Software Engineering Practices Used to Build a Juvenile Justice Education Platform - Final Report

Khaled Kyle Wong[1], Nguyen Nguyen[1], Tim Nguyen[1],
Nyan Jonathan Tun[1], Jason Wong[1], Alexander Chen[1]

[1] *UCLA Computer Science Department, CS130*

Abstract

The group was tasked with building an application that will teach youths about the Los Angeles Juvenile Justice System. The application aims to educate convicted youths about the justice process, to mitigate the mental stress of the conviction. The user-facing application runs on iOS, and presents educational content to the youths. The application pulls phrases and words from a backend database. Administrators can modify the words through a web-based dashboard that modifies the database. Our team has been responsible for the backend database and admin dashboard; a separate but associated team has developed the iOS application. We introduce the problem area, discuss the timeline of our development/delivery, and describe our design. We have successfully delivered the completed product to the client, and have given a live demonstration of the product in class.

1. Problem Definition

1.1. The Necessity of an Educational Platform

When youths are convicted, they are put through a juvenile justice system that closely follows the adult prosecution process; the process starts with an arrest, followed by a detention period, which leads to an official trial in court. The convicted youths are generally undereducated about this process, which only adds to the stress and fear of the conviction process [2]. Attempts have been made to mitigate this problem by providing convicted youths with informational pamphlets to review during their detention periods, and before their trials; however, the pamphlets are either too wordy, too complicated to comprehend, or get neglected out of stress [3]. It is vital to provide the youths with a more effective education platform, to enable them to successfully navigate the conviction process. The platform should solve the problems of the pamphlets by being less wordy and more accessible and appealing to the youths. The natural format for this new platform would be an interactive application that the youths can learn from during their detention period. This idea has manifested as the "Juvenile Justice Education platform" with which our team has been tasked to build.

It is equally important to educate the youth's parents as well, because parental education about legal proceedings is vital to prevent the youth from re-offending. Research studies confirm this, showing that when parents learned about the judicial process and participated more in their children's legal proceedings, the youth was less likely to re-offend within a year [1]. The same study shows that parents of convicted juveniles are misinformed about critical details regarding the court proceedings. For example, only 20.69% of parents were able to correctly identify which party gets to decide how a convicted youth will plead, if a parent hires a lawyer for the youth [1].

On the whole, the Juvenile Justice Education platform will certainly help to increase education levels of both the youths and their family members. The application will not only reduce the stress and fear of the youth during the trial process, but can also help to prevent youths from re-offending in the future.

1.2. Overview of Educational Platform's Features

The educational platform consists of a user-facing iOS app, and an admin-facing dashboard, both of which are connected through a backend database. We detail these features below.

The role of the backend database is to store textual content that will be displayed in the iOS app. The textual content includes a mapping from judicial vocabulary words to definitions. The textual content also includes a mapping from the names of key players in the judicial system, to a description of the roles of each key player.

The user-facing iOS app has three main sections. The first section aims to inform the youth about the key players of the judicial system. This section will show a picture of the standardized court layout, with the key players sitting in their appropriate positions. This section will be interactive: the youth can select each of the key players to learn more about their roles. The next section is a glossary, where the youth can type in a judicial phrase to quickly find the definition of the phrase. The final section shows a list of links to resources that the youths can click on to learn more. The app must be released to the iOS app store, to allow the judicial officials to review and approve the application for use by the youths, and also to provide the youths and their families with an easy way to download the app. As

previously explained, it is important for the youths to be able to use this application anonymously, so the app will not require a sign in.

The admin-facing dashboard will allow the admins of this project to modify the content that appears in the iOS app. Using the dashboard, an admin will be able to modify the descriptions of key players of the judicial system. The admin will also be able to perform Create/Read/Update/Delete operations on the vocabulary content that shows up in the iOS app's glossary section. Any access to the admin features will be protected via a login system; Alicia and Jyoti will be the original admins of the system. Any admin will be able to invite additional admins through the dashboard, which will trigger an email workflow to allow the invitee to complete the invitation process.

1.3. Division of workload across discussion sections

There are two teams assigned to the Juvenile Justice Educational Platform project. Our team is from Discussion 1A, an the other team is from discussion 1B. As suggested by the clients, and approved by Dr. Eggert, our teams will be collaborating to build this project. Our team will develop the admin-facing dashboard, and the datastore. The other team will develop the user-facing iOS application. Our team will expose an API to the other team, who will pull textual content from the API.

1.4. Stakeholders

There are three main groups of stakeholders for this project: the youths (and family), the admins of the project, and the members of the LA Justice System. We describe each group below.

The first stakeholder group is the youths and their family. The youths may be as young as 13 years old, with an estimated reading level of 7th grade comprehension. The youths will use the iOS application to learn about the conviction process. The youths must be able to use the application anonymously, to prevent the youth's lack of knowledge from being used against them during the court proceedings. As previously mentioned, it is equally important to educate the youth's family as well.

The second stakeholder group is the admins of this project, referred to as the clients throughout this paper. The clients consist of Alicia Verani, and Jyoti Nanda, both of whom are associated with the UCLA School of Law. The clients have proposed this project in an effort to help the youths; thus, monetization and profit is not a concern in the scope of this project. The admins have expert understanding of the justice system, and will be able to provide all the educational content for the application, through the admin dashboard.

The last stakeholder group consists of the members of the LA Justice System. While these members will not be directly using the educational system, they are vital to the project objective, since they will need to approve of the educational platform and give the youths permission to use the platform prior to the trial.

2. Features Implemented

First, Firebase [6] authentication method via a combination of email and password is used to log the administrators in. As requested by the client, only signed-in administrators will have access to modify the application content through the admin dashboard. This feature is fully functional, and additional error reporting capability is also provided to administrators. For instance, if the admins entered ill-formatted emails, this will be caught in an exception handler that follows immediately after calling Firebase signInWithEmailAndPassword() function. Furthermore, if the administrators enter well-formatted emails but wrong password, this error will also be caught and reported. With regards to the user-facing iOS application, users (primarily the convicted youths), will be able to access the content without having to sign up for an account.

Second, the feature of protecting "private" components from user who has insufficient privilege is also finished. For instance, if a user is not logged in, they will not be able to navigate to the admin panel page by manually entering the URL into the web browser. Furthermore, when an unauthorized user tries to go to the admin panel page, they will be automatically redirect back to the landing page where they have to log in. The authorization checker component is implemented using the Higher Order Component (HOC) pattern in Reactjs [4], which allows it to be reused by other components that require authorization. In addition, if an administrator is logged in and enters the website default path, they will be automatically redirect to the admin panel page. This feature is desired as it will prevent an administrator from mistakenly login twice.

Third, administrator authentication state persistence is also implemented fully with Firebase. As of now, the setting is set to LOCAL which means that the admin authentication state is saved when the tab closed or even when the entire browser is closed. This setting can be switched to SESSION where the authentication state is saved only between tabs but not browser or it can be set to NONE which does not save any authentication state.

Fourth, there is an issue with using Firebase onAuthStateChanged() method or the React Context to get the administrator authentication state when navigating between components since the components need the authentication state to determine whether the user has sufficient privileges to view the page. The problem stems from the fact that Firebase onAuthStateChanged() is an asynchronous function, which means it does not provide the accurate authentication state right away. However, in the meantime, React components will call the render function

instead of waiting for the accurate authentication state to come back. This often causes a flash of the screen where the component renders it as if the administrator is not logged in then once the authentication state comes back from the onAuthStateChange() asynchronous function, the component will rerender based on this value. The fix for this issue is also accomplished. The solution is to use an indicated in the React component state to tell the component whether to display anything in rendering the component. Thus, if the onAuthStateChange() asynchronous call has not returned, the indicator would be set to false, and the component would not display anything. Once onAuthStateChange() returns, the indicator is set to true for display. This feature was crucial, as we do not want unauthorized user to be able to see a flash screen of what an admin panel page would contain.

Fifth, implementing the sign in page as a modal that floats over the main landing page is also finished. In the beginning, the sign in page would have taken the administrator to a separate page with a different route, but this is found to be unnecessary, so this component is refactored to be displayed in a modal over the main landing page instead.

Sixth, when an admin is logged in, they will also be able to see the log out button which allows them to log out. This feature is completed as well.

Seventh, a Firebase wrapper for the iOS application is written. Because Firebase is a real-time database, there are nuances with performing Create/Read/Update/Delete (CRUD) operations to the databases such as working with promises or supplying callback functions, so the purpose of the Firebase wrapper is to make the function calls to perform CRUD operations cleaner.

Eighth, another feature that is being implemented is giving administrators the ability to reset their password if they forget it. This feature would allow administrators to specify their email, and an email will be sent from Firebase to the admin, which will let them reset their password.

Ninth, the admin panel page which is where all the information regarding what content is displayed to the regular users, convicted youth and their family, is completely finished. Administrators are able to view the information that is currently being displayed to the end users, and they are also able to make changes to the definitions in the courtroom page, add/remove terms with their definitions in the glossary page.

Tenth, because there is no sign up functionality as it is required by the client that no arbitrary individual should be able to sign up for an administrative account, there has to be a way for current admins to add more admins to the application. After exploring the trade offs between different ways to build an invitation system mainly considering user ease of use as well containment of services, we want a system where it is easy for current administrator to as little work as possible to add a new administrator. This leads to the solution that when a current admin adds a new admin via email, an email will be sent to the new admin, but this will require a new email sender service. However, we want to keep third party services to the minimal to save our clients' money. Thus, we end up going down the route with Firebase where when a new admin's email is added, an account with this email and a random password will be created. After this step, the internal system will automatically call the forget password function which will send an email to the new administrator and allow them to reset their password.

3. Approach/Design

3.1 Architecture

The overall application can be split up into two encapsulated projects. One team will take care of the iOS user-facing application and the other will handle the back-end data store and API handlers for queries from the user application. The architecture for the latter team will be the sole focus of the report. For the back-end, Firebase was the top contender of choice for four reasons. Firstly, the team has prior experience with the platform to be able to utilize it with little additional learning curve and overhead. Second, Firebase also provides a free quota, which will allow deployment of the service with minimal or even no operational cost. Thirdly, it works well because the amount of data to be stored is not projected to grow substantially. Lastly, it has a built-in testing framework, firebase-functions-test. For deployment, we will be using Github Pages as it is simple and easy to use. The number of admins for the repository is also not projected to change drastically throughout this cycle of development. The front-end for the admin dashboard will be built using React due to its user-friendliness, long-term viability, and the team’s experience with it. For the front-end testing framework, Jest [9] will be used.

3.2 Component Interaction

For the interactions across various components, the Firebase data store will be central to most parts of the application. The React admin app will be interacting with the Firebase data store to modify, add or delete information using CRUD operations in real-time. This component will be designed for Alicia and Jyoti and other future assigned admins for this juvenile justice system project. The iOS user application will then communicate with the data store to query for courtroom and glossary definitions used through a Firebase wrapper in Swift. The wrapper will expose a simple API to the user application for the other team to interact with. Within the React admin app, a JavaScript Firebase wrapper will be used to communicate with a Firebase authentication component.

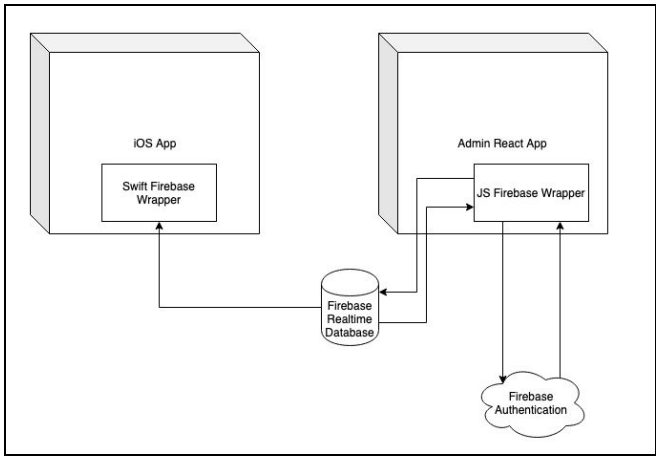

Figure 1: Flowchart of interaction across components

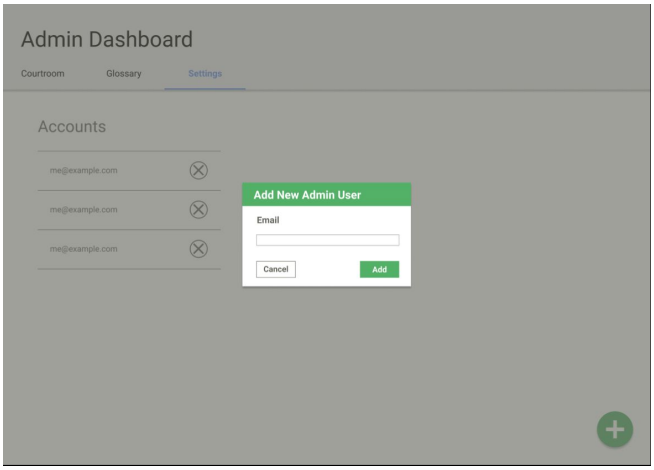

Figure 2: Adding a new admin user

3.3 Design

The React admin app was mocked-up using Figma [5]. The Figma prototype was showcased to the clients, which enabled our development team to reach mutual agreement about the user interface of the dashboard, before any code was written. We ended up going through two design revisions (the original proposed design, and one follow up design). Preparing a prototypical design thus saved us from having to rewrite code and redesign the user interface through HTML/CSS in the code. Once the client signed off on the Figma design, the development team was able to start implementing the design in code. We had very few disagreements amongst the development team, with respect to the desired user-interactions, and with respect to the dashboard's appearance. After the entire application was implemented, we found that the real implementation was very similar to the finalized mock up in Figma. These experiences showcase the tremendous benefits of creating a mock-up design prior to development; because of this, it is clear that the Figma design was a justified time investment. Some of the various use cases that the design handles include:

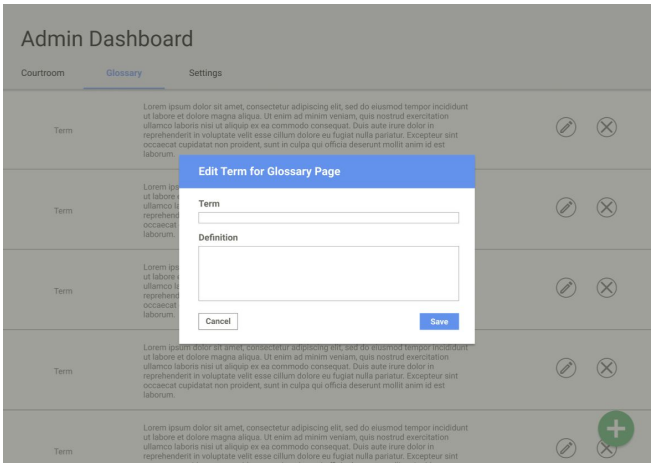

Figure 3: Editing a term in the Glossary tab

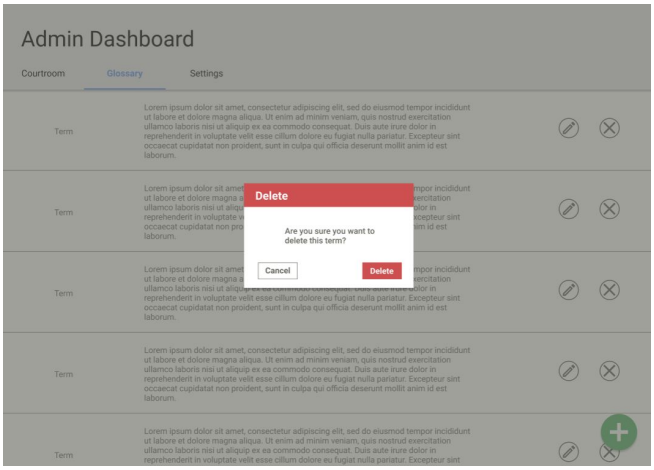

Figure 4: Deleting a term

In terms of structure, first a framework (Material UI) and color palette were chosen; we considered accessibility when selecting this color palette, to be user-friendly towards colorblind users. Then reusable components were mocked and then the individual pages were created. Finally, interaction support was added through Figma as well.

3.4 UML Use Case Diagram

The following diagrams are UML Use Case Diagrams, which were created with LucidChart [8]. They are meant to demonstrate how different users interact with the system that is built. Since there are two major components of the projects, one UML Use Case Diagram is made for each corresponding component. At the brief overview, one UML Use Case Diagram is made from the administrator perspective while the other one is made for the end users like the convicted youth and their family. Since we are collaborating with another team, we are responsible for developing the administrative dashboard. The other team are building an iOS application that allows convicted youths to browse the information that is chosen to be displayed by the administrators.

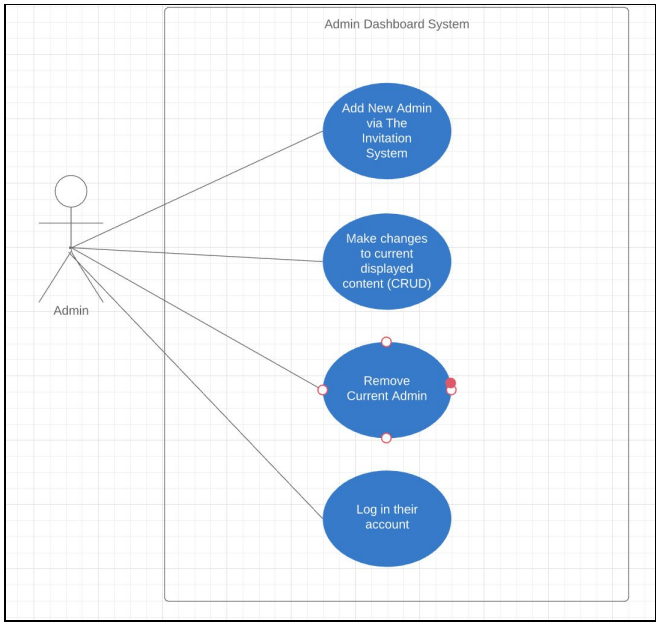

Figure 5: Admin Dashboard System

Figure 5 is the UML Use Case Diagram [7] that describes the functionalities for the admins that we are developing. As shown in Figure 5, the administrators will be able to add new admins by the invitation system that we build since we do not want anyone to be able to sign up for an admin account. The admins will then be able to log in to their account to perform administrative tasks. Second, as the core functionality of this dashboard, the admin will be able to view and make any changes to the current information that is being displayed to the convicted youth through creating, reading, updating, or deleting the information.

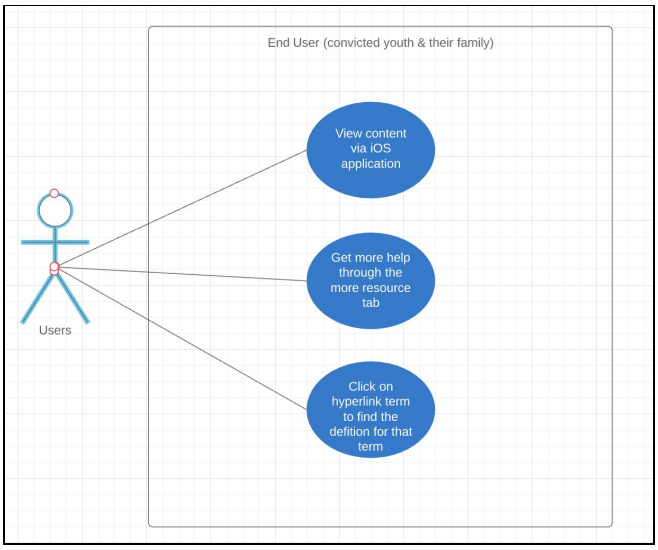

Figure 6: Emd User (convicted youth & family) System

Figure 6 is a UML Use Case Diagram [7] that depicts the functionalities that the youths will have. They will have a tab where they can view the of the information regarding the trial process along with the terminology. There will also be another tab that lets the youth find more related resources and articles. Lastly, there will be a feature where the youth can click on any word that has a hyperlink to be taken to the word definition.

4. Software Engineering Practices

4.1 Task Planning and Allocation

For task planning, the team decided to use Trello [10], a task tracking visual collaboration platform. From the requirements, all the team members wrote down the tasks they thought were most crucial on the visual board and then individual members picked the tasks they wanted to work on. The board categorized between unpicked tasks, tasks in progress, and finished tasks to let all members know which were left. Comments were also occasionally used to discuss unclear task progress / scope.

4.2 Version Control

The team defaulted to the commonly used Git for version control. Each member working on a separate feature would push changes to his or her own branch (concurrent

development through optimistic VC) and then request pull request to merge into the development branch. After thorough examination, development branch would be pushed to the master branch, which will be the one used for deployment. The team also used branch protection to enforce that PRs to the development branch got at least one approval before merging and that PRs to master got two.

4.3 Code Styling

To ensure standardized code quality and style across the team, Eslint, Prettier, and Husky were used. Eslint is a linter tool that outputs error messages when its standards of proper code styling are not met. Prettier is a plugin that formats code consistently and Husky is used to set up git hooks for eslint.

4.4 Documentation

The team also has multiple documents to help efficiently onboard future developers of the team or interested parties of the overall project. The SRS is a good start to understanding the mission, client needs and scope of the project and the README.md file within the repository helps with configuration of the project for deployment or further modification.

4.5 Collaborative Development and Peer Review

Both pair programming and tool-assisted peer review tools were used throughout development. The goal was to be able to utilize the advantages of both strategies. Pair programming would help with providing a tighter feedback loop as well as having more than one member understand a component of the project. Github code reviews were helpful in formally assessing code and having a more formal alternative for risk management and analysis.

For example, as Jason was an experienced React developer, and since Nguyen had no prior experience with React, Jason and Nguyen pair-coded near the beginning of this project. Nguyen was given the "driver" role, and Jason was given the "observer" role. This helped ensure that Nguyen learned how to code in React firsthand; this also helped Nguyen learn the commands for spinning up a React development server on his local machine. By the middle of the project, Nguyen had developed sufficient expertise to allow him to work independently. Evidently, pair-programming ensured a rapid transfer of knowledge and a fast learning curve for new developers, which would not have been possible through independent learning or formal inspections.

Github code reviews helped ensure that all team members were synced up on the state of the project, and which tasks were done. Using the Github website, we mandated that all pull requests to our development branch required at least one peer-approval, before merging. Similarly, we mandated two peer-approvals when merging develop into master, as part of our branch protection scheme.

4.6 Testing

The team used Jest and Enzyme to test various components of the project. Jest is a JavaScript testing framework and Enzyme is a JavaScript testing utility for React, and helps with rendering DOM objects for front-end testing. Enzyme allowed snapshot tests which renders a UI component and takes a snapshot to compare with a reference snapshot taken from the past when determined functional. The purpose is to make sure that unexpected UI changes do not occur.

5. Software Design Patterns

5.1. Observer Pattern

The observer pattern is an event-driven design pattern that involves two key entity types: the subject, and the observers. In this pattern, the subject maintains a list of observers who have registered with the subject. The observers register in order to receive updates about state changes from the subject; the subject publishes these state changes by calling functions on all its observers.

We used this design pattern when accessing our Firebase realtime database through the React user interface. The observers were the react components that needed to use data from the Firebase database. The Firebase Realtime Database was the subject. Each component would register with the subject during initialization (using React's componentDidMount lifecycle method). As part of this strategy, the component would provide an update() method that would be called by the subject, to notify observers of state changes. This update() method would refresh the UI to reflect the latest data from the database. The subject, which was the Firebase Realtime Database handler, would maintain a list of subjects; every time a node in the Database was created/updated/deleted, Firebase would loop through its subjects and call their update() methods. In this way, we effectively used the observer pattern to allow React to fetch data from Firebase.

5.2. State Pattern

The state pattern is a design pattern that parallels the concept of a state machine. In this pattern, an object maintains an internal state that drives its behavior; the object's behavior is a function of its internal state. We used ReactJS to implement the user interface of the project, and the convention in ReactJS is to implement components using the state pattern. In specific, all React components are controlled by a React `state` (and also by React `props`); Both of these are key-value mappings that hold the component's internal state. Every time the internal state of a component changes, the component is re-rendered, causing

the behavior of the component to change. This precisely implements the state design pattern. A component's internal state can be changed by the component's parents, the component itself, or the component's descendants.

5.3. Modularization via Components

Modularization is an important design strategy in which code is separated into individual modules; this encourages a separation of concerns, as different developers can work on independent, reusable components; this ultimately adheres to the DRY (don't repeat yourself) principle of software engineering. ReactJS inherently encourages modularization via components, since each "unit" of interface must be defined as a separate React-Component. We ultimately found that this modularization did in fact prevent redundant code; it helped organize the development process, because it was easy to delegate different components across the team, and combine the components in one top-level parent component. Further, modularization made testing simpler, since the independent nature of each React Component allowed us to unit test the basic functionality of each component, before combining them in a larger parent module; this followed the bottom-up development approach of developing and testing smaller units of work, before putting them all together.

5.4 Higher Order Component Design Pattern

Higher Order Component (HOC) Design pattern is a very popular design pattern in React that allows frequent code reuse by wrapping components inside other components, to enable the reuse of component logic. One HOC in our project is the withAuthorization component. A child component must be passed into the withAuthorization component, and in this way, the developer is able to control whether the passed-in component is a public or private page. If the page is a private page, and the user is not logged in, the application logic would redirect the user back to the landing page. However, if the page is a public page and the user is logged in, the application logic would automatically route the user to the admin panel page.

```
  componentDidMount() {
    const { firebase, history } = this.props;
    this.listener = firebase.auth.onAuthStateChanged(authUser => {
      this.setState({ authUser, display: true });
      if (!authUser && !isPublic) {
        history.push(ROUTES.LANDING);
      } else if (authUser && isPublic) {
        history.push(ROUTES.ADMIN_PANEL);
      }
    });
  }

  componentWillUnmount() {
    this.listener();
  }

  render() {
    const { authUser, display } = this.state;
    if (display) {
      if ((authUser && !isPublic) || (!authUser && isPublic)) {
        return <Component {...this.props} />;
      }
      return null;
    }
    return null;
  }
 }

 return withRouter(withFirebase(WithAuthorization));
};

export default withAuthorization;
```

Figure 7: withAuthorization HOC

In order to perform the permission-based routing mentioned above, one would have to obtain the current signin status and compute the appropriate permission check. One can use the withAuthorization HOC code in figure 6 as followed:

```
export default withStyles(styles)(withAuthorization(Admin)(false));
```

Figure 8: Admin Component

```
const LandingPage = withAuthorization(Landing)(true);

function LandingExport() {
  const classes = useStyles();
  return <LandingPage classes={classes} />;
}

export default LandingExport;
```

Figure 9: Landing Component

Both the Landing and Admin Components were wrapped inside the withAuthorization component; because of this, both components exhibited appropriate behavior with respect to authorization-based routing. If these components were not wrapped inside the withAuthorization HOC, the code in withAuthorization would have been copy-pasted into in every single component that needs the authorization checking and routing; this would be a poor alternative,

because it violates the DRY (don't repeat yourself) principle of software engineering. The HOC strategy was thus a good development strategy, because it allowed us to avoid redundant code, resulting in a more concise, more readable, and less error-prone codebase.

6. Project Timeline

The following itemizes the timeline of completed events, up to and including final product delivery.

6.1. Timeline of Events

- 04/24: Project team was assigned
- 05/02: [Client meeting] First meeting with Alicia and Jyoti to discuss requirements
- 05/06: Finished first iteration of SRS
- 05/10: Decision to collaborate with associated team from Discussion 1B team for project
- 05/10: Internal team meeting to discuss design
- 05/13: Internal team meeting to review software design progress
- 05/15: [Client meeting] Second meeting with Alicia and Jyoti: review SRS and mock-ups
- 05/17: Midterm presentation
- 05/22: Finished back-end
- 05/27: [Client meeting] Deliver MVP to Alicia and Jyoti
- 05/30: Finalize stretch goals and freeze code
- 05/31: [Client meeting] Project demo
- 06/03: Finished writing basic unit tests
- 06/05: Final class presentation
- 06/07: Finalize unit tests
- 06/13: Submission of final report

7. Key Takeaways

The team had many takeaways from the completion of the front-end dashboard.

The first was to produce responsive UI (robustness) for various screen sizes. This is particularly important because responsiveness cannot be easily simulated by the design, but is only applicable once written in code. Being able to support any screen size is important from a usability perspective and adds significant convenience to users.

The second was to consider accessibility more (aria, color schemes for color blind users). Although the client would not benefit significantly from accessibility improvements, future users of the admin dashboard may find the application easier to use if it followed accessibility guidelines more closely.

The third is to ensure code was structured properly for loose coupling and appropriate modularity. This would significantly improve code organization and make the codebase more readable, more understandable, and easier to maintain.

The final takeaway is to have extensibility in mind for future development (having TODOs, and method stubs for added functionality in the future). The clients plan to hire developers to work off of the existing codebase, so improving our knowledge transfer in this way would allow new developers to onboard more easily.

8. Future Goals

There were three main goals that were not covered within the scope of the current development cycle that can extend into future developments.

8.1 Internationalization (i18n)

In the middle of development, the clients proposed that they would want Spanish support in the future given the prominence of native Spanish speakers in Southern California. From the back-end, one option is to simply add more terms and definitions translated into a language of the client's choice. Though this would be feasible, a more proper practice as to add unique IDs to the terms and then associate various term additions to the same ID but in different languages. On the user application end, more work may need to be done. A button on the home page would be required and font and sizing would have to be readjusted according to the language.

8.2 Preview Changes

A benefit for administrators would be to be able to see how changes to terms and definitions would propagate to the user-facing application before the changes are actually published. There could be an app running through an emulator or on mobile which is purely for testing purposes.

8.3 Courtroom Page + Glossary Page Term Linking

The final feature that would be beneficial in the future would be to have definitions in the courtroom page link to glossary definitions which would be less simplified and more comprehensive. The purpose would be for youths and guardians alike to be able to dive deeper to understand more than the rudimentary information at hand inside the interactive courtroom page.

9. Reflection

9.1 Design Decisions

Having a serverless architecture with Firebase was very efficient in terms of implementation, since Firebase's frontend API allowed us to perform all CRUD operations from ReactJS directly. Furthermore, we did not have to spin up servers or control the traffic with load balancers, since the Firebase realtime database automatically scales with the

size of the query result; this ultimately prevented scaling issues entirely.

However, there are some drawbacks that we would like to address. First, we want to give the admins the ability to 1) view a list of current administrators, and 2) completely remove admins accounts from the system (i.e. admin deletion). However, these functionalities are not completely provided in our current implementation of the dashboard. In particular, the latter feature (deletion of an admin) was not doable within our time constraints; we provided an alternative feature that allows admins to enable or disable other admins from the settings page of the dashboard. To implement this, we created an authorization node in the Firebase realtime database, where we would save a list of users into a database, along with a boolean value representing if the user was enabled or not. When an admin tries to sign in we first check this authorization node in the database. If the user's email address is not present in the database, we deny access. Else, if the user's email address is present in the database, we check the value of the "enabled" boolean. If the user is disbabled, we forgo the firebase authentication check, and simply deny access. Finally, if the user is enabled, we hand off the authentication workflow to the firebase authentication API.

```
// If user already exists, do not allow the add
if (users.indexOf(emailInput.replace(/\./g, ',')) !== -1) {
  this.handleShowSnackbar("Admin with this email already exists");
  return;
}

firebase
  .doCreateNewUser(emailInput)
  .then(() => firebase.doPasswordReset(emailInput))
  .then(() => {
    this.writeUserToDb(emailInput, true);
  })
  .then(() => {
    this.setState({
```

Figure 10: Account Creation

```
const { email, password, users, usersObject } = this.state;
const { firebase, history } = this.props;

// Key into the data from firebase
const emailKey = email.replace(/\./g, ',');

if (!users) {
  this.setState({
    error: new Error("Unable to fetch firebase user list")
  });
  return;
}

// If user is not in auth DB
if (users.indexOf(emailKey) === -1) {
  this.setState({
    error: new Error(
      "Not authorized. This email does not have an admin account."
    )
  });
  return;
}

// User is in the auth DB, but they have been disabled
if (!usersObject[emailKey].enabled) {
  this.setState({
    error: new Error(
      "Not authorized. The admin account associated with this email has been disabled. Contact your manager to
    )
  });
  return;
}
```

Figure 11: Email Checking

Delete a user

The Firebase Admin SDK allows deleting existing users by their `uid`:

Node.js | Java | Python | Go

```
admin.auth().deleteUser(uid)
  .then(function() {
    console.log('Successfully deleted user');
  })
  .catch(function(error) {
    console.log('Error deleting user:', error);
  });
```

Figure 12: Firebase Delete User

Figures 8 and 9 show all the code that we wrote as a work around for the admin deletion feature; the work around was to provide an admin enable/disable feature. Nevertheless, as shown in Figure 10, if we wrote a server in NodeJs, Java or Python, we could have simply used the code snippet in Figure 10, to delete a user. The reason we could not do this in ReactJS is that the "Firebase Authentication" API depicted in Figure 10 is not available in frontend languages at this time. Overall, using the "Firebase Authentication" API to delete Firebase users would have been much cleaner than our current implementation that we have; the downside of this strategy is that it requires writing and deploying a server to perform the Firebase Authentication API calls, which was not possible given the short time constraints. Going forward, future developers with more relaxed time constraints might consider refactoring the code to utilize this alternative approach.

10. Conclusion

This report has reviewed the software engineering practices that our team used while developing the Juvenile Justice System educational platform. The platform will present convicted juveniles with an accessible means of educating themselves about the justice system. We described the admin-facing dashboard features that we have delivered. We have provided a detailed review of our design as well. We reviewed the project development timeline, which included client meetings, coordinations with the other team, and milestones for features. We have successfully completed our planned timeline of events, and have delivered the project to the clients on time.

11. References

[1] Cavanagh. C., Cauffman., E. (2017). “What They Don’t Know Can Hurt Them: Mothers’ Legal Knowledge and Youth Re-Offending.” Psychology, Public Policy, and Law 23(2), pages 141-153.

[2] Kupchik. A. (2004). “Children in an Adult World: Prosecuting Adolescents in Criminal and Juvenile Jurisdictions”. U.S. Department of Justice.

[3] Verani, A., Jyoti, N. (2019). “Overview of the Juvenile Justice System in the Los Angeles County”. Handout discussed during UCLA CS130 client presentations.

[4] “React: A javascript library for building user interfaces.” https:// reactjs.org/.

[5] “Figma vs Sketch - What Sets Figma Apart?” *Figma*, https://www.figma.com/figma-vs-sketch/.

[6] “Firebase: A comprehensive mobile development platform”. https://firebase.google.com/.

[7] Sommerville, I. (2011). Software Engineering. Pearson education, 9 edition.

[8] “Online Diagram Software & Visual Solution.” *Lucidchart*. https://www.lucidchart.com/.

[9] “Jest: a delightful JavaScript Testing Framework.” https://jestjs.io/.

[10] “Trello: work more collaboratively and get more done.” https://trello.com/en-US