Software Development Life Cycle Report

Bread Bank UH

ICS 427, Spring 2023

Raphael Bumanlag

Robin Dumlao

Christian Reasoner

Yhanessa Anne Sales

Table of Contents

Release

I. Incident Response Plan
II. Final Security Review
III. Certified Release & Archive Report

Introduction

I. Team Name

- BreadBankUH

II. Members

- Raphael Bumanlag
- Robin Dumlao
- Christian Reasoner
- Yhanessa Anne Sales

III. Application Title, Description, and Functional Requirements Specification

- BreadBankUH
- A simple mock banking application using fictitious currency that allows users to access their bank account information and perform various banking transactions through a web browser. This replication of a functioning banking web app shall create the groundwork experience toward creating a secure application in a real world environment.
    - This can include checking account balances, viewing transaction history, transferring funds, paying bills, and more.
    - The application is secured with a login and password to protect against unauthorized access.
    - Once logged in, users may view and interact with our bank interface as necessary.
    - New users to the application will be prompted to create an account or login using a preexisting account.

IV. Type of Program

- Web Application

V. Development Tools

- Language: Javascript
- IDE: IntelliJ, Visual Studio Code

Requirements

I. Security and Privacy Requirements

- **Logging and Auditing**: The application should implement security monitoring, admin permissions, and logging to detect any suspicious activity and investigate any security incidents.
- **Authentication**: The application should require a form of authentication, possibly multiple forms of authentication if possible such as 2FA.
- **Incident Response**: Having a plan in place to respond to security incidents and data breaches that includes communication with customers and regulatory bodies.
- **Access control**: Limiting the actions that users can perform within the application based on their role and permissions.

Authentication is important for banking applications because it ensures that only authorized users are able to access sensitive information and perform actions that pertain to their account within the application. Access control allows the bank to limit what actions authorized users can perform, and what information they can access. This also includes system admins, which play a crucial role in maintaining the security and stability of our banking application. They are responsible for installing and configuring the necessary software and hardware, monitoring the system's performance, and troubleshooting any issues that may arise. They also ensure that the system is in compliance with regulatory requirements and industry best practices. Additionally, they may be responsible for maintaining backups and disaster recovery plans, as well as managing user accounts and permissions.

Overall, system administrators help ensure that the banking application is reliable, secure, and available to users at all times. Logging allows the bank to track user activity and detect any suspicious behavior. It also allows the user to have a sense of organization and clarity with their transactions. Incident response allows the bank to quickly and effectively respond to security incidents, such as data breaches, in order to minimize damage and protect sensitive information. Some sort of messaging or alert system for our software is a great way to interact with the user and keep them involved whenever there is a security/privacy issue.

II. Quality Gates/Bug Bars

In a banking program, security and privacy are typically implemented through a combination of technical, physical and administrative controls. Some examples of these controls are:

- **Technical controls**: encryption of sensitive data, firewalls, intrusion detection and prevention systems, access control systems, etc.
- **Physical controls**: security cameras, security personnel, secure data centers, etc.
- **Administrative controls**: policies and procedures for managing access to sensitive data, incident response plans, security awareness training for employees, etc.

Banking applications handle a variety of sensitive information, including personal identification information (such as names, addresses, and Social Security numbers), account information (such as account numbers, balances, and transaction history), and financial information (such as credit card numbers, loan information, and investment data). Additionally, banking apps may also handle sensitive information related to security and fraud detection, such as login information and biometric data. Due to the sensitive nature of this information some of the most important security and privacy levels to include relate to the following:

- **Notice and Consent**: Notice and consent refers to the process of informing users about the types of data that the application collects and how that data will be used, and obtaining their explicit consent for the collection and use of that

data. This is an important aspect of data protection and privacy, as it allows users to make informed decisions about whether to use the application and how to use it.

    - By extension, the user should also be notified when there is an interaction done between the web application and their account, and are notified of the policies that the web app will go to in order to contact the user should anything happen to their information. This will prevent false entities masquerading as the web app to trick users into offering their personal information for malicious purposes.

- **User Controls**: User controls are features that allow users to manage their own data and privacy settings within a banking application. For example, a user may be able to control what types of notifications they receive, or they may be able to specify which data elements they want to share with the bank. User controls are important because they give users more control over their data and help to ensure that they are comfortable with the way the application is using their data.
- **Data Protection**: Data protection refers to the measures that are taken to safeguard sensitive data, such as personal information and financial transactions, from unauthorized access, use, or disclosure. This includes technical controls such as encryption and firewalls, as well as physical and administrative controls. Data protection is important in a banking application because the loss or unauthorized disclosure of sensitive data can have serious consequences for both the bank and its customers.
    - Data protection should also ensure that the interactions between the web application and to the user are secure, and that through these security measures, it prevents vulnerabilities in tampering and spoofing that may be attempted.

*It's also important to comply with legal and regulatory requirements such as PCI-DSS for credit card information, and GDPR for personal data protection.

III. Risk Assessment Plan for Security and Privacy

We will utilize Microsoft's SDL Privacy Questionnaire as a general guideline to assess our app for security and privacy.

The date for when the public will first have access to our web application BreadBankUH is May 2nd, 2023. This is a tentative date and is subject to change. The system security and privacy plan is not a one person job as it requires careful review and analysis of the technical infrastructure, data, testing, and architecture of the web application. Therefore, every member of the team is responsible for developing and implementing security and privacy aspects into our banking application.

By having everyone be held liable for the application's privacy, it will help reduce the number of careless software mistakes at an acceptable level. In this context, acceptable level is defined as the possible threats not influencing the user's security and privacy experience on the application. Risks will be continuously assessed throughout the software development life cycle.

BreadBankUH is categorized as a P3 on the Privacy Impact Rating Scale. The application will NOT exhibit any of the following behavior:

- Store personal identifiable information (PII) on the user's computer or transfer it from the user's computer
- Provide an experience that targets children
- Continuously monitor the user
- Install new software or change file type associations, home page, or search page
- Transfer anonymous data

Users will have to conform to the Privacy Notice and Consent agreement, which notes the risks involved upon registration. Refer to the Quality Gates section for more details. An element of the application that is crucial for security and privacy review is the user's information and transaction history as well as user login credentials. BreadBankUH recognizes the sensitivity of these informations and will not disclose them to unauthorized parties.

Design

I. Design Requirements

Design Goals: The design of the web app is intended to reflect its promise toward delivering a user friendly application that is not only convenient to use, but also secure. With this vision in mind, the main priorities focus to ensure that both the user data is secure, and that the service application is secure while the user is interacting with the program.

Protection and Security: As the web app continues to develop, it is our responsibility and drive to build a foundation for a secure and well protected database. This shall start with a simple password protection for each user, which establishes the lowest means of necessary security. Once that stage has been achieved, then the next step is to create a user personalized security question that they may choose to answer, as well as reconfirmation from the user in order to access their account should it have been breached.

In terms of user data security, our goal is to be able to establish the ability to create a 2 Factor authentication (2FA) that will create the final layer of security for users, as they will need to provide the sufficient information that the web app will supply the user with. Thus, there will be varying levels of user authentication that they may choose from, giving them the options as well as recommending users to bolster their user protection.

Auditing and Surveillance: For privacy purposes, there should be no case in which the user should transfer personal data (PII) without them being first notified and confirmed. While a user is currently using the web application, if the program detects another instance of the same user, then it should shut out the previous instance in order to prevent multiple instances at a time. Any irregularities within the program shall send notifications towards the administrators, as well as any users that may have been

involved should a breach occur. Each interaction that is done should create a notification to the user as a confirmation that the corresponding action has been confirmed by the system. The large amount of notifications may seem to the user as overbearing, but it is to ensure that every recorded action on our website is apparent, to help create a secure environment.

II. Attack Surface Analysis and Reduction

There will be 3 levels of privileges for users.

- **Low**: The only permissions this user will have is the ability to view the landing page of the application, which will mainly consist of login and register buttons. This low level privilege is assumed that the user is not a customer of BreadBankUH as they will not have further access to any other features of the application.
- **Medium**: Existing customers will have the ability to access their bank accounts with their login credentials to check their account balances, view transaction history, transfer funds, and pay bills. Other privileges include the ability to change their nonsensitive PII as well as password. New customers will have the same privileges as existing customers. However, it is necessary for them to register and create an account to get access to these features.
- **High**: System administrators will have the highest level of privileges for the application. They will have the ability to manage all existing users accounts and make changes to them if needed and/or requested by the customer

Calculating the relative attack surface will be based off of Microsoft's attack surface analysis approach. One of the major security concerns of the web application is unauthorized access to a user's account and information as it is the main point of entry through forms and fields, if 2FA is not enabled.

Like most bank web applications, user login credentials are appealing to hackers because it leads them access to sensitive information. A malicious user may attempt to

exploit the code when developers fail to implement security features and maintain them. Session fixation is one of the top web app application vulnerabilities and it is no exception that BreadBankUH is susceptible to this problem. This is most common in applications that utilize cookie-based sessions. A way to combat this is to create a new session ID each time a user logs in. If user is idle for 10 minutes or more, the application will automatically log them out. List of other vulnerabilities include but not limited to:

- Unencrypted passwords
    - This problem will arise if passwords are not kept secured and protected
- SQL injection
    - Input made by the user can be manipulated if the attacker overloads the database.
- Security Misconfiguration
    - The lack of properly implementing technical security features.

III. Threat Modeling

Identifying Threats:

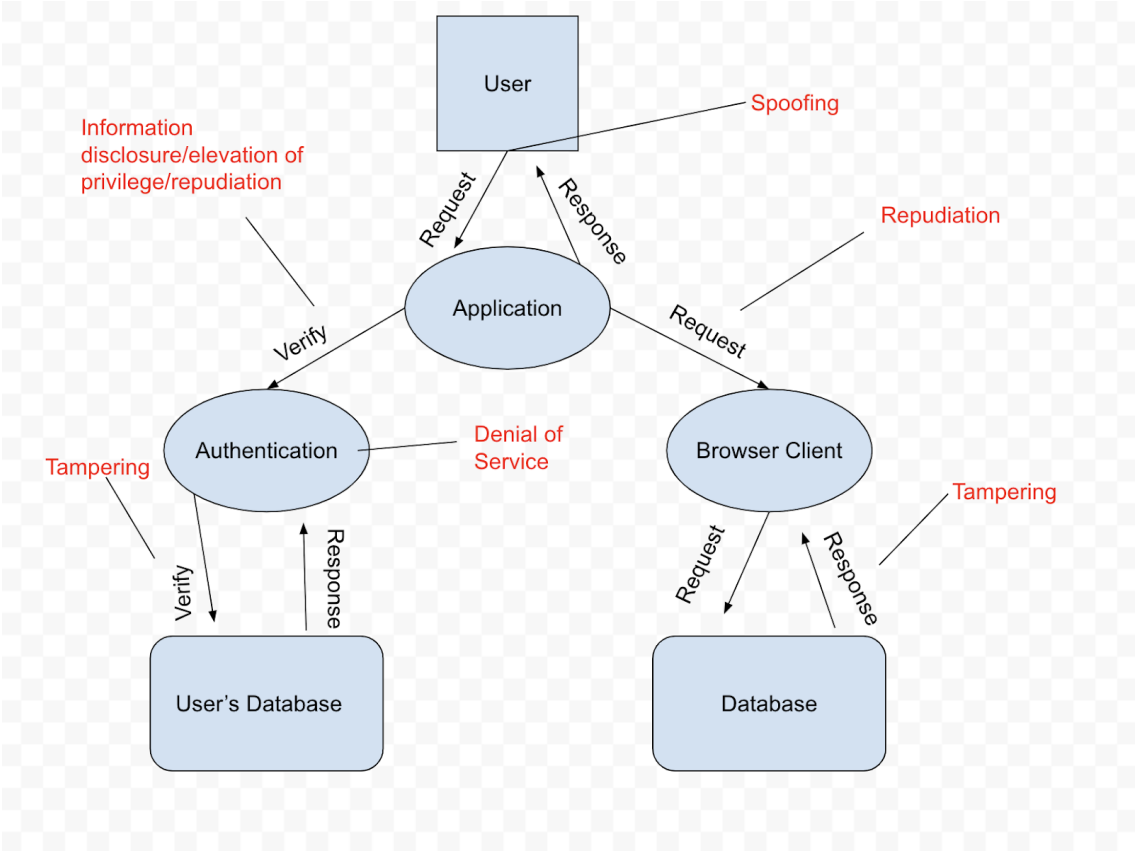
- Authorized user
    - Denying to adhere with the notice and consent agreement
    - Failure to login on a secured device
    - Stolen device
- Attacker
    - Bypassing login authentication

Categorizing Threats (STRIDE):

- **S**poofing
    - Spoofing as impersonated user or admin
- **T**ampering
    - Manipulating data in database
    - Changing passwords
- **R**epudiation:

    - Denying service and assistance to users who are suspected as threats
- **I**nformation disclosure
    - User data collection without permission
- **D**enial of service
    - DoS attack
    - Application crash
- **E**levation of privilege
    - Impersonation of inappropriate user role
    - Security misconfigurations

Diagram:



Implementation

I. Approved Tools

Languages will be Javascript, HTML, and CSS. Below is a table that highlights the essential tools to be used by developers in the BreadBankUH application. The table includes basic details such as each tool's name, version, and brief description. Additionally, it also includes the website for each tool, which serves as a quick reference to streamline the process of obtaining more information about the technologies used in the application's development.

NAME	VERSION	DESCRIPTION	Website
IntelliJ IDEA	2022.3.2	IDE	https://www.jetbrains.com/idea/
Meteor	2.9	JavaScript Web Framework	https://www.meteor.com/
Node.js	^14	Server side Javascript Environment	https://nodejs.org/en/
React	^18.2.0	Javascript Library for User Interface	https://reactjs.org/
React Bootstrap 5	^2.7.2	CSS Framework	https://react-bootstrap.github.io/
MongoDB		Database	https://www.mongodb.com/
ESLint	^8.34.0	Static Analysis Tool	https://eslint.org/
Windows	10/11	Operating System	https://www.microsoft.com/en-us/windows/windows-10-specifications https://www.microsoft.com/en-us/windows/windows-11-specifications
MacOS	Big Sur v11.3	Operating System	

Git/GitHub	^2.3(Git)	Version Control/ Collaboration	

While the table above lists the main tools used in the BreadBankUH application, it is important to remember that there are several other dependencies that are necessary for the application to function properly. The tools used in the BreadBankUH application are included in the package.json file, which is located at BreadBankUH/app/package.json. This file provides a centralized location where developers can view metadata, manage, and update these tools as needed.

This tech stack is based on a template provided by the University of Hawai'i at Manoa Information & Computer Sciences Department. Link for more information: https://github.com/ics-software-engineering/meteor-application-template-react

II. Deprecated/Unsafe Functions

Meteor:

- Unsafe: Meteor.subscribe() without a subscription handle. When a Meteor subscription is created without a handle, it is difficult to manage its lifecycle, which can lead to issues like memory leaks, unresponsive UI, and data inconsistency.
- Alternative: Meteor.subscribe() with a subscription handle to manage the lifecycle of the subscription.

Node.js:

- Unsafe: fs.appendFile(), fs.writeFile() without specifying file permissions. If file permissions are not specified when writing to a file, it can lead to security vulnerabilities and unauthorized access to sensitive data.
- Alternative: fs.appendFile(), fs.writeFile() with specific file permissions using the mode option.

React:

- Deprecated: componentWillReceiveProps()
- Alternative: getDerivedStateFromProps().

MongoDB:

- Unsafe: db.eval() allows arbitrary JavaScript code to be executed on the server, which can lead to security vulnerabilities and unintended data access.
- Alternative: Use server-side scripts with caution and avoid the use of eval().

III. Static Analysis

For the purposes of this project, due to the recent familiarity of using JavaScript tools and techniques through integration with the Intellij IDE, our team shall utilize ESLint during the development process in order to create a productive environment where the debugging process makes for ease of access. ESLint also offers the flexibility of configuring rules to allow us to enforce rules that cater toward the project's needs. Through the use of ESLint, we aim our focus toward the proper coding styles in ways that look and feel intuitive.

ESLint is a JavaScript linter that can be configurable towards our needs. From style issues for clarity of code to debugging issues that it may catch before the actual runtime of the program, ESLint will help to ensure that our web application process is secure and efficient, rather than spending unnecessary amounts of time in the debugging process. Throughout the development process, our team will document the procedures as the web application continues to develop while enforcing proper coding standards. Documentation of ESLint can be found here: https://eslint.org/docs/latest/use/core-concepts

Initial Instance - February 18, 2023

- Created Github repository of the first BreadBankUH instance.
- Setup readme.md file with our team members with respective tasks
- Setup ESLint analysis tool to BreadBankUH, however, no mockup pages yet initialized, so testing is minimal currently.

Verification

I. Dynamic Analysis

After conducting some research and reading several reviews of various dynamic analysis tools, our team has decided to use Iroh.js. It is a dynamic analysis tool for JavaScript that helps optimize our web application's code for better performance. Several claims from reviews captured our attention and led us to choose Iroh.js.

- One of the main benefits is its user-friendly interface, which enables developers to write code with ease and analyze it to make improvements during the development process. This also helps with debugging.
- Another advantage is the ability to modify code during runtime, allowing us to see what is happening in real-time and test out what works and what doesn't.
- Additionally, Iroh.js is configurable and customizable to fit the specific needs of our web application.

Based on our experience using Iroh.js as of March 28, 2023 thus far, we have encountered no issues and found that it lives up to its expectations. The tool has fulfilled all of our desired functionalities with efficiency.

Our team is satisfied with Iroh.js's capabilities, which have seamlessly integrated into our development process, resulting in a smooth workflow. We can approve of the the tool's reliability and effectiveness in optimizing our web application's performance. We will continue to review the tool during the duration of development and update our comments throughout the report.

Further documentation of this tool can be found here:

- Github: https://github.com/maierfelix/Iroh
- Website: https://maierfelix.github.io/Iroh/

II. Attack Surface Review

This table is from the previous section of approve tools with a few changes

NAME	VERSION	DESCRIPTION	Website
IntelliJ IDEA	~~2022.3.2~~ →2022.3.3	IDE	https://www.jetbrains.com/idea/
Meteor	2.9	JavaScript Web Framework	https://www.meteor.com/
Node.js	^14	Server side Javascript Environment	https://nodejs.org/en/
React	^18.2.0	Javascript Library for User Interface	https://reactjs.org/
React Bootstrap 5	^2.7.2	CSS Framework	https://react-bootstrap.github.io/
MongoDB		Database	https://www.mongodb.com/
ESLint	^8.34.0	Static Analysis Tool	https://eslint.org/
Iroh.js	0.3.0	Dynamic Analysis Tool	https://maierfelix.github.io/Iroh/
Windows	10/11	Operating System	https://www.microsoft.com/en-us/windows/windows-10-specifications https://www.microsoft.com/en-us/windows/windows-11-specifications
MacOS	Big Sur v11.3	Operating System	
Git/GitHub	^2.3(Git)	Version Control/ Collaboration	

Although most of our tools have remained unchanged, we recently received a new release of the IntelliJ IDEA version, which included several patch notes highlighting new features and bug fixes. As part of our commitment to staying ahead and keep our product

protected, we will periodically update our tools to ensure that they remain up-to-date and provide the most secure and optimal environment for our web application.

We understand the importance of maintaining our technology stack to keep up with the evolving technologies and tools and to provide our users with the best possible experience. As such, we remain careful in our efforts to stay up to date of the latest updates in software tools and technologies.

Verification II

I. Fuzz Testing

As a part of our effort to ensure security and reliability of our web application, we attempted to break or hack into our program using various techniques.

For our first attempt, we conducted a fuzz testing process for logging in our online banking system with email and password. The goal of this process was to identify any potential weaknesses with our login system. Upon registration, users are required to follow a few password requirements when creating one. Passwords must contain at least one numeric character, at least one special character, a capital letter, and must be at characters long. This enhances security and makes it more difficult for hackers to guess or crack passwords. This also helps with preventing any type of unauthorized access to user accounts and sensitive information. We also made sure to implement form validations so that the password requirements are enforced. We tested this by logging in with correct credentials and random incorrect credentials in an attempt to uncover any defects. Fortunately, this fuzz testing was successful and we feel confident that our web application will demonstrate some kind of trust with our users.

Our second attempt at fuzz testing was unsuccessful due to a huge significant security flaw that was found in our source code. Upon reviewing the code, it was discovered that the source code contained the default account login credentials that was easily accessible in one

of the files. This would allow attackers to access the admin account, which has access to the information of all registered users. We recognize that this is a concern to the importance of secure coding practices to ensure that sensitive information is private to unauthorized parties. We will be taking steps to make sure that this information is not exposed in the code and will implement extra security measures to protect our users' accounts and data.

Our third attempt at fuzz testing was black box fuzzing. This involved having testers to test our web application. They have no knowledge of the internal workings of the app and have no access to the code. Their goal was to break into the system by attempting to login as many times as they want with the given valid email addresses we have provided . This was a success as they were not able to have access to the system. After this testing, the team have been contemplating if we should implement a feature where the user will only have a certain amount of login attempts before they are locked out of the account if they input the wrong password.

II. Static Analysis Review

During the development and continued creation of this project, ESLint has proved to be a useful tool to create a smooth experience for coding and debugging. With the assistance of ESLint, it has allowed us to focus on the development process, as the proper coding standards are continuously being met as code is being written. This has created opportunities to create initial code that follows coding standards while maintaining clarity throughout the application. Compared to a nonstatic tool assisted coding experience, many of the small syntax errors or other equivalent complexities have not been a problem as they are found immediately when occurred rather than found during the end of compilation.

ESLint has helped to incentivize proper coding that also increased the efficacy of the development process. Currently, there has been no concerns with the use of ESLint, there have been no troubles since the initial setup of the JavaScript linter, with the only caveat being small coding style problems such as indentation or line spacing. However, these small

incidents are manageable and help to enforce the same coding style for each of the contributors in the coding development process.

III. Dynamic Analysis Review

In contrast to a static analysis tool such as ESLint, which is a JavaScript linter that checks over the program code prior to runtime, Iroh allows us to survey runtime data of the code during execution. Iroh has been able to collect the data of how the program behaves at runtime and be able manipulate the code in realtime. There are no new updates about our experiences with this tool. It remains the same as the previous section.

Release //4-6 pages

I. Incident Response Plan

In the event of software threats, in the future, the UH Bread Bank development plans to take proactive steps in combating such threats by implementing comprehensive strategies. This includes vulnerability assessments, extensive testing, and software update to ensure the system is up to date and secure. This is to ensure that the customers and users feel safe and confident in the situation.

Furthermore, UH Bread Bank will also take steps to educate its team and users on best practices for cybersecurity, creating secure passwords/answers, avoiding phishing, and using two-factor authentication. Additionally, the development team will be willing to collaborate with industry experts to stay informed on the emerging threats and implement appropriate security measures to mitigate them.

Due to UH Bread Bank being a small development team, it is important to maintain an effective privacy escalation process to ensure that our users' data is protected. In order to do this, we have determined that it is necessary for each of our 4 members to take on a role in the privacy escalation team. Though this is pretty challenging to hold a role besides developers, we feel that it is necessary to maintain a trusted level of security for our users' information. We believe that this approach will allow the team to effectively manage privacy and security issues and uphold our commitment to safeguard users' data.

Privacy Escalation Team:

Team Member	**Role**	**Description**
Raphael Bumanlag	Escalation Manager	This role will be responsible for handling the team's response to any threats and ensure that all necessary steps are taken to address the issue. They will also be responsible for being the primary point of contact for all the team members and will oversee the process.
Robin Dumlao	Legal Representative	This person will handle all legal matters related to any possible security breaches and/or incidents concerning the event. This also includes cooperation with law enforcement should the occasion present itself.
Christian Reasoner	Public Relations Representative	Manage external communication during and after an incident. The PR representative is

		responsible for developing and executing a communication plan that minimizes negative impact to the company's reputation, restores customer trust, and ensures regulatory/legal compliance.
Yhanessa Anne Sales	Security Engineer	Security Engineer will handle the technical part of the breach by analyzing and determining the potential causes. They will also be responsible for implementing enhanced security measures to prevent similar incidents in the future.

Contact Email: uhbreadbank@email.com

This email address serves as a way for users to contact to report anything related to security or privacy breach. This email address will be monitored by the escalation managers, who will quickly assess the situation and take appropriate action.

Procedures in the event of an incident:

1. Incident Identification: Identify the incident, its severity, and its potential impact on the application. Monitor any of the logs or customer feedback to detect any unusual behavior or patterns.
2. Incident Triage: After identifying the incident, triage it by categorizing its severity and impact on your application. This will help to prioritize the incident and allocate resources based on its severity. Depending on whether the severity is low or extreme will determine the incident resolution and the steps necessary to the resolution.
3. Incident Communication: Notify all users/stakeholders involved in the incident, including the management team, development team, support staff, and customers. Clearly communicate the incident, its severity, and the estimated time to resolution.
4. Incident Analysis: Perform a thorough analysis of the incident to identify the root cause, the impacted areas, and other related issues. Gather as much data as possible from logs, metrics, and customer feedback.
5. Incident Resolution: Implement a solution to resolve the incident and restore normal application functionality. This may involve rolling back changes, patching vulnerabilities, or making quick fixes.
6. Incident Follow-Up: Perform a post-incident analysis to assess the effectiveness of your response and identify any areas for improvement. Review your incident response plan and make necessary changes to prevent similar incidents from occurring in the future or to further enhance the response plan.
7. Incident Reporting: Document the incident, including its severity, impact, root cause, resolution, and any follow-up actions taken. Make sure to include as much detail possible as it will help with future incident responses. Use this information to update your incident response plan and report to the management team on the incident's impact and the effectiveness of the

response.

II. Final Security Review

For the purpose of the final security review, UH Bread Bank shall be re-evaluated utilizing the previous reviews of static and dynamic analysis, along with the threat model to create a comprehensive final check before release.

Final Threat Modeling Review

Based on the threat model designed for this application, there are authorized users in which the system is able to be utilized to avoid unauthorized access while reassuring secure user logins. It is paramount that UH Bread Bank holds these possible threats as a priority, and therefore shall continue to monitor the threat model throughout.

In terms of STRIDE, prioritizing the use of authorized user logins and surveillance of any possible threats by means of unauthorized access shall reduce the possibility of threats in the future. The occasional prompt for the user to reauthorize their login will allow for the continual reassurance of users. For the purposes of supposed information tampering or spoofing, the verification of user information and login authorization shall aim to reduce the problem of such an occasion to happen.

Final Static Analysis Review

Throughout the development of UH Bread Bank, in cooperation with ESLint, the coding environment has been allowed to concentrate our team's focus on the development of the application rather than concerns about syntax and other possible static analysis hindrances. In the final reviews surrounding our static analysis tool, ESLint has created a smooth transition for several developers to collaborate and cooperate to create UH Bread Bank.

Due to the constant presence of ESLint as a development team, it created a similar coding environment for each of our developers to operate in, causing a uniform experience with less worries on the continual cycle of code creation and re-understanding of created code at a later stage. This has created minimal problems in the code creation process, and even till its end of the development stage there has been little to no problems with improper coding standards.

Final Dynamic Analysis Review

Through the use of Iroh, it allowed us during the development process of UH Bread Bank to be able to monitor and survey how the application operates during runtime. The ability to change data in real-time as the application ran aided in concentrating on certain aspects of the code without having to recompile the program and test again.

In comparison to ESLint, Iroh JS was able to take advantage of its runtime monitoring ability to allow us as a development team to observe how different types of data and information on the web application interact with each other during code execution, as well as to see if things worked as intended.

Quality Gates and Bug Bars

When it comes to the development process of UH Bread Bank, our team has established several quality gates and bug bars to help create several developmental milestones with a general bar set to maintain a level of consistency and competency for the code throughout each iteration.

Overall Grading

In terms of the final reviews in the perspective of the Functional Specification, the team has decided that it has a grade of “Passed FSR with exceptions.” This means that the program has met most other FST requirements and demonstrated expected

functionality. However, there were some minor identified issues during the testing that can be addressed before the program can be compliant with the FSR.

III. Certified Release & Archive Report

Link to the release version of the program:
https://github.com/BreadBankUH/BreadBankUH/releases

UH Bread Bank is a mock banking application that aims to detail the security measures that encompasses cybersecurity development cycles. Through the use of user information and fictitious currency, it allows users to interact with their personal data in a safe and secure manner. As the general outlook of a banking system exemplifies the necessity of having proper security measures, it pushed our team to lean into the cybersecurity development cycle as a scope for the project to ensure it may hold up in future iterations.

Through the various possible transactions available through the web application, it may replicate that of a functional banking app, creating the initial hallmark toward our team's development experience toward aiding our efforts to contribute to real world applications in the foreseeable future. With this possible learning experience, UH Bread Bank can be taken from in different aspects in order to recall and enforce proper coding practices along with creating a product with cybersecurity as the focal point.

Additional features we would have liked to implement is the two factor authentication if given the opportunity to do so. However due to time constraints and other commitments, this feature was unable to come to fruition in the way that we envisioned.

Technical Notes/User Installation:

Online Repository Download Link: https://github.com/BreadBankUH/BreadBankUH

1. Install a IDE of your choice (VSCode, IntelliJ, etc.)
2. Make sure you have the latest version of Node.js (https://nodejs.org/en)
3. Install Meteor JS (https://docs.meteor.com/install.html)

4. Clone the UH Bread Bank source code from GitHub (https://github.com/BreadBankUH/BreadBankUH)
5. Open the project in your desired IDE
6. In the terminal cd into app: cd app
7. Type following command: meteor npm install
8. Then type following command: meteor npm run start
9. The application should run on localhost:3000
10. To uninstall, simply delete the file from your computer files. Or to stop running the program press, ctrl + c