you will need to create a session. This session will interact with the database and handle transactions.

```
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

This function is used by FastAPI to provide a session to handle database queries in the routes.

3. Frontend Development:

The frontend was developed using **React Native** for mobile-first, responsive design. Judges can access their dashboard to evaluate projects, while admins can track event progress and scores in real time.

4. Security:

- **JWT Tokens** is used for authentication to ensure secure login.
- **TLS 1.3** encryption is implemented for all data transmission between the app and the backend.
- **Role-Based Access Control:** Admins, judges, and participants have specific permissions, ensuring that sensitive information is protected.

5. Setting Up the Project

Step 1: Clone the Repository

First, clone the **OwlJudge** repository from GitHub:

```
git clone https://github.com/ccseowljudge.git
cd ccseowljudge
```

Step 2: Install Dependencies

Use pip to install the required dependencies as mentioned earlier:

```
pip install -r requirements.txt
```

This will install FastAPI, SQLAlchemy, Redis, and other required libraries.

Step 3: Set Up the Database

a) Create PostgreSQL Database:

- If you're using PostgreSQL locally, create a database:

```
psql -U postgres
CREATE DATABASE owljudge;
```

b) Configure Environment Variables:

- Create a `.env` file in the root directory of your project to securely store database credentials:
DB_HOST=localhost

```
DB_USER=username
DB_PASSWORD=password
DB_NAME=owljudge
```

c) Run the Database Migrations:

- Use Alembic or SQLAlchemy's `Base.metadata.create_all()` method to create the tables:

```
python run.py
```

Step 4: Configure Frontend

1. React Native Setup:

- Make sure you have **Node.js** and **npm** installed.
- To install React Native dependencies:

```
npm install
```

2. Mobile App Build:

- To build the mobile app for both iOS and Android, use the following commands (ensure you have **Expo CLI** installed):

```
eas build:configure
eas build --platform all
```

Step 5: Start the Application

1. Run the Backend:

- To start the backend server using Uvicorn (for FastAPI), run:

```
bash
```

Copy

```
uvicorn main:app --reload
```

2. Run the Frontend (React Native):

- To start the mobile app in development mode:

```
npm start
```

V. Test Plan and Test Report

Test Methodology:

- **Manual Testing:** Performed to simulate real-world usage scenarios, including user login, event creation, and score submission.
- **Automated Testing:** Focused on security (e.g., login attempts, data encryption), performance (e.g., load testing), and system stability.
- **User Acceptance Testing (UAT):** Real users (admins and judges) tested key features such as login, score submission, and event creation.

Test Results:

- **User Login:** Pass – Users can log in successfully with valid credentials.
- **Score Submission:** Pass – Judges can submit scores without issues.
- **Event Creation:** Pass – Admins can create events with correct data.
- **Real-Time Score Aggregation:** Pass – Scores update in real-time as judges submit evaluations.
- **System Performance under Load:** Pass – The system can handle 100 concurrent users without performance degradation.

| Requirement | Expected Outcome | Pass/Fail | Severity |
|--|--|-----------|----------|
| User Login | Users can log in successfully with valid credentials | Pass | Minor |
| Score Submission by Judge | Judges can submit scores without issues | Pass | Major |
| Event Creation by Admin | Admin can create events with correct data | Pass | Critical |
| Event Deletion and Modification | Admin can delete or modify events correctly | Pass | Critical |
| Email Notifications to Judges | Judges receive email notifications after assignment | Pass | Minor |
| Project and Judge Assignment | Judges are assigned to projects correctly | Pass | Critical |
| Judge Notifications for Project Assignment | Judges receive notifications about project assignments | Pass | Minor |
| Score Auto-Saving | Scores are auto-saved periodically to prevent data loss | Pass | Major |
| Score Locking After Submission | Scores are locked to prevent unauthorized changes after submission | Pass | Critical |
| Real-Time Score Aggregation | Scores update in real-time as judges submit evaluations | Pass | Critical |
| Tie-Breaking Mechanism | Tie-breaker mechanism works when scores are tied | Pass | Major |
| Export Results to CSV/PDF | Results can be exported in CSV or PDF format | Pass | Major |
| System Performance under Load (100 users) | System can handle 100 concurrent users without performance degradation | Pass | Critical |

| | | | |
|-------------------------------------|---|------|-------|
| Real-Time Event Analytics for Admin | Admin can access real-time event analytics during the event | Pass | Minor |
| Password Recovery | Password recovery process works as expected | Pass | Minor |

VI. Version Control

Version Control System:

- The project uses **GitHub** for version control. All team members commit their changes regularly, and the repository is maintained under an organization account for better collaboration.
- **Branching Strategy:** Each feature or fix is developed in its own branch, which is merged into the main branch after thorough testing.

Access Control:

- The repository is private during development, with access restricted to the project team members. Once the project is complete, the repository will be made public for final release.

VII. Conclusion

The **OwlJudge Mobile app** successfully digitizes the judging process for event such as the **C-Day** event at **Kennesaw State University**, improving the accuracy, efficiency, and transparency of event management. The system meets all functional and non-functional requirements, including real-time score aggregation, secure authentication, and scalability for large events. The project is now complete and fully operational, with all tests passing as expected.

Moving forward, future enhancements include the integration of AI-driven analytics to assist with scoring adjustments and expanded scalability to support multi-institutional events.

VIII. Appendix

1. Project Plan:

a) Project Overview / Abstract

The CCSE OwlJudge Web Application is a scalable, secure, and user-friendly platform designed to enhance the digital judging process for Kennesaw State University's C-Day event. The application will enable judges to evaluate capstone projects efficiently, ensuring fairness and transparency. It will feature real-time scoring, project tracking, and seamless integration with university systems.

b) Project Website

URL: Temporary Placeholder:

www.ccseowljudge.com (This will no longer be used because we are developing a mobile app)

c) Deliverables

The project will produce the following key deliverables to demonstrate its success and functionality:

1. **Final Report:** A comprehensive summary of the project, including design choices, results, and challenges encountered.
2. **Source Code:** Fully documented and structured code available on a GitHub organization account.
3. **Website with GitHub Integration:** A central platform that connects all project components.
4. **Final Presentation Video:** A video walkthrough showcasing the application's features.
5. **Weekly Activity Reports:** Regular updates to track progress and maintain accountability.
6. **Technical Documentation:** Detailed guides for setup, usage, and further development.
7. **Prototype Presentations:** Iterative demos to gather feedback during development.
8. **Fully Deployed Application:** The final version of the application, hosted on AWS.

d) Group Meeting Schedule

Meetings will be held regularly to ensure coordination among team members:

- **FaceTime:** Fridays at 6 PM for informal discussions and check-ins.
- **Teams:** Mondays and Wednesdays at 11 AM for in-depth technical planning.
- **In-person:** As needed, for critical technical discussions and debugging sessions.

e) Collaboration and Communication Plan

- Teams for virtual meetings

- Google Drive for shared documents
- GitHub for version control and code sharing
- Trello for task management

f) Project Schedule and Task Planning

The project will follow the Software Development Life Cycle (SDLC) stages:

Initiation Phase:

- Kickoff meeting: Jan 22, 2025
- Requirements gathering: Jan 29, 2025

Planning Phase:

- Database schema design: Feb 17, 2025
- System architecture planning: Mar 03, 2025

Execution Phase:

- Backend development: Mar 10, 2025
- Frontend development: Mar 17, 2025

Closure Phase:

- Testing and QA: Apr 07, 2025
- Deployment: Apr 13, 2025
- Final Adjustments: Apr 17, 2025
- Final presentation C-Day: Apr 22, 2025

Gantt Chart:

| Project Name: OWL JUDGE | | Report Date: 1/23/25 | | | | | | | | | | | | | | | | | |
|-------------------------|--|----------------------|---------------------|----------------------------|--------------|----|----|----|--------------|----|----|----|--------------|----|----|----|-------|----|----|
| Phase | Tasks | Complete% | Current Status Memo | Assigned To | Milestone #1 | | | | Milestone #2 | | | | Milestone #3 | | | | C-Day | | |
| Requirements | Meet with stakeholder(s) SH | 50% | Delayed | Lumiere, James | 10 | 4 | | | | | | | | | | | | | |
| | Define requirements | | | | | | | | | | | | | | | | | | |
| | Review requirements with SH | 10% | | Lumiere | | 10 | 15 | | | | | | | | | | | | |
| | Get sign off on requirements | 0% | | James | | | 5 | 10 | | | | | | | | | | | |
| Project design | Define tech required * | 0% | | Bill | | | | 10 | 4 | | | | | | | | | | |
| | Database design | 0% | | James, Bill | | | | 5 | 10 | 10 | | | | | | | | | |
| | Front End design | 0% | | Lumiere, Minh | | | | | | 10 | 10 | | | | | | | | |
| | Develop working prototype | 0% | | Bill | | | | | | 10 | 10 | | | | | | | | |
| | Test prototype | 0% | | Lumiere | | | | | | | 5 | 10 | 5 | | | | | | |
| | | | | | | | | | | | | | | | | | | | |
| Development | Review prototype design | 0% | | Bill | | | | | | | | 8 | 5 | 10 | | | | | |
| | Rework requirements | 0% | | James | | | | | | | | 8 | 10 | 20 | 20 | | | | |
| | Document updated design | 0% | | Lumiere | | | | | | | | | | | 10 | 10 | | | |
| | Test product | 0% | | Minh | | | | | | | | | | 8 | 5 | 20 | | | |
| Final report | Presentation preparation | 0% | | Lumiere, Minh, Bill, James | | | | | | | | | | | 15 | 10 | 10 | | |
| | Poster preparation | 0% | | Lumiere, James | | | | | | | | | | | | | 10 | | |
| | Final report submission to D2L and project owner | 0% | | | | | | | | | | | | | | | 5 | | |
| Total work hours | | | | | 337 | 10 | 14 | 20 | 25 | 14 | 30 | 25 | 26 | 20 | 38 | 35 | 45 | 10 | 25 |

* formally define how you will develop this project including source code management

| | |
|------------------------|--|
| Legend | |
| Planned | |
| Delayed | |
| Number Work: man hours | |

g) Other Plans

Risk Assessment:

- Identify risks related to AWS resource constraints, performance bottlenecks, and data security.
- Mitigation Plan: Optimize resource usage, conduct regular performance testing, and encrypt sensitive data.

h) Version Control Plan

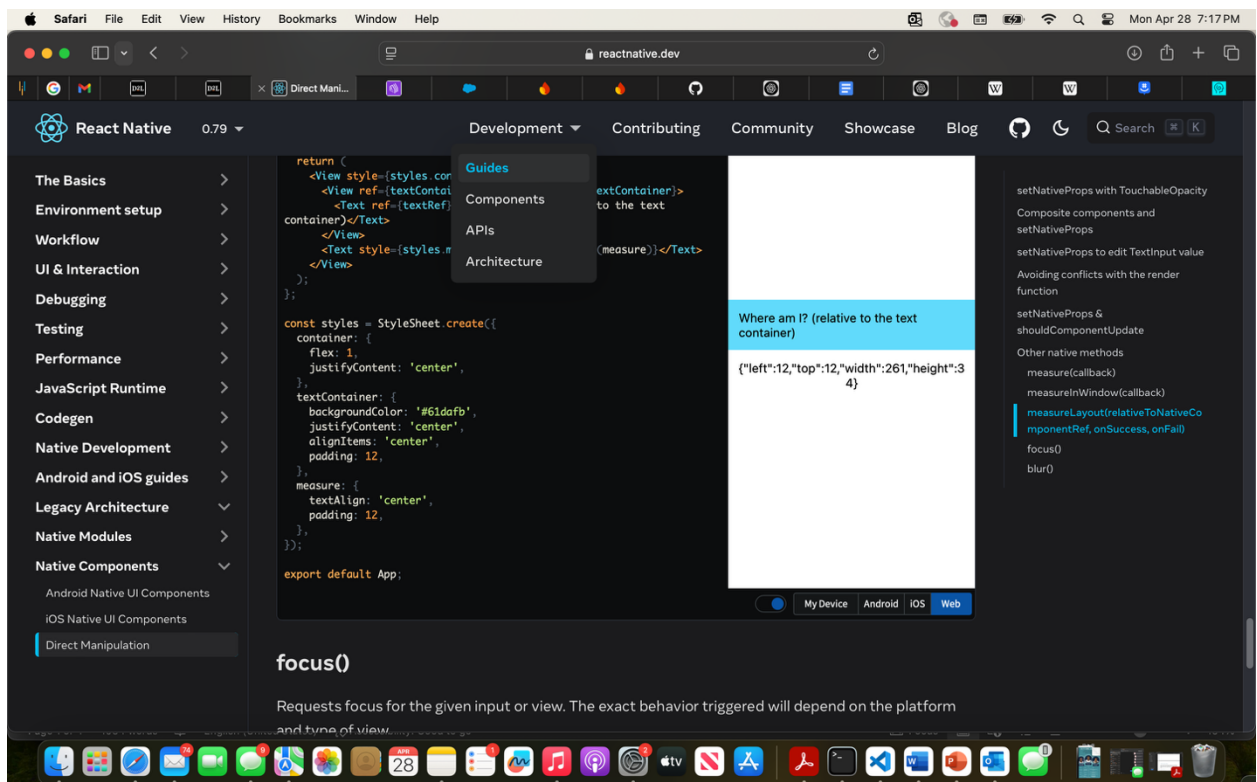
Version Control: GitHub organization account

- **Purpose:** To manage version history and collaboration efficiently.
- **Access Control:** Team members only; public repository for final release.

2. Training Verification (if applicable):

We all completed the react native tutorial and below are screen shots of the final module completed by each membe.

- Roan Simo Masso



- Nathanael Fokou

Course Home Content Discussions Assignments Quizzes Other ▾ Classlist Grades

Table of Contents > Tutorials > Mobile Apps > Flutter & React Native

Flutter & React Native ▾

React Native 0.79 ▾

Development ▾
Contributing
Community
Showcase
Blog

The Basics >
Environment setup >
Workflow >
UI & Interaction >
Debugging >
Testing >
Performance >
JavaScript Runtime >
Codegen >
Native Development >
Android and iOS guides >
Legacy Architecture ▾
Native Modules >
Native Components ▾
Android Native UI Components
iOS Native UI Components
Direct Manipulation

measureLayout example

```

return (
  <View style={styles.container}>
    <View ref={textContainerRef}
      style={styles.textContainer}>
      <Text ref={textRef}>Where am I?
    </Text>
    <View
      style={styles.measure}>[JSON.stringify(measure)]</Text>
    </View>
  </View>
);

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
  },
  textContainer: {
    backgroundColor: '#61dafb',
    justifyContent: 'center',
    alignItems: 'center',
    padding: 12,
  },
  measure: {
    textAlign: 'center',
    padding: 12,
  },
});

```

Where am I? (relative to the text container)

{'left':12,'top':12,'width':261,'height':34}

setNativeProps with TouchableOpacity
Composite components and setNativeProps
setNativeProps to edit TextInput value
Avoiding conflicts with the render function
setNativeProps & shouldComponentUpdate
Other native methods
measure(callback)
measureInWindow(callback)
measureLayout(relativeToNativeComponentRef, onSuccess, onFail)
focus()
blur()

- Lumiere Nathan Kue

React Native 0.79 ▾

Development ▾
Contributing
Community
Showcase
Blog

The Basics ▾
Introduction
Core Components and Native Components
React Fundamentals
Handling Text Input
Using a ScrollView
Using List Views
Troubleshooting
Platform-Specific Code
More Resources
Environment setup ▾
Get Started with React Native
Set Up Your Environment
Integration with Existing Apps
Integration with an Android Fragment
Building For TV Devices
Out-of-Tree Platforms
Workflow >
UI & Interaction >
Debugging >
Testing ▾
Testing
Performance >
JavaScript Runtime >
Codegen >
Native Development >

```

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
  },
  textContainer: {
    backgroundColor: '#61dafb',
    justifyContent: 'center',
    alignItems: 'center',
    padding: 12,
  },
  measure: {

```

My Device Android iOS web

focus()

Requests focus for the given input or view. The exact behavior triggered will depend on the platform and type of view.

blur()

Removes focus from an input or view. This is the opposite of `focus()`.

Is this page useful? 🍏 🍌

Edit page for next release
Edit page for current release

Previous
« iOS Native UI Components

setNativeProps with TouchableOpacity
Composite components and setNativeProps
setNativeProps to edit TextInput value
Avoiding conflicts with the render function
setNativeProps & shouldComponentUpdate
Other native methods
measure(callback)
measureInWindow(callback)
measureLayout(relativeToNativeComponentRef, onSuccess, onFail)
focus()
blur()

Last updated on Apr 14, 2025