INTRODUCTION

Team:

Anna Campainha

Alyssandra Cabading

Angela Lau

Glen Larita

Team Name: SafeG3Ar

Programming Language: JavaScript

Type of Program: Web Application

For this project, we have formed a team consisting of four student developers: Anna Campainha, Alyssandra Cabading, Angela Lau, and Glen Larita. Because this project is designed to focus on the security and privacy aspects of development, we have decided to name our team "SafeG3Ar". We will be creating a web application called "*GID the Planner"* which is a daily planner for users. Users can create an account and plan out their day, add a to-do list and save the data to their account for viewing later. Users will then be able to view their planner and to-do list by logging into their account, where they will be presented with a calendar and a to-do list that they will be able to edit or add data. Users will also be able to make edits to their profile and update information.

For this application, we will not enable users to have the ability to view other users' planners or to-do lists, as the purpose of the application is for the user to create personal reminders and updates for themselves. As of right now, the only tools that we will be using are the selected programming language JavaScript and our preferred IDEs. But as the project's development progresses we will be incorporating more tools to create our web application.

REQUIREMENTS

Security and Privacy Requirements

As stated in the section above, users will be using our web application to plan out their daily activities and add information to their to-do list that they may intend to keep personal or private. The main security issue that our application presents is the disclosure of personally identifiable information stored on our application. Because of this, our group intends to add security and privacy measures to ensure all information entered by the user is secure, preventing the ability of attackers to access this data while also providing a good experience for the user. On top of ensuring security measures are implemented, we will also lower the risk of information disclosure, provide implied and informed consent throughout the application, and give users the ability to delete or edit their inputted information. Because the purpose of the application is to help the user keep track of their daily life, we plan to have the user's inputted information only visible to the user, who also has to be logged in to their account to view the information. Users of the application should not be able to have access to view or edit any other user's information.

Since our application will be storing personally identifiable information such as the name and phone number of the user with a possibility of gathering sensitive personally identifiable information (PII) from the "to-do" portion of our application, we will add a simple layer of authentication by having the user create a username and password to login to their account. To build on top of this, we will also have the user input their phone number and email when their account is created. With this, we will be able to implement a two-factor authentication system that will send some sort of code or notification to the user's phone or email, where the user can approve of the notification. This will help to ensure that the user logging in is who they say they are, minimizing the risk of having an attacker or entity using another user's account to access information. We also plan on implementing some sort of password strength feature that allows users to determine how strong their passwords are.

We also plan on having every new user who signs up with the application be required to acknowledge a form stating that the user will be inputting their personal information into our application. Every user that signs up will not have any authorization to administrative privileges, and all administrative accounts are to be used only by the developers to view information about the website, such as the number of user accounts.

To keep track of potential issues found throughout the development process, we will use Github's project management boards to list any bugs or issues found on our website. By using the kanban style management provided by Github, we will have the ability to stay updated with each other on what issues are present and whether they have been resolved. We can also use this style of management for listing any new security requirements that we have decided to implement into our application, and what actions can be taken to address those requirements. We will also establish regular team meetings to discuss our progress and keep each other informed.

Quality Gates (or Bug Bars)

Knowing what kind of service our application will provide to the user and what kind of information our application will be handling, we can establish what kind of bug bars can be used to provide a severity threshold for the security and privacy vulnerabilities that our application should satisfy. As the development of our application proceeds, we will make adjustments to these thresholds as needed. Below are the bug bars that we have concluded would apply to our program.

Security Bug Bars

Server

Critical:

- Obtaining privileges higher than authorized to a user, results in the user having the ability to view and make unauthorized changes to data and the web application, or the ability to execute arbitrary code (Elevation of Privilege).
- All write access violations and exploitable read access violations. For example, an attacker can read and write a user's to-do list or planner by acquiring higher privilege.
- The ability for attackers to make the PII of users viewable from anywhere on the application.

- Unauthorized access and modification of file system by an anonymous remote client.

Important:

- The ability for attackers to view information from anywhere in the application that is not meant to be disclosed to anyone other than the user of that information, such as personally identifiable information (PII). An example of this would be the attacker enabling the display of PII on a page that is meant to be accessible by users and non-users.
- The user can achieve the ability to view information that is not meant to be disclosed by masquerading as another user.
- Attackers can modify any data or information in the application. This includes the data and information displayed on set pages of the application that may alter the experience of the user or may display inaccurate instructions or information.
- The ability for attackers to bypass security features. For example, an attacker being able to bypass two-factor authentication, or access a web page that they should not have access to.
- Anonymous remote clients can establish numerous sessions without session timeouts, resulting in a denial of service (DoS) due to the consumption of all the server's available resources.

Moderate:

- Disclosure of information that is not meant to be disclosed from a specific location or URL on the application. For example, being able to view anonymous data of a user through a link that is not meant to be publicly viewed.
- The attacker or entity can masquerade as a random user to access information not meant to be disclosed to anyone other than the owner of that information.
- Temporary modification of the web application and any data or information displayed on the website. This modification will disappear after the application is restarted.

Low:

- Information disclosure that is not targeted towards a specific user and information is anonymous data.

Client

Critical:

- The ability for the user to obtain higher privileges than assigned, resulting in the ability to run arbitrary code or make modifications to data of other users such as PII.
- The ability for the user to have unauthorized access to view and modify other users

Important:

- Instances where an attacker can view information of the application or users on the application where the information was not meant to be disclosed.
- Unauthorized file system access. For example, if the application gained access to the user's local file system without any kind of consent.
- The ability for the attacker to display a UI that is visually identical to the application's UI that users must rely on to make trust decisions, but has differing functionality that the original page UI.
- Permanent modification of application or information used to make valid trust decisions. This modification is still present even after restarting the application.
- Authenticated users can access and write to the file system.

Moderate:

- The ability for attackers to modify user data or application settings.
- An authenticated user can establish numerous sessions without session timeouts.

Low:

- Temporary modification of data on the application or of user information that does not persist after restarting the application.

Privacy Bug Bars

End-User Scenarios

Critical:

- Collection and persistent storage of sensitive personally identifiable information (PII) without prominent notice and consent.
- Users cannot access, modify, and/or remove sensitive PII from the user interface.
- Users can access, modify, and/or remove sensitive PII that is collected and persistently stored in a database, without a proper authentication mechanism.
- Lack of age verification at user account creation.
    - Users must confirm they are over the age of 13 to create an account.

Important:

- Collection and persistent storage of non-sensitive PII without prominent notice and consent.
- Users cannot access, modify, and/or remove non-sensitive PII from the user interface.
- Users can access, modify, and/or remove non-sensitive PII that is collected and persistently stored in a database, without a proper authentication mechanism.
- Lack of ‘forgot password’ feature.
- ‘Forgot password’ feature does not have additional security measures.

Moderate:

- The temporary storage of non-sensitive PII is accessible without a proper authentication mechanism.
- Lack of two-factor authentication system at user login.

Low:

- Lack of ‘password strength/weakness’ feature.

Risk Assessment Plan for Security and Privacy

Knowing the bug bar and the security quality gates established above, we can then determine how to assess security and privacy concerns within our application that will allow us to follow the thresholds established. We believe that we should lower the number of scenarios described in the bug bar section as much as possible, to be able to deploy the application with bugs no higher in severity than “low”. Below is a checklist of behaviors created by the Microsoft SDL process in their SDL privacy questionnaire that aids in determining the privacy impact rating of our application. All behaviors that are checked off are present in our application:

- ✓ Stores personally identifiable information (PII) on the user's computer or transfers it from the user’s computer (P1)
- ✓ Provides an experience that targets children or is attractive to children (P1)
- ✗ Continuously monitors the user (P1)
- ✗ Installs new software or changes file type associations, the home page, or search page (P1)
- ✗ Transfers anonymous data (P2)

For this project, we plan to have all developers be equally responsible for privacy. We decided to take this approach so that we have more minds to work with to provide a more secure application. The more minds available, the higher the likelihood of discovering any bugs or privacy concerns as the project progresses. We will have until May 2, 2022, to complete and fully release this application. During this time, our team will work diligently towards providing an application that not only assists users in their daily planning but does so securely and privately.

To reach this goal, we started by reviewing the risks that our application presents. This mostly consists of disclosure of personally identifiable information, unauthorized modification of the web application due to bugs such as providing more privilege than intended to a user and spoofing by masquerading as another user and obtaining information about that user. As stated above, the PII that will be handled in our application includes full name, email, and phone number. Our application also will handle anonymous information, such as those entered into the to-do list and daily planner. Knowing these are most likely the main vulnerabilities in our application, there are a few security features that we plan on implementing into our application that will lower the risks listed above.

To address the issues with spoofing and disclosure of information, we plan on implementing two-factor authentication. This feature will be implemented upon the user logging in, where they will be prompted with a page that will enable them to send a code to their email or phone. Once sent, they will be required to enter this code into the application before proceeding to fully log in. We will also implement a password strength/check determination function, that will help the user come up with a strong password so that it is less likely that attackers will be able to retrieve and use a password.

We also plan to have information viewable to only the owners of that information. They will only be able to view this information if logged into our application. If an individual is not logged in to our application, they will not be able to view any data or information other than what is displayed on the homepage. Users will also have the ability to remove or change their information, such as those entered into the planner and to-do list. We will also require that users must be over the age of 13 to use our application by asking the user for their age when they sign up.

Due to these risks, it is best to focus the threat model on those areas of our application to help with analyzing these threats and taking the appropriate actions against those threats. As our application development process moves forward, we will adjust our threat models accordingly to the different security risks we find, to address as many security concerns as possible.

DESIGN

Design Requirements

The main design requirements will be focused on ensuring the user understands the intended functionality of each application feature. An example of this is how the user will access their digital planner securely.

This is achieved by implementing an authentication feature that requires the user to access an existing account or create a new one, as mentioned in the *Security Requirements*. This initial request will be prompted onto the landing page of the site where users are instructed to either select "*Sign-In*" or "*Sign-Up*". If the user selects "*Sign-In*" they are prompted to the respective page. Here they are to provide their existing username and password to securely access their existing planner.

The "*Sign-Up*" page on the other hand will inform a new user of the steps to create an account to access their prospective digital planner. On this form fillable, the user must provide their *First* and *Last Name*, a valid *email*, *phone number*, and *password*. To ensure the user understands the intended use of the application they will be presented with the *Terms of Agreements*, to which they must agree to successfully create their account.

The design requirements must be clear and intentional. In other words, there must be little to no room for the misconception of the application features on the user's end. The interface must be constructed in such a way that the intentionality is upfront for a potential user who may not be familiar with such systems. This can be achieved simply by using familiar symbols, short information messages, legible fonts, and even adequate use of contrasting colors. However, it is of high priority for users to be able to easily follow through with any necessary procedures, especially ones that deal with the security aspects of the application. This is further achieved by the functional and non-functional requirements of the application.

The functional requirements of the application are what the end-user expects the system should offer. Whereas the nonfunctional requirements are described as the quality attributes of the system. In simpler terms, it is "how" the system would work. For instance, a functional requirement of the application is the authentication of the user when they log in or create their existing account. Its respective nonfunctional requirement is for the system to return a message response informing the user of the results of their query. Possible system responses include*: "The username entered does not exist", "Incorrect password. Try Again.", "Successfully logged in"*. An extra precaution for returning users would be two-factor authentication via email or phone to ensure that the correct user is accessing the account. Another non-functional requirement applicable to this is a potential email verification when a user registers for the first time. This serves as an additional security measure to ensure that the end-user is not an automated bot.

Another functional requirement is password recovery capability. A great portion of end-users may find themselves unable to re-access their accounts due to a forgotten password. On account of such situations, the system must provide the users an option to recover access to their account, often presented on the sign-in page as a simple “*Forgot password?*” message.

Further design requirements are the navigation capabilities of the end-user. Once the user has successfully logged in to their account they should only be able to access pages relative to their data. This data includes their existing scheduled “to-do” list and on-file records of their *name*, *password*, *phone number*, and *email address*. The design requirements must also make it possible for users to easily log out of their accounts manually using a logout button. As a security measure, the system must also account for an automatic logout protocol when the browser has been exited.

Attack Surface Analysis and Reduction

The Attack Surface Analysis will ensure the app by assessing and testing for security vulnerabilities. The data in the application will be a part of the Attack Surface. The key components of the applications Attack Surface will be authorization, authentication, password, user validation, personal notes, and data that is collected in the app. Aside from the user's data, the data within the application and additional packages implemented in the application must be assessed before implementation. Some of the packages require secret keys that can be accessed through the source code. By configuring the settings.json file we will be able to prevent any possible leaks of secret keys. Below lists the critical categories that will be in our Attack Surface Analysis and Reduction:

- Authentication: *Sign-In*

  Reduction:
  - User roles: Admin or Regular user
  - Hacker vs. real user

    - Password entry limit or a two-part authentication
- Admin Database

  Reduction:
    - Authenticate roles to users
    - Add in extra steps to become an admin
- Personal notes, locations, and information

  Reduction:
    - Secure the information to each user
- Packages

  Reduction:
    - Check for vulnerabilities in packages
- External Parties

  Reduction:
    - Deployment of the app: check if the site releases the source code through dev-tools

Threat Modeling

Identifying and *Categorizing Threats*

<u>*STRIDE*</u>

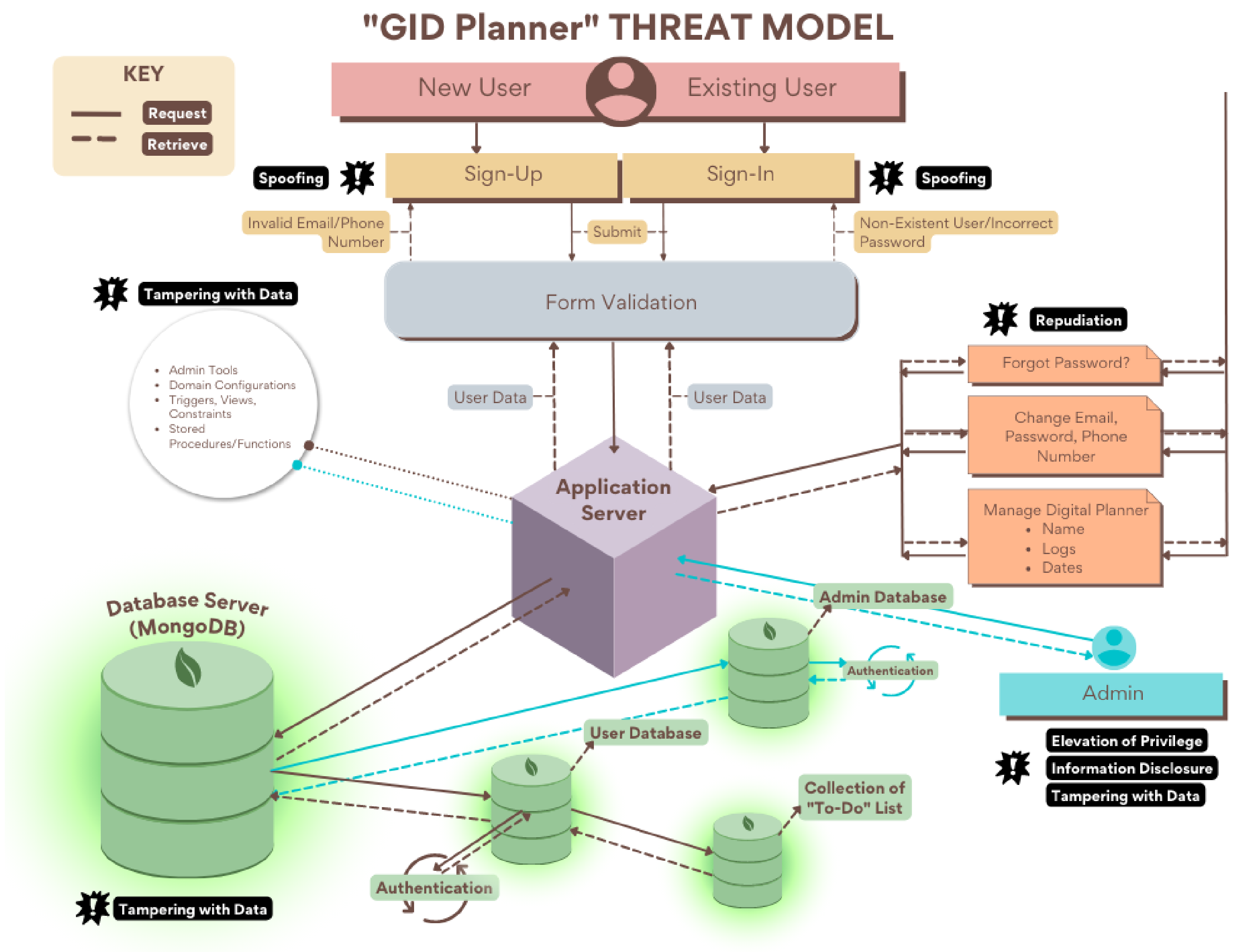
- Spoofing Identity
- Tampering with Data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege

Sign up and Sign in Threats

1. An attacker posing as an Existing User or Admin. (Spoofing Identity)
2. Attacker requesting to override password access of the existing user. (Repudiation)

Server Threats

1. Attackers can modify data through spoofing or running the external code of the web application. (Spoofing and Tampering)
2. The attacker can acquire higher privileges than assigned and can view and modify data in the database or the application. (Elevation of privilege)
3. Unauthorized information disclosure occurs through an attacker. (Denial of Service)
4. Tampering and modification of data on web applications where the information modified are to be trusted by the user. (Tampering)
5. Interruption of the flow of traffic to filter desired encrypted data via an unsecured Wi-Fi network. (Tampering)



"GID the Planner" Threat Model

IMPLEMENTATION

Approved Tools

Our application will be implementing a variety of different tools to not only ensure that our development process is smooth and efficient but also ensure that we limit the number of bugs and security vulnerabilities in our application. Our development of these applications will be done primarily in Javascript and HTML, so many of the tools listed below will enhance our development while utilizing those languages. The tools that we have decided to implement into our application also include the use of static code analysis and potentially dynamic code analysis. Listed below are the tools that we will be utilizing:

DevSkim version 0.6.9

- DevSkim is a static analysis framework that is used as an extension or plugin to an IDE that tests for security vulnerabilities in an application as the developer writes code ("*Devskim",* 2022). DevSkim supports many coding languages, including the languages being used for our project.
- For our application, we are using Visual Studio Code as our main code editor due to its ability to install extensions and enhance the development process. DevSkim is one of the extensions offered on Visual Studio Code, and our team has proceeded to install the extension. This extension allows us to write secure code by underlining any code that could cause a security vulnerability with squiggly lines. If the mouse hovers over the error, a description of the issue and a "how-to address" description are displayed. This enables our team to address a vulnerability as soon as it is written (Stocco, 2021).

ESLint version 7.32.0

- Another static analysis tool our team will use is ESLint, which is a linting framework that finds problematic code when writing Javascript. It allows for customizability and generates automatic fixes so that the developer can find errors in their code before running it ("*About ESLint*", n.d).
- Our team is using ESLint to find syntax errors and functional problems with the javascript code we write. This ensures that we write code that we know is functional before we run the application, helping us narrow down any bugs found during runtime. This also allows us to code using the same coding standard for a more compatible and seamless development process.

Iroh.js version 0.3.0

- A dynamic analysis tool that our team will be applying to our application is Iroh.js. This tool allows developers to analyze the functionality of code and collect data during runtime, which can assist with debugging behavioral issues of the program (Maier, 2018a).

- Although the package is no longer frequently updated, according to *Snyk.io*, the package has had no known security issues (“*Iroh - NPM Package Health Analysis”,* 2018). Our team will use Iroh primarily for analyzing code that is involved in the functionality of the application, as well as making sure data that is being passed throughout the application is the correct data. We may also use Iroh to test functions to make sure they are working as intended.

TestCafe version 1.16.0

- Our team intends to implement the use of TestCafe to write an acceptance test for our application and make sure that it runs in a browser environment as intended before deployment. Throughout the development process, we will continuously write scripts to make sure that the functionality of our website performs as we intend it to. We can also use it to debug and pinpoint an issue in the code that is causing a test to fail.

Npm Audit

- Our team will implement the use of the npm audit command throughout the development process to ensure that all packages being used are free of vulnerabilities (“*NPM-Audit”,* n.d). Our team will use npm audit whenever a new package is installed into our application so that our team can keep track of any possible vulnerabilities introduced into our project. The current version of npm installed in our application is 8.3.1.

The rest of our approved tools used in the development process consists of usages for establishing a local server and client for running our application locally, storing our mock data into a database, and the use of Javascript libraries to increase our productivity when designing and implementing functionality to our application. These tools include:

- React.js version 17.0.2
- Meteor.js version 2.6.0 → Meteor.js version 2.7.1 (Updated Version)
- MongoDB version 5.0.5 (for our database)
- Node.js version 12.18.4 (for our local client/server)
- Visual Studio Code version 1.64.1 (IDE/Text Editor)

- Semantic UI React version 1.3.1
- Material UI version 4.12.3 *UPDATE: Stopped all further implementation on 04/11/22

Deprecated/Unsafe Functions

Along with the approved tools listed above are deprecated or unsafe functions that come along with them. Our team has assembled a list of these deprecated functions from some of the tools that we will be using for the development of this application. Please note that this list is not a complete list of all the deprecated functions present in each tool, but rather a list of the deprecated functions that we may implement into our application, and the safer alternatives/adjustments that we can utilize as their replacement.

- React.Js version 17.0.2
    - When using the ".addEvenListener" function, use the capture phase "{ capture: true }" as the third event argument, which helps with detecting events from react components that call "e.stopPropagation()" (Abramov & Nabors, 2020).
    - A couple of event systems have been deprecated and changed:
        - *onFocus* event has been switched to *focusin* (Abramov & Nabors, 2020)
        - *onBlur* event has been switched to *focusout*
- Meteor.Js version 2.6.0
    - Keys for IOS devices on App.launchScreens are now deprecated, and will be replaced in favor of new storyboard compliant keys ("*Changelog | Meteor API Docs",* n.d).
    - Mongo
        - *useUnifiedTopology* is no longer an option and is set to true.
        - *poolSize* is no longer an option. Must use *max*/*minPoolSize* for the same behavior ("*Changelog | Meteor API Docs",* n.d).
- Material UI 4.12.3 (UPDATE: Stopped all further implementation from 04/11/22)

- Originally, our team decided on using Material UI 4.12.3. However, upon further inspection and research, we have learned that MUI v4 has been deprecated and is no longer maintained (“*Material-UI/Core”,* 2021).
- We will now begin the implementation of Material UI version 5.4.2, which is the latest version of MUI.
- This version of MUI offers more stability, customization, bug fixes, and security patches that version 4 did not offer (“*Material-UI/Core”,* 2021; ”*Migration from v4 to v5*”, n.d).
- Use “<Box>” component and “sx” instead of “styled()” for components as it offers more efficiency and consistency (Tassinari et al., 2021).

Static Analysis

Static Analysis is a key part of any application development process. It is what allows us, developers, to perform *code reviews* while developing the different components of our system. The role of static analysis serves as a step toward maintaining secure code. The key characteristic of static analysis is “[the examination of] the driver code without running the driver”. This is especially helpful when we need to evaluate code paths that are relatively difficult to perform time-extensive practice runs on.

During this phase of the development process, the two main static analysis tools our group utilized were ESLint and DevSkim. These two plugins were the most ideal tools as they were fairly familiar plugins that are made readily available in our code editor, *Visual Studio Code*.

ESLint allowed us to analyze our code and quickly identify quality and style issues, such as incorrect use of syntax or functional dependencies. Similar to grammar and spelling checks in most document systems, the lines of codes with errors are brought to attention with a simple red squiggly line. This simple use of error-catching allowed us to carry out one of the following actions: 1) Manually fix the syntax/functional errors. 2) Utilize the suggested actions to be taken, when the mouse hovered over the line of error. The plugins also made it possible for automatic issue fixes with a simple use of the “*command”* key followed by a period. But this approach wasn’t a favored implementation as it often included error fixes that turned out to be much more problematic for our code during run time. ESLint has been especially helpful with ensuring that the coding standard and style of our system are consistent and compatible with the version of the React component we are utilizing. For instance, any invalid hook calls present in our code were brought to attention immediately. We didn't have to rely on error detection during run time to identify those specific errors. An added benefit of ESLint is that we do not have to rummage through every file to find all the red squiggly lines. We simply invoke `npm run lint` on the terminal under the appropriate application directory, then all the detected errors are listed. From here we can simply access the location of the error with the simple click of a button.

DevSkim is a slightly newer plugin for us, but after further reading on its usage, we were convinced that it was a great starting point for detecting potential security vulnerabilities in our code. Similar to ESLint it is a static verification tool that doesn't require the driver to be running to provide inline error detection. It highlights potential security flaws, similar to ESLint detection errors, with a red squiggly line. To decipher the issue description of the underlined error, we are to simply hover our mouse over the line of error. Occasionally, safe alternatives are made available in the lightbulb menu located to the left of the error line of code, which allows us to fix the issue with a few clicks. Furthermore, a great feature of DevSkim is that "guidance for the parameters will be inserted in the form of <some guidance info>" (*DevSkim*, n.d.). These are for the cases in which alternative suggestions have different parameters than the unsafe API call initially used. This is especially helpful for developers to gain an understanding of the structure might they be unfamiliar with such usage.

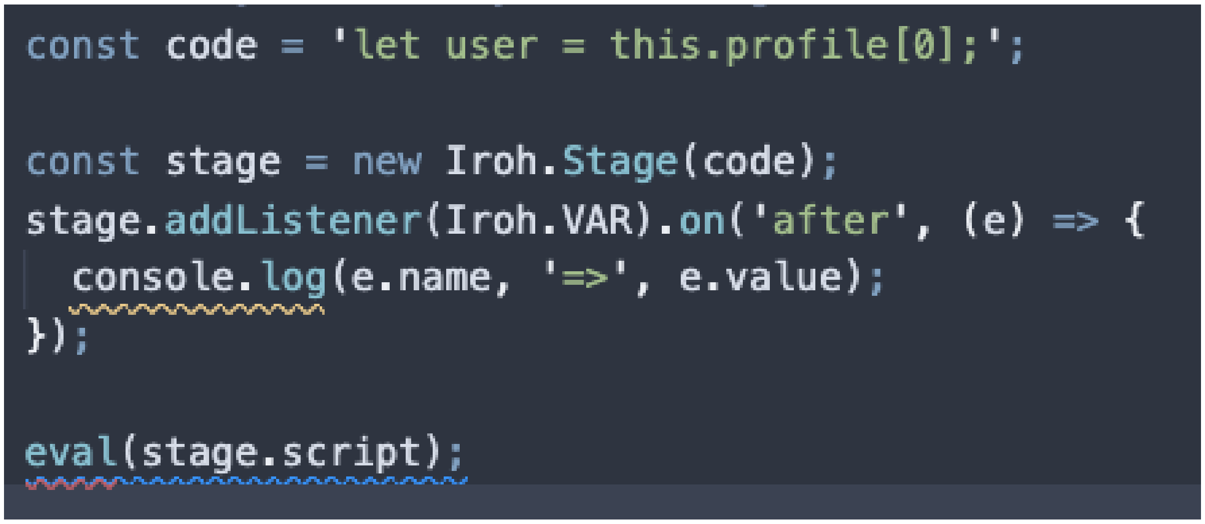
Up to this point, ESLint has been an active plugin that we consistently use throughout the code development process. DevSkim on the other hand has not raised many error codes thus far. While this is a good sign that there are no current present security issues detected, we must take into consideration that we cannot simply rely on it. As we recall, static analysis tools are convenient for detecting source code errors not during run time, but they can often generate false-positive results that are otherwise detected by dynamic verification tools. Therefore, in future phases of the project, we hope to implement some sort of dynamic analysis to further secure the robustness of our code. This can be accomplished by the implementation of the *Iroh.js* tool and further human review analysis.

VERIFICATION

Dynamic Analysis

Along with the static analysis, our team also has decided to include the use of dynamic analysis to further enhance our development process. The dynamic analysis tool that our team has chosen to implement is Iroh.js, which is an API package that allows you to view and intercept runtime information, as well as visualize runtime data and collect information that will help our team understand data moving throughout our application (Maier, 2018b).

After testing Iroh with some of our variables and functions in our web application, we have found it to be a very useful tool that will contribute greatly to our development workflow and the understanding of our application's behavior. Below is a screenshot providing an example of how Iroh was implemented into our web application to check if a variable was holding the correct data.

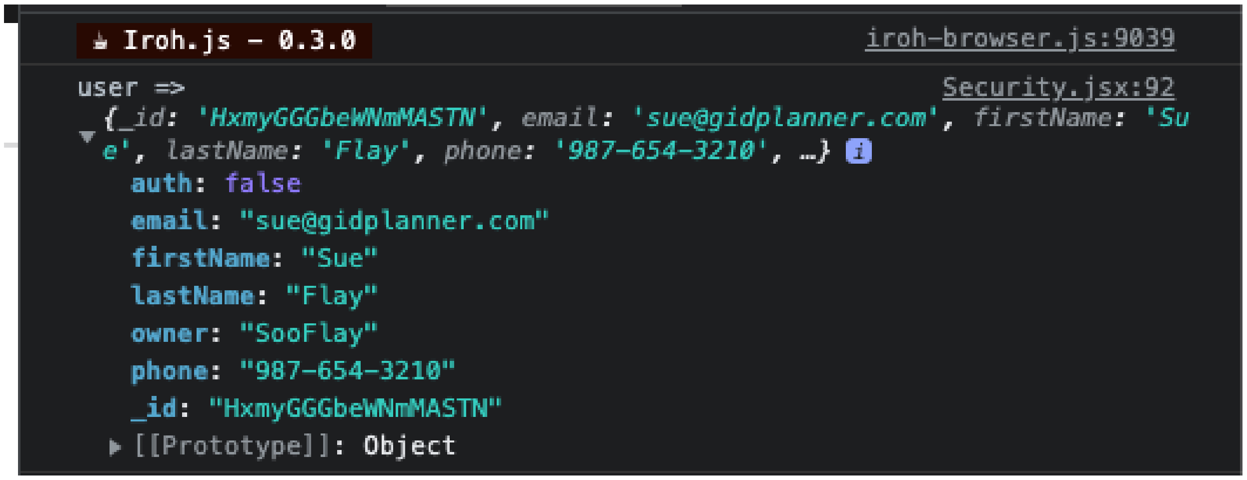


```
const code = 'let user = this.profile[0];';

const stage = new Iroh.Stage(code);
stage.addListener(Iroh.VAR).on('after', (e) => {
  console.log(e.name, '=>', e.value);
});

eval(stage.script);
```

The "user" variable is meant to hold the currently logged in user's profile object that was originally passed to the class as a prop and filtered down, which will then be used in later functions that are crucial for the user to allow enabling of two-factor authentication or disabling of two-factor authentication. The Iroh code creates a stage object to connect the running code and the Iroh code, which Iroh will then use to check the contents of the variable after it has been declared and initialized (Maier, 2018c). When the program is run and when we visit the page where the code is located, the profile object should appear in the console as the Iroh code was placed in the render prop.



```
Iroh.js - 0.3.0                                             iroh-browser.js:9039
user =>                                                          Security.jsx:92
▼ {_id: 'HxmyGGGbeWNmMASTN', email: 'sue@gidplanner.com', firstName: 'Su
  e', lastName: 'Flay', phone: '987-654-3210', …}
    auth: false
    email: "sue@gidplanner.com"
    firstName: "Sue"
    lastName: "Flay"
    owner: "SooFlay"
    phone: "987-654-3210"
    _id: "HxmyGGGbeWNmMASTN"
  ▶ [[Prototype]]: Object
```

As you can see in the above screenshot, the currently logged-in user's profile object is correctly stored and printed to the console. Although this is code that should not be kept in the program for security purposes, it is very helpful for making sure that variables and functions are working as intended during the development process.

Although Iroh is a very useful tool, our team did run into some issues while implementing it into our application. Since Iroh has not been recently updated, it does not have full support for new technologies, such as ES6's arrow functions. Because of this, our team has had some issues testing functions using ES6 arrow function syntax. For future functions, our team will have to be sure to convert any arrow functions to normal Javascript syntax when using Iroh. Unfortunately, this may not always be possible as some of the functions implement the use of callback arrow functions as a parameter. Another issue that our team ran into was the actual implementation of Iroh into our project. It seemed that the package was not able to be imported into a *.jsx* file using node syntax, so instead, we had to import it in a script tag from the client's *main.html* file.

Regardless of some of the compatibility issues that Iroh presents with our application, we find its features useful for our development process and will continue to make use of its features when possible throughout the future of our application's development.

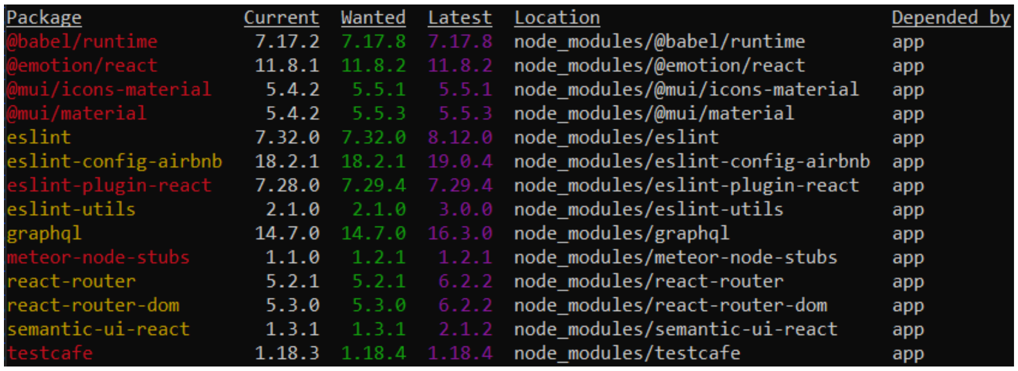
Attack Surface Review II

Our Attack Surface has remained consistent with holding our app to the top consideration of the users' data and experience. We have implemented a two-step authentication that will protect from hackers trying to access a profile that is not theirs. During the development of *GID*, we have added packages to implement components of the application. The packages we have installed have 0 vulnerabilities. We updated Meteor to version 2.7, which included a new core package – `accounts-2fa` that provides support for two-factor authentication. The other packages that were upgraded in this version change did not significantly affect our application; aside from new support for 2FA, the update resolved some minor issues and bugs. We have also updated our *npm* version from 8.3.1 to 8.5.5, and its list of dependencies did not affect our application. Running `npm audit` returned 0 vulnerabilities.

The following packages and updates have been made:

- Update JavaScript Framework *Meteor* to 2.7-rc.4 `meteor update --release 2.7-rc.4`
- Update to newest version of npm (8.3.1 -> 8.5.5) `npm install npm@8.5.5`
- Installed Package for two-factor authentication `meteor accounts-2fa`
- Installed MUI (Front-End) Packages `npm install @mui/material @emotion/react @emotion/styled @mui/lab @mui/icons-material`
- Installed package `npm install buffer`

By running `npm outdated,` we saw that there were 14 outdated npm dependencies in our package.json file. We will review the version documentation for each package, and carefully update them as needed in our next project milestone. A list of the packages that we currently run and should update can be found below.


```
Package                Current  Wanted  Latest  Location                              Depended by
@babel/runtime          7.17.2  7.17.8  7.17.8  node_modules/@babel/runtime           app
@emotion/react          11.8.1  11.8.2  11.8.2  node_modules/@emotion/react           app
@mui/icons-material      5.4.2   5.5.1   5.5.1  node_modules/@mui/icons-material      app
@mui/material            5.4.2   5.5.3   5.5.3  node_modules/@mui/material            app
eslint                  7.32.0  7.32.0  8.12.0  node_modules/eslint                   app
eslint-config-airbnb    18.2.1  18.2.1  19.0.4  node_modules/eslint-config-airbnb     app
eslint-plugin-react     7.28.0  7.29.4  7.29.4  node_modules/eslint-plugin-react      app
eslint-utils             2.1.0   2.1.0   3.0.0  node_modules/eslint-utils             app
graphql                 14.7.0  14.7.0  16.3.0  node_modules/graphql                  app
meteor-node-stubs        1.1.0   1.2.1   1.2.1  node_modules/meteor-node-stubs        app
react-router             5.2.1   5.2.1   6.2.2  node_modules/react-router             app
react-router-dom         5.3.0   5.3.0   6.2.2  node_modules/react-router-dom         app
semantic-ui-react        1.3.1   1.3.1   2.1.2  node_modules/semantic-ui-react        app
testcafe                1.18.3  1.18.4  1.18.4  node_modules/testcafe                 app
```

SDLC: VERIFICATION

Fuzz Testing

For our application, both manual and automated hacking techniques were used in an attempt to crash and break into our application. Three of the techniques that we utilized were password cracking, trying to force access to parts of the web application with different privileges, and attempting to access different URL links outside of the user's account.

Password Cracking

To ensure that passwords for the accounts created are secure, we implemented the use of John the Ripper, which is a password cracking tool that is used to do password security audits and password recovery ("John the Ripper", n.d). It can be used to try and break programs using several passwords cracking tools that come with it (Chandel, 2018). We incorporated John the Ripper with our application by first creating a file that held three test cases of username and passwords that had been used with our web application. Before running the file with John the ripper, we had to hash the passwords with the use of the SHA1 algorithm. Once the passwords were hashed, we then ran *John the Ripper* with the target file. John will first use a word list that is provided when it is downloaded that contains a large list of commonly used passwords. If the list has been iterated through and the password has not yet been cracked, it will then start to use brute force by testing values using ASCII incrementally.

The first test case utilized one of the earliest default passwords we had for the default users. Almost instantly, *John the Ripper* was able to crack the password. We then tested a newer password that we had assigned to the user accounts for the second test case. This took much longer, and *John the Ripper* ended up resorting to incremental ASCII in an attempt to crack the new password. After an hour of running, John was not able to crack the password and the session was terminated due to the likelihood of the test running for a long period. We determined John was not able to crack the password because the password was more personalized to the default users and application. The last test case utilized the username and password that was tested in the first case but with added special characters and capitalization to it. This session ended up being very similar to the second test case, and John the Ripper was not able to crack the password in a reasonable amount of time.

From this test, John the Ripper was not able to crack the passwords provided to it two out of three times. Furthermore, it was unable to crack nearly the same password that it was able to crack in the first case, just with small differences in capitalization and special characters. Due to this, we have concluded that if time permits, we will implement some sort of password check when the user signs up with our application that ensures the user enters a password that contains a special character and a number. This will result in a more secure user account. On top of this, they will already have the ability to enable two-factor authentication, which will add another layer of security to the user's account.

Access Privileges

Each user will have a unique role as a user or an administrator. Admin users will have access to view all users in the web application but none of their personal information. A user will only have access to their personal information and will not be able to view other users' information. Our code distinguishes between users by assigning a "role" to the users signed up. The users "sue@gidplanner.com" and "obi@gidplanner.com" are assigned the role of a *user* while "admin@gidplanner.com" will be assigned the role of *admin*. Both roles have a different user experience than the other because of their access privileges. One of our main goals is to keep user information secure from other users, and utilizing the accounts package from meteor enables us to do so.

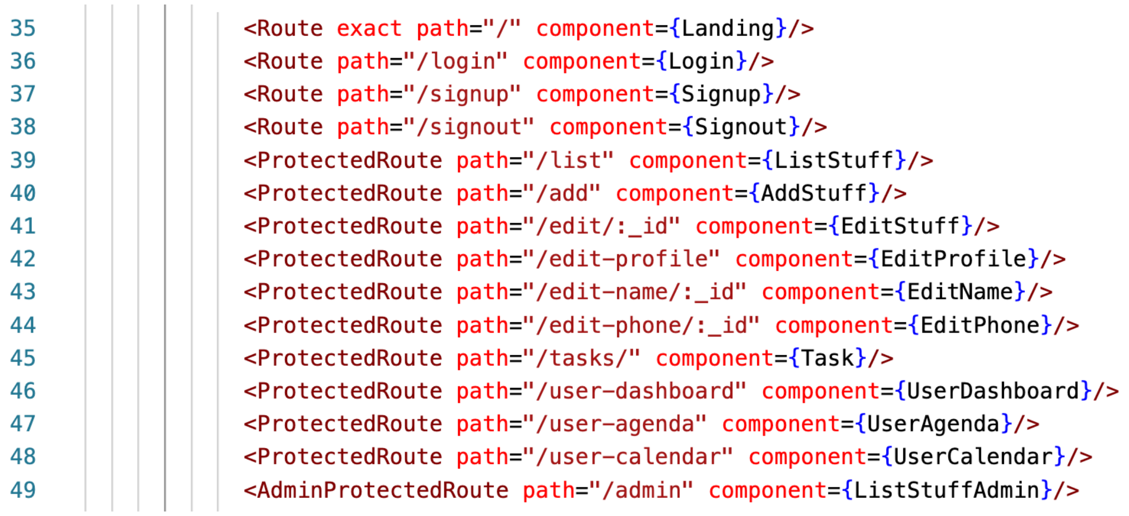
The admin can log in under the email “admin@gidplanner.com” where they will be able to access their NavBar and information unique to the username while the user will have a different layout. The users' daily lists and calendars will be private and the admin will not be able to manage that information. The user will be able to log in under their email address where they will be redirected to the “User Dashboard”, which will issue their tasks, lists, and calendar. The user will have a more distinct NavBar than the admin and will have access to adding to their calendar, searching, and returning to their dashboard.

To test this, we logged into our application as one of our default users. We then attempted to access pages only available to the admin accounts. We also made multiple attempts to access the editing information of other users by inputting the URL associated with their profile editing pages. The user was able to successfully arrive at the editing page, however, if the user tried to submit the form they would not be able to do so unless they also provided the other fields of the profile accounts in the form submission. This would not be very unlikely, as even if the user were to somehow obtain the fields to input the other profile information, such as the owner, phone, and docID are all hidden. We also attempted to use various functions in the console to see if a user’s information could be changed from another account by using the information provided in their profile fields. All attempts were unsuccessful.

This is thanks to the security measures that Meteor implements within their “accounts” package. Even when attempting to view the user’s information in the console, Meteor does not show any other information other than the username. Due to this, we believe our team is on the right track when it comes to implementing user privacy with Meteor accounts.

Unique Hyperlinks

In our App.jsx file, we have routed each page other than the landing, login, signup, and sign-out pages to be protected which means it is unique to the user. The ID of the user logged in will be used to identify and sort the data that belongs to them. Each page that the users navigate will follow a hyperlink that is unique to their data. A non-user of that data will not have the same link, since it is secured data from user to user. This is the same situation for the role of an admin. The admin will have access to the list of the users and be able to manage the app with the constraints we implemented for security purposes. As the admin navigates their side of the app each hyperlink will be unique from a non-admin user due to lines 39-49 pictured below where the ‘ProtectedRoute’ and ‘AdminProtectedRoute’ are used for security purposes.



```
35  <Route exact path="/" component={Landing}/>
36  <Route path="/login" component={Login}/>
37  <Route path="/signup" component={Signup}/>
38  <Route path="/signout" component={Signout}/>
39  <ProtectedRoute path="/list" component={ListStuff}/>
40  <ProtectedRoute path="/add" component={AddStuff}/>
41  <ProtectedRoute path="/edit/:_id" component={EditStuff}/>
42  <ProtectedRoute path="/edit-profile" component={EditProfile}/>
43  <ProtectedRoute path="/edit-name/:_id" component={EditName}/>
44  <ProtectedRoute path="/edit-phone/:_id" component={EditPhone}/>
45  <ProtectedRoute path="/tasks/" component={Task}/>
46  <ProtectedRoute path="/user-dashboard" component={UserDashboard}/>
47  <ProtectedRoute path="/user-agenda" component={UserAgenda}/>
48  <ProtectedRoute path="/user-calendar" component={UserCalendar}/>
49  <AdminProtectedRoute path="/admin" component={ListStuffAdmin}/>
```

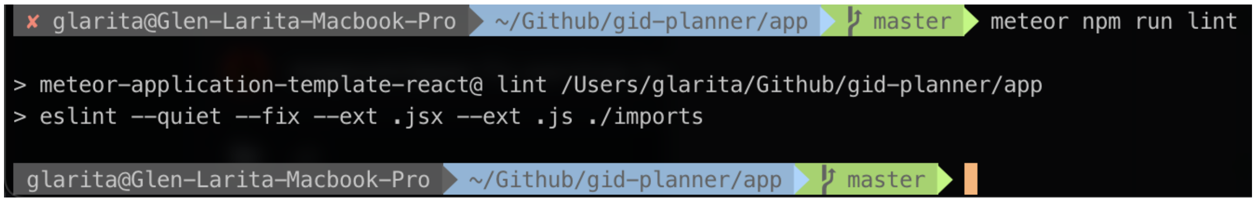
We tested this by inputting multiple URLs of logged-in accounts into the browser while not being logged in as a user. We also attempted to access other accounts and admin accounts while logged in with a user role account. All attempts to access accounts when not logged in were unsuccessful, and all accounts were not able to access information from other accounts using similar methods to the ones used for access privileges. To try and bypass this, we attempted to use Meteor methods in the console to try and print out the user’s information. However, when using methods such as “Meteor. user()”, the attacker will only be able to obtain the username of the currently logged-in user. If the methods are called when the attacker is not logged in with an account, they are unable to view any accounts at all. Even if the user were to somehow get the passwords, Meteor encrypts all passwords made with the “accounts” package with bcrypt algorithm, which helps protect the passwords when there is a leak in the database (“Passwords”, n.d).

Because of this, I believe that our team is taking the appropriate steps toward user privacy with the implementation of Meteor’s “accounts” packages, and we will continue to utilize Meteor’s secure methods included in the package.

Static Analysis Review

In our application, we have continued to utilize both DevSkim and ESLint as our main static analysis tools, both of which have been very helpful in finding any potential errors or security risks in our application.

Our team implements the use of ESLint frequently throughout development. When one of our team members experiences an ESLint error, it is addressed immediately before it is pushed to the master branch. We also use the script "meteor npm run start" before every merge to the master branch to reassure that there are no ESLint errors at the time. Below is an example of running this script in the terminal.



```
✘ glarita@Glen-Larita-Macbook-Pro  ~/Github/gid-planner/app  master  meteor npm run lint

> meteor-application-template-react@ lint /Users/glarita/Github/gid-planner/app
> eslint --quiet --fix --ext .jsx --ext .js ./imports

 glarita@Glen-Larita-Macbook-Pro  ~/Github/gid-planner/app  master 
```

In the image above, no ESLint errors have been found on the master branch, as we consistently make sure the branch contains the cleanest and error-free version of the application.

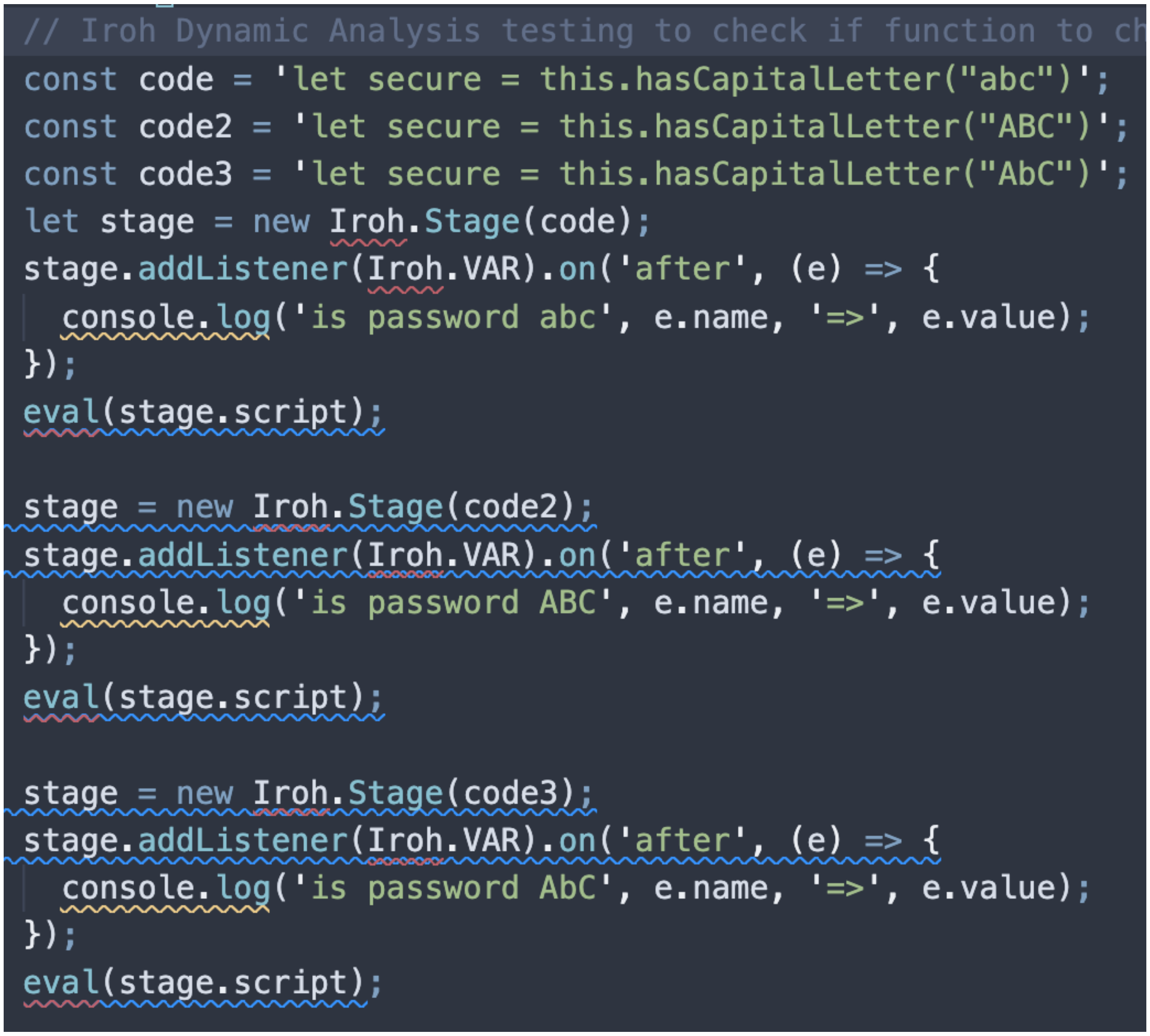
So far, our team has not encountered many Devskim notifications, which is a good thing because that means we are implementing secure code into our application. Out of the few times that we have experienced a Devskim notification during development, we immediately addressed it. Because of this, our application is currently running with no DevSkim notifications present.

Dynamic Review

In our last Dynamic analysis update, our team stated that we are implementing the use of Iroh.js to make sure our application is functioning as intended. Since then, not many functional changes have been made to our application's code due to the limited time frame of this assignment, and thus we have not gotten the opportunity to implement more dynamic analysis on our application. However, we will continue to utilize *Iroh.js* in our project, as we have a few more features that we wanted to analyze and implement into our application, an example being password checking when a user creates an account.

**Update 04/30/2022*

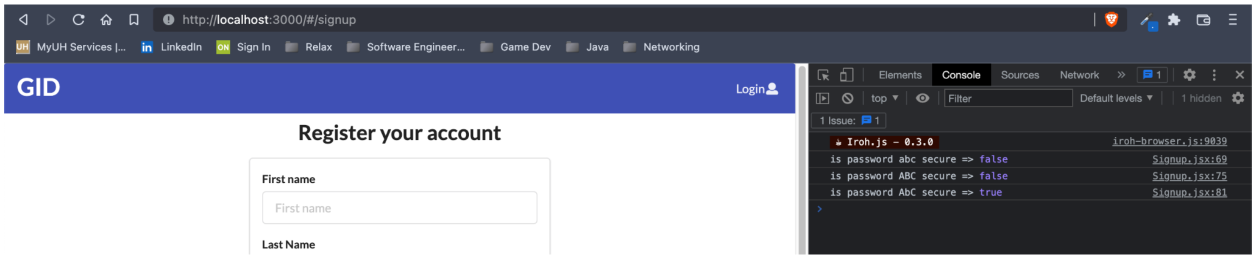
During the implementation of the password checking function, Iroh was utilized in a separate working branch to test if one of the functions used in the check was working as intended. This function takes a string as a parameter and returns true if the string contains both capital and lowercase letters, and false if it only contains all lowercase or all uppercase characters. Below is an example of how Iroh.js was utilized to test the function.



```javascript
// Iroh Dynamic Analysis testing to check if function to ch
const code = 'let secure = this.hasCapitalLetter("abc")';
const code2 = 'let secure = this.hasCapitalLetter("ABC")';
const code3 = 'let secure = this.hasCapitalLetter("AbC")';
let stage = new Iroh.Stage(code);
stage.addListener(Iroh.VAR).on('after', (e) => {
  console.log('is password abc', e.name, '=>', e.value);
});
eval(stage.script);

stage = new Iroh.Stage(code2);
stage.addListener(Iroh.VAR).on('after', (e) => {
  console.log('is password ABC', e.name, '=>', e.value);
});
eval(stage.script);

stage = new Iroh.Stage(code3);
stage.addListener(Iroh.VAR).on('after', (e) => {
  console.log('is password AbC', e.name, '=>', e.value);
});
eval(stage.script);
```



```
Iroh.js - 0.3.0                                iroh-browser.js:9039
is password abc secure => false                      Signup.jsx:69
is password ABC secure => false                      Signup.jsx:75
is password AbC secure => true                       Signup.jsx:81
```

Iroh.js was utilized by storing the code to be executed into a variable. An Iroh stage was then created to allow the code to be run in that variable, resulting in the printing out of the strings and their corresponding boolean values that were returned by the function. As seen in the second image, the string “abc” and “ABC” returned false since they did not contain both uppercase and lowercase characters.

RELEASE

Incident Response Plan

Our team is aware of the threats that are present in the online space, and we understand that if there is a vulnerability it will most likely be exploited. In the case of an incident, our team has come up with an incident response plan to address any potential risks posed to our application. This includes but is not limited to data breaches, denial-of-service, theft, privacy breaches, and failure to meet our commitments to privacy with our users (“2012”, n.d). Our team has decided to assemble a privacy escalation team as follows:

Escalation manager: Anna Campainha

- Responsible for finding the source of escalation and determining its severity.
- Communicate the escalation, its impact, and its validity with all necessary parties (“2012”, n.d).
- Provide the proposed solution and a timeline of when that solution will be implemented.
- Ensure that the escalation is resolved and communicate it with all parties.

Legal Representative: Angela Lau

- Represent the legal aspects of *GID-Planner*.
- Responsible for resolving and activating any legal concerns under the scope of the company and doing so consistently throughout the response process (“2012”, n.d).
- Determine the location of all parties affected by the incident to identify all relevant jurisdictional laws.
- Following resolution actions to any concerns, assess the effectiveness of the incident response plan with the team and revise the plan appropriately.

Publics Relation Representative: Alyssandra Cabading

- Represent *GID-Planner* as its spokesperson, and update the public with information on the status of the company.
- Help *GID-Planner* and its team maintain a good relationship with the public and its users.
- Responsible for resolving any public concerns consistently throughout the response process (“2012”, n.d).
- Have a firm understanding of individual social media platform tools and guidelines to effectively communicate company-related information to the public.

Security Engineer: Glen Larita

- Responsible for implementing the use of exceptional technical skills to resolve bugs or vulnerabilities associated with the source of escalation.
- Assist with coordinating incident response and discovering security measures to improve the incident response (Coursera, 2022).
- Ensure that protection is implemented into the resolution to ensure the incident does not occur again.
- Maintain implemented security measures to ensure a private and safe user experience.

Contact

If you notice any threats or bugs in the *GID-the-Planner* application, please contact Anna Campainha by email at ac62@hawaii.edu.

Incident Response Procedures

Evaluation

- Upon receiving a notification that there is an escalation or incident, the escalation manager will evaluate the incident and determine if more data needs to be gathered (“2012”, n.d). They will also be responsible for determining when the escalation occurred, who experienced it and reported it, and where in the application it occurred.

Gather Information

- The legal representative will advise the escalation manager to determine the optimal course of action in evidence collection and documentation.
- The escalation manager will then gather as much information on the incident as possible. They will cooperate with the reporting party to determine the impact of the incident, expectations of how the incident should be resolved, and any known facts about the escalation (“2012”, n.d);.

Dissemination

- Once enough information has been gathered, the escalation manager will then inform the other team members of the incident, and propose possible solutions to the issue (“2012”, n.d).
- The legal representative will represent the team regarding any legal concerns about the incident and will address them as necessary. They are also responsible for supporting the team through any legal confrontation and making sure that any parties involved in the incident are properly addressed.
- The public relations representative will use the information gathered from the escalation manager and the rest of the team to make announcements on how the team will address the issue. They are to also make sure that a good relationship between all parties is maintained throughout this process. Their goal is to keep the reputation of *GID-the-Planner* as high as possible throughout the response process.

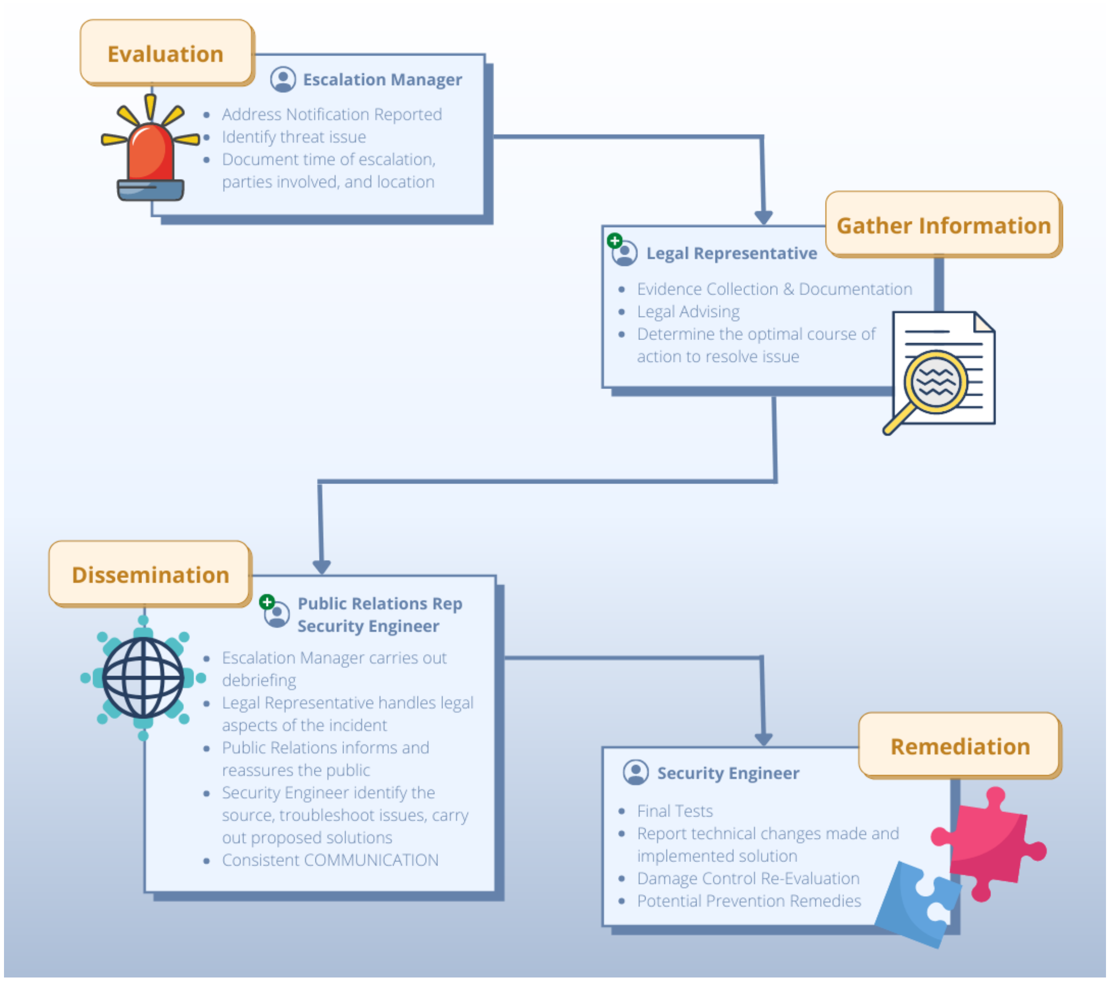
- The security engineer will use the information provided by the escalation manager to pinpoint the source of escalation and troubleshoot the issue. They will use their skills to implement one of the proposed solutions given by the escalation manager to smoothen the response process and to maintain a good reputation.
- During this step in the response process, all team members are to communicate and update one another to ensure that effective communication between all parties involved in the legal and public relations process is maintained. Communication may also prompt the escalation manager to gather more information if necessary, and the security engineer may be able to use any information gained from communication to reach a more effective solution.

Remediation

- Once a resolution has been reached, the security engineer will run tests to ensure that the solution is acceptable and completely resolves the issue. Once the resolution resolves the reporting party's and user concerns, as well as ensures that the incident does not happen again, the security engineer will inform the other team members of the technical changes that were made and how they resolved the problem. The team will then take the appropriate actions based on their roles, and an evaluation of the response process will be done. The purpose of this evaluation is to determine what could have been done to avoid the incident and to discuss any faults in the team's response to the issue. Below is a diagram to provide a visual representation of our team's incident response process.

DIAGRAM — ***Incident Response Procedures in a Nutshell***

SafeG3Ar
Incident Response Procedures
for GID-the-Planner


Evaluation
Escalation Manager
Address Notification Reported
Identify threat issue
Document time of escalation, parties involved, and location
Gather Information
Legal Representative
Evidence Collection & Documentation
Legal Advising
Determine the optimal course of action to resolve issue
Dissemination
Public Relations Rep
Security Engineer
Escalation Manager carries out debriefing
Legal Representative handles legal aspects of the incident
Public Relations informs and reassures the public
Security Engineer identify the source, troubleshoot issues, carry out proposed solutions
Consistent COMMUNICATION
Remediation
Security Engineer
Final Tests
Report technical changes made and implemented solution
Damage Control Re-Evaluation
Potential Prevention Remedies

Final Security Review

Threat Model, Static & Dynamic Analysis

In the *GID-the-Planner* Threat Model, the areas we were most concerned with were the potential threats via the *server* and the *sign-up / sign-in* pages. To combat potential security injections, we implemented a 2-Factor-Authentication option for users to utilize. The two-factor authentication makes use of the Meteor's *accounts-2fa package* and also involves the use of third-party *2FA* applications such as *DUO* or

Google Authenticator. This served as an extra layer of security protection to prevent hackers from accessing user accounts.

Our team has applied the use of both *ESLint* and *DevSkim* as static analysis tools throughout the development of *GID-the-Planner.* These tools assisted us in finding bugs and potential vulnerabilities in our application, as code is being written. Before every merge to the master branch, we verify that our code is free of bugs by running *ESLint*, and if any errors or warnings are found, we address those issues right away. We also implemented dynamic analysis tools into our application with the help of *Iroh.js*, which allowed our team to make sure that functions and variables during runtime were working properly. Resolving these errors as they arise ensures us that at every milestone of development, our application is free of errors and vulnerabilities, as they are much more troublesome to correct if they are passed on to later phases of development.

Quality Gates/Bug Bars

Upon creating an account with *GID-the-Planner,* users will be asked to provide a First Name, Last Name, Username, Phone Number, Email, and Password that must meet the requirements of at least one special character, one number, and a capital letter. This password check was implemented within a function, and our team implemented the use of *Iroh.js* to run a dynamic analysis test on that function to ensure that it was working as intended. After a user has created an account and logged in with the right credentials, they are able to access their personal information from the settings page. As we have disabled Meteor's *autopublish* package, data is published to the client from the server only when it is explicitly specified by us, therefore, privacy-sensitive data can only be accessed by their respective, authorized users. Those who have admin credentials on the site are able to view the names, emails, usernames, and phone numbers of all users but have no access to data found in a user's dashboard or settings page. It is important that not even admins are permitted to view this kind of information because we want to protect the privacy of our users as much as possible. Limiting the scope of data access mitigates the possibility of unauthorized access and malicious attacks.

Another concern we've addressed in our application is the potential for a user to remain logged in, possibly forever. This poses many risks in cases where a user may forget to log out of a public computer, exposing their personal information to any unauthorized persons. A lack of session timeout may also lead to the temporary denial of service if multiple remote clients establish sessions and keep them open, consuming all of the server's available resources. To reduce the chances of these risks, we've implemented a stale session timeout, in which users will be automatically logged out of their account if no activity has been detected for 30 minutes. *GID-the-Planner* users can feel at ease knowing that if they forget to log out, their information will not be vulnerable for long.

Final Security Review Grading

Proceeding *fuzz testings* and performing end-user analysis we concluded that the different features of our program Passed Final Security Reviews. All issues identified before the v1.0.0 release have been resolved. For instance, in the old version of the *accounts-2fa package* (Meteor 2.7), there was a security bug affecting the package. The issue was primarily due to a *NoSQL* injection problem. The Escalation Manager was able to identify the solution via the *Meteor* support blog. The Escalation Manager proceeded to inform the Security Engineer of the security vulnerability and resolve the issue with a simple Meteor version update by invoking

```
meteor update --release 2.7.1.
```

Before the final security review, there were also a few ESLint errors that were relatively minor and simple to fix and didn't pose a significant threat in terms of system security. Furthermore, we ensured that all aspects of the 2FA proceeding with the latest Meteor update were still functioning seamlessly and presented with no further issues. After we have thoroughly reviewed the system through end-user analysis as well as static analysis, we determined no further issues in the immediate security features.

Certified Release & Archive Report

Link to release version v1.0.0 of GID the Planner

Technical Portion — README.md

28 March 2022 — Security Implementations

GID-the-Planner utilizes a "settings" page that enables users to edit their name and phone number if they wish to do so. A *Security Settings* tab was also included to allow users to enable two-factor authentication might they choose to do so. The user can also disable two-factor authentication. The two-factor authentication implements Meteor's *accounts-2fa package* and also involves the use of third-party 2FA applications such as *DUO* or *Google Authenticator*. To enable 2FA, the user must scan a QR code that is generated using a 2FA application. They will then be required to submit a code that the application provides them. Once submitted, 2FA for the user's account will be enabled, and the user is required to enter a code from the application every time they log in.

11 April 2022 — Fuzz Testing

John the Ripper was used to assisting with cracking the passwords for our default users on the application. This gave us a sense of how to improve our application's security. We made attempts to break into and crash our application by making efforts to access private URL's from another logged-in user, as well as testing meteor methods in the web applications console. These attempts all turned out to be unsuccessful thanks to the security components of the *Meteor* packages. We will indeed go forward with implementing the password check when a user creates an account to add a layer of security for application users. Our team has also decided to stop all further implementation of *MUI* due to the time constraints of this project. We will move forward with the primary UI framework being *Semantic UI React*.

Overall, our team is very satisfied with the production of GID-the-Planner. However, if we were given more time we would aim to implement more security features, such as implementing some sort of data encryption to prevent malicious actors from viewing data over a network. We would also aim to add more features for the users to interact with, such as a more polished user interface and the ability for users to edit any lists or tasks that they have created.

Installation

1. Install Meteor
2. Go to https://github.com/SafeG3Ar/gid-planner, click the "**Code**" button, and download "**gid-planner-master.zip**" as a ZIP file. Once downloaded, unzip in the local machine and open with JS IDE of choice.
3. In the terminal, `cd` into the `app/` directory of your local copy of the repo, and install third party libraries with:

```
meteor npm install
```

4. Lastly, you need to *install MongoDB Database Tools*, this will allow you to populate the vehicle database during startup.

Packages and Plug-Ins

After installing Meteor and running the program, exit the running app to install the following packages. In the root directory, cd into the app directory and use the following command line prompt to install the app:

Two-Factor Authentication

GID-Planner utilizes Two-Factor Authentication for security purposes. To install the packages for the 2FA, use the following prompts:

```
npm install buffer
meteor add accounts-2fa
meteor update --release 2.7-rc.4
```

Material UI

```
npm install @mui/material @emotion/react @emotion/styled
@mui/lab
```

Running the Application

Once the libraries are installed, you can run the application by invoking the "start" script in the package.json file:

```
meteor npm run start
```

The first time you run the app, it will create some default users and data. Here is an example of how the output may appear:

```
meteor npm run start

> meteor-application-template-react@ start
/Users/annacampainha/Desktop/GitHub/gid-planner/app
> meteor --no-release-check --exclude-archs web.browser.legacy,web.cordova
--settings ../config/settings.production.json

[[[[[ ~/Desktop/GitHub/gid-planner/app ]]]]]

=> Started proxy.
no postcss-load-configowser                  -
=> Started MongoDB.
I20220501-11:23:15.409(-10)? Creating the default user(s)
I20220501-11:23:15.424(-10)?   Creating user admin@gidplanner.com.
I20220501-11:23:15.631(-10)?   Creating user sue@gidplanner.com.
I20220501-11:23:15.697(-10)?   Creating user obi@gidplanner.com.
I20220501-11:23:15.770(-10)? Creating default data.
I20220501-11:23:15.770(-10)?   Adding: Basket (SooFlay)
I20220501-11:23:15.810(-10)?   Adding: Bicycle (ObiWan)
I20220501-11:23:15.811(-10)?   Adding: Banana (Adder)
I20220501-11:23:15.813(-10)?   Adding: Boogie Board (Adder)
I20220501-11:23:15.817(-10)? Creating default lists.
I20220501-11:23:15.817(-10)?   Adding: (SooFlay) list
I20220501-11:23:15.860(-10)?   Adding: (SooFlay) list
I20220501-11:23:15.864(-10)? Creating default profiles.
I20220501-11:23:15.864(-10)?   Adding:(Adder) profile
I20220501-11:23:15.902(-10)?   Adding:(SooFlay) profile
I20220501-11:23:15.904(-10)?   Adding:(ObiWan) profile
I20220501-11:23:15.906(-10)? Creating default tags.
I20220501-11:23:15.906(-10)?   Adding: (Important) tag
I20220501-11:23:15.944(-10)?   Adding: (Critical) tag
I20220501-11:23:15.947(-10)? Creating default tasks from
private/sample_task.json.
I20220501-11:23:16.214(-10)? Monti APM: completed instrumenting the app
=> Started your app.
=> App running at: http://localhost:3000/
```

Note Regarding "bcrypt warning":

You will also get the following message when you run this application:

```
Note: you are using a pure-JavaScript implementation of bcrypt.
While this implementation will work correctly, it is known to be
approximately three times slower than the native implementation.
In order to use the native implementation instead, run

  meteor npm install --save bcrypt

in the root directory of your application.
```

On some operating systems (particularly Windows), installing bcrypt is much more difficult than implied by the above message. Bcrypt is only used in Meteor for password checking, so the performance implications are negligible until your site has very high traffic. You can safely ignore this warning without any problems during the initial stages of development.

Viewing the Running Application

If all goes well, the template application will appear at http://localhost:3000. You can log in as:

- Test User: SooFlay (pw: giditdone);
- Admin User: Adder (pw: giditdone);
- Or, by creating your own account (Note: Because this application is still being developed, please use a fake email for your safety.)

ESLint

You can verify that the code obeys our coding standards by running ESLint over the code in the imports/ directory with:

```
meteor npm run lint
```

For details, please see

http://ics-software-engineering.github.io/meteor-application-template-react/

References

2012 Microsoft Corporation. (2012, May 12). *Appendix K: SDL privacy escalation response framework (sample)*. Microsoft. Retrieved April 30, 2022, from https://docs.microsoft.com/en-us/previous-versions/windows/desktop/cc307401(v=msdn.10)?redirectedfrom=MSDN

About ESLint. (n.d.). ESLint. Retrieved February 21, 2022, from https://eslint.org/docs/about/

Abramov, D., & Nabors, R. (2020, August 10). *React v17.0 Release Candidate: No New Features – React Blog*. React. Retrieved February 21, 2022, from https://reactjs.org/blog/2020/08/10/react-v17-rc.html

Appendix C: SDL Privacy Questionnaire. (2012, May 22). Microsoft Docs. Retrieved February 21, 2022, from https://docs.microsoft.com/en-us/previous-versions/windows/desktop/cc307393(v=msdn.10)?redirectedfrom=MSDN

Appendix M: SDL Privacy Bug Bar (Sample). (2012, May 22). Microsoft Docs. Retrieved February 21, 2022, from https://docs.microsoft.com/en-us/previous-versions/windows/desktop/cc307403(v=msdn.10)?redirectedfrom=MSDN

Appendix N: SDL Security Bug Bar (Sample). (2012, May 22). Microsoft Docs. Retrieved February 21, 2022, from https://docs.microsoft.com/en-us/previous-versions/windows/desktop/cc307404(v=msdn.10)?redirectedfrom=MSDN

Buckner, C. (n.d.). *Ethical Roadmap for Data Breach or Cyberattack*. Ethical roadmap for data breach or cyberattack. Retrieved May 1, 2022, from https://www.sdcba.org/index.cfm?pg=Ethics-in-Brief-2018-1203

Chandel, R. (2018, June 5). *Beginners Guide for john the ripper (part 1)*. Hacking Articles Raj Chandel's Blog. Retrieved April 11, 2022, from https://www.hackingarticles.in/beginner-guide-john-the-ripper-part-1/

Changelog | Meteor API Docs. (n.d.). Meteor Docs. Retrieved February 21, 2022, from https://docs.meteor.com/changelog.html#v2620220201

Coursera. (2022, April 29). *What is a security engineer? 2022 career guide*. Coursera. Retrieved April 30, 2022, from https://www.coursera.org/articles/what-is-a-security-engineer

Howard, M. (2006, June 6). Fending Off Future Attacks by Reducing Attack Surface. https://docs.microsoft.com/en-us/previous-versions/ms972812(v=msdn.10)?redirectedfrom=MSDN

Iroh - NPM Package Health Analysis. (2018). Snyk. Retrieved February 21, 2022, from https://snyk.io/advisor/npm-package/iroh

Maier, F. (2018a, January 18). *Iroh: Dynamic Code Analysis Tool - Exploit, Record and Analyze Running JavaScript*. GitHub. Retrieved February 21, 2022, from https://maierfelix.github.io/Iroh/

Maier, F. (2018b, January 18). *Maierfelix/Iroh*. GitHub. Retrieved March 27, 2022, from https://github.com/maierfelix/Iroh

Maier, F. (2018c, January 18). *Iroh/getting_started.MD at master · Maierfelix/Iroh*. GitHub. Retrieved March 27, 2022, from https://github.com/maierfelix/Iroh/blob/master/GETTING_STARTED.md

Material-UI/Core. (2021, July 30). NPM. Retrieved February 21, 2022, from https://www.npmjs.com/package/@material-ui/core

2022 Microsoft. (n.d.). *Devskim*. Visual Studio | Marketplace. Retrieved February 22, 2022, from
https://marketplace.visualstudio.com/items?itemName=MS-CST-E.vscode-devskim

Microsoft Security Development Lifecycle Practices. (n.d.). Microsoft. Retrieved February 21, 2022, from https://www.microsoft.com/en-us/securityengineering/sdl/practices

Migration from v4 to v5. (n.d.). MUI. Retrieved February 21, 2022, from https://mui.com/guides/migration-v4/

MongoDB Data Encryption. (2021). MongoDB. Retrieved February 21, 2022, from https://www.mongodb.com/basics/mongodb-encryption

n.p. (n.d.). *Passwords*. Meteor API Docs. Retrieved April 11, 2022, from https://docs.meteor.com/api/passwords.html

NPM-Audit. (n.d.). NPM Docs JS. Retrieved February 21, 2022, from https://docs.npmjs.com/cli/v8/commands/npm-audit

Openwall. (n.d.). *John the Ripper password cracker*. Openwall. Retrieved April 11, 2022,

from https://www.openwall.com/john/

Silva, Denilson. “NoSQL Injection in the accounts-2fa package | by Denilson Silva | Apr, 2022.” *Meteor Blog*, 1 April 2022, https://blog.meteor.com/nosql-injection-in-the-accounts-2fa-package-c06fcef43708. Accessed 2 May 2022.

Stocco, G. (2021, October 21). *Microsoft: DevSkim*. GitHub. Retrieved February 21, 2022, from https://github.com/Microsoft/DevSkim

Tassinari, O., Silbermann, S., Najdova, M., Dudak, M., Kunaporn, S., Leal, D., & Brookes, M. (2021, September 15). *Introducing MUI Core v5.0*. MUI. Retrieved February 21, 2022, from https://mui.com/blog/mui-core-v5/