Mai Abe
Julie Chai
Jun Okabe
ICS 491

Assignment 5: Release

A. Introduction

1. Team Name : Team JMJ
2. Team Members : Julie Chai, Mai Abe, Jun Okabe
3. Application Name : EarnIt
    - EarnIt is a web application that keeps track of the user's monthly income, track and categorize expenses, and allows the user to set a goal of how much they want to save each month.
4. Type of Program :
    - A Web Application built using Meteor.js, a full-stack JavaScript framework.
5. Function Requirement Specifications :
    - Application will require a sign-up feature (with a privacy agreement step).
    - Application will require a user login feature.
    - Application will require an application overview page.
    - Application will allow the user to input his/her monthly income.
    - Monthly income will be added to the savings amount for the current month.
    - Application will allow user to input expenses.
    - Each expense will be deducted from the savings amount for the current month.
    - Application will allow the user to set an amount she/he wants to save each month.
    - Application will allow user to view current savings.
6. Development Tools :
    - Meteor (Javascript Application Development Platform)
    - MongoDB (NoSQL Database to store application information)
    - IntelliJ IDEA (Integrated Development Environment)
    - ESLint (A static analysis tool used to improve the code quality)

- Node Package Manager (Install and Update JavaScript packages that will be used in the project)
- GitHub (for version control, source code organization, and project task organization)

B. Requirements

1. <u>Security and Privacy Requirements</u>

- System shall require passwords in order to access to the account. A secure authentication mechanism must be placed to ensure only authorized users to be able to login.
  - passwords should not be transmitted and stored in plain text (Hash passwords)
  - passwords must meet the minimum set requirements to enforce password strength
    - At least 6 characters
    - At least 1 number
- System shall notify the user that user information, such as username, password (hashed), and financial information (income, expenses, savings goal) will be collected and saved in the database. The user must agree to a privacy agreement when they wish to use the application services.
- System shall lock the account when there is 3 bad attempts to log in. This security feature will prevent or reduce the efficiency of malicious individuals who plan to execute brute-force attacks.
  - The account must require 5 minutes of no attempt in order to try again.
- System shall logout automatically when session is timed out. This security feature will prevent any possible tampering (For example, an authorized user uses the application and forgets to logout on a public computer).
  - The session timeout shall occur when there is no activity for 30 minutes.
- System will send a link to reset the password to a user who forgot it. This feature will help securely re-establish the user's ability to access the application.

- The data recorded from the user's prior activity shall only be available to the authorized user (administrators). A security feature must be implemented to prevent privilege elevations for regular users. Also, sanitization of user input must be implemented to prevent any user from possibly accessing the database (For example, NoSQL injections).
    - The data includes monthly income, expenses, and savings record.

Keeping track of project tasks and issues

We will keep track of the project tasks, issues, and possible security flaws using GitHub project board feature. The project board is a feature on GitHub that helps us organize project tasks and issues. Using a project board, we will be able to create our own workflows.

We will organize the project based on the web pages the application will require. In each page-based project, we will have "To Do", "In Progress", and "Done" columns. In the "To Do" column, a list of tasks and issues will be placed. In the "In Progress" column, tasks and issues will be assigned to a team member to indicate the current tasks that are being completed. In the "Done" column, completed tasks and issues are to be placed.

2. Quality Gates/Bug Bars

Table 1: End-User Privacy Bug Bar

Severity Level	Description
Critical	• Lack of notice and consent for sensitive personally identifiable information (PII) of user. • Lack of user controls ○ The user having no ability within the UI to stop collection of non-sensitive PII. • Lack of data protection ○ PII that is stored in the database, is not stored with a proper authentication method, which allows non-authenticated users to access and manipulate the data. • Lack of internal data management and control ○ People who have no business needs for the information have access to stored PII.

Important	● Lack of notice and consent for non-sensitive PII of user. ● Lack of data protection ○ PII that is stored in the database, is not stored with a proper authentication method, which allows non-authenticated users to access data.
Moderate	● Lack of user controls ○ The user having no ability within the UI to remove PII data in the form of hidden metadata (in a file) and may be accessible to others. ● Lack of data protection ○ Temporary non-sensitive PII that is stored allows non-authenticated users to access data. ● Lack of internal data management ○ Stored data does not have any retention policy
Low	● Lack of notice and consent for PII of user (stored in a file).

Table 2: Enterprise Administration Privacy Bug Bar

Severity Level	Description
Critical	● Lack of enterprise controls ○ Data transfer of sensitive PII without notification or consent from the enterprise administration
Important	● Insufficient privacy disclosure ○ Disclosure to enterprise administrators (for example, deployment guides) does not disclose storage or transfer of PII
Moderate	● Lack of enterprise controls ○ There is no method to help enterprise administrators prevent accidental disclosure of user data.
Low	● Lack of enterprise controls

Table 3: Server Security Bug Bar

Severity Level	Description
Critical	● Privilege Elevation ○ The ability of a non-authenticated remote user to execute arbitrary code or obtain higher-level privilege for unauthorized access to files and data.
Important	● Privilege Elevation ○ The ability of an authenticated remote/local user to execute arbitrary code or obtain higher-level privilege for unauthorized access to files and data. ● Information Disclosure ○ Ability to locate and access information that was not designed to be exposed. ● Tampering ○ Permanent modification of data/files that persist after restart. ● Spoofing ○ Ability for a remote user to impersonate a valid user (of their choice).
Moderate	● Information Disclosure ○ Ability to access information (from a known location) that was not designed to be exposed. ● Tampering ○ Temporary modification of data/files that do not persist after restart. ● Spoofing ○ Ability for a remote user to impersonate a random valid user.
Low	● Information Disclosure ○ Random and unintended access to information that was not designed to be exposed.

Table 4: Client Security Bug Bar

Severity Level	**Description**
Critical	● Privilege Elevation ○ Ability of a remote user to execute arbitrary code or obtain a higher-level privilege than intended, such as for unauthorized access to files and data.
Important	● Privilege Elevation ○ Ability of a remote user to execute arbitrary code with additional user actions. ○ A user with low-level privilege can obtain a higher privilege. ● Information Disclosure ○ Ability to locate and access information (anywhere) that was not designed to be exposed. ● Tampering ○ Ability to change user data permanently. ● Spoofing ○ Ability to impersonate the UI of a specific page that the users rely on to make decisions (make valid trust decisions). ● Security Features ○ Ability to ignore or avoid any security features that were implemented in the application.
Moderate	● Information Disclosure ○ Ability to access information (from a known location) that was not designed to be exposed. ● Spoofing ○ Ability to impersonate the UI of a specific page that the user is familiar with.
Low	● Information Disclosure ○ Random and unintended access to information that was not designed to be exposed. ● Tampering ○ Ability to change user data and files temporary (will not persist after restart of web application).

3. Risk Assessment Plan for Security and Privacy

Security risk assessments allows us to identify parts of our web application that requires threat modeling and security reviews. Our web application, EarnIt, is a personal financial organizer, which will store and transfer personal identifiable information (PII), such as usernames, passwords, and financial information, to a MongoDB database. Thus, it is essential to use threat modeling and extensive security reviews on features that involve user signup, user login, and financial information input/output. Thus, these security risk assessments will help strengthen the security of the application by minimizing any vulnerabilities (brute-force attacks, data tampering, and elevation in privileges) that are present in the application.

Privacy risk assessments allows us to determine Privacy Impact Rating (P1, P2, and P3), which is a degree of risk from a privacy standpoint. Since our application will store PII in the database (username, password, and financial transactions), our application will be considered as a P1, which indicates "High Privacy Risk". Therefore, the user must agree to a privacy agreement during "sign-up", which will notify new users that application will store PII in the database. Any unauthorized access will be prevented using a user authentication API provided by Meteor. Regular users will only be able to access their own data and administrators (those involved in creating the application) will be able to access all data and accounts.

C. Design

1. Design Requirements

- Application Main Page :
  - Must be clear, understandable, and professional-looking for the users. The public page that does not require a user login must have the guided tour page that explains about the use of the application and its purpose.
- User Signup feature :
  - If a username already exists, the user will be notified with a message that they will not be able to proceed.

    - User will not be able to proceed if the entered password does not meet the minimum requirement.
        - If the password does not fulfill the password requirements (at least 6 characters and 1 number), then the user will be shown a message to inform them of what needs to be fulfilled.
    - User will be given an option to login if he/she has an account already.
    - In order to create an account, the user will need to agree to a privacy agreement.
- User Authentication / Login feature :
    - User will receive an alert message when login was not successful. The alert message shall tell the user that he/she :
        - entered the wrong password or
        - the user does not exist (because the user has not signed up for an account yet)
    - User will be shown a message that he/she must try again after 5 minutes when 3 unsuccessful attempts were made.
    - User will be given an option to sign in if he/she does not have an account yet.
    - User will be given an option to send a temporary password to the user email address if the user forgets his/her password.
- User main page :
    - Must notify the user whether the user was able to save the amount they set as a goal at the end of the month.
- Logged out state :
    - All private information (user's income and expenses) must not be accessed, viewed, or modified once the user is logged out of an account.
- Page not found page must appear when user inputs a URL that does not exist for the application.

2. Attack Surface Analysis and Reduction

Users will either be regular or privileged. Regular users will only have access to their accounts and data, and privileged users will have access to all accounts and data. Privileged users will only be those involved in creating the application. To perform an attack surface analysis, we analyzed an application called Money Lover, a personal finance application similar to EarnIt. Money Lover tracks and categorizes the user's in-and-out money and creates manageable budgets. Here is a list of possible vulnerabilities for Money Lover that could apply to our application:

- Injection - An attacker could execute a command on the user data database that deletes or changes information such as email, password, and financial information.
- Broken authentication and session management - An attacker could obtain users' passwords and access their accounts.
- Sensitive data exposure - An attacker could obtain users' personal financial information such as their income, bills, and spending habits.
- Broken access control - A user could possibly wrongly access other users' information, or a regular user could be wrongly granted privileged access.
- Cross-site scripting - An attacker could use scripts to redirect users to malicious content.
- Insecure deserialization - An attacker could access serialized objects, execute injections, and elevate user privileges.
- Cross-site request forgery - An attacker could trick an authenticated user into performing an unintended action.

3. Threat Modeling

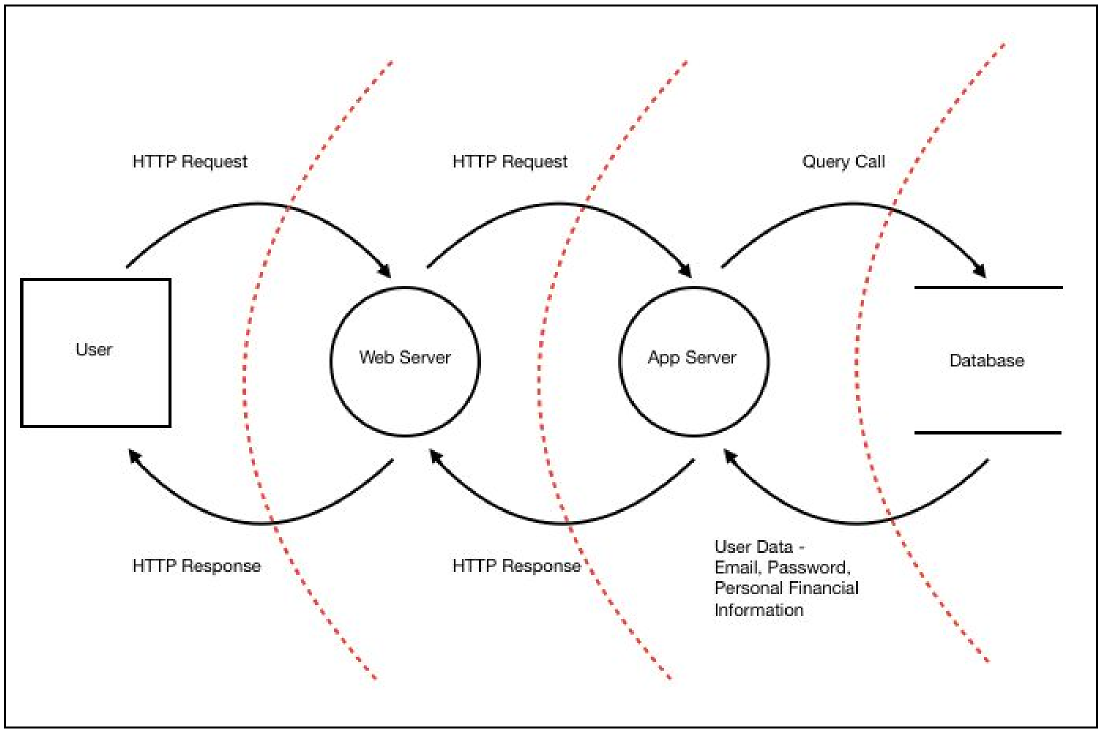


Figure 1: Threat Model of EarnIt, a Meteor Web Application

Table 5: STRIDE Threat Model

Threat	STRIDE Category
Injection	Tampering with data
Broken authentication	Spoofing identity
Sensitive data exposure	Information disclosure
Broken access control	Elevation of privilege
Cross-site scripting	Tampering with data
Insecure deserialization	Tampering with data / Elevation of privilege
Cross-site request forgery	Spoofing identity

D. Implementation

In the Implementation phase of the Security Development Lifecycle, we will establish a list of approved tools that will be used throughout the development process, a list of banned/deprecated functions that should not be used in our application, and describe our secure software development plan using a static analysis tool called ESLint. If we follow the guidelines that we have created in this section, we will be able to identify and remove possible security issues during the early periods of development.

1. Approved Tools

Table 6: Comprehensive List of Tools Needed for EarnIt

Development Tool	Optimal Version (Recommended)	Comments
Meteor	1.8.1	● A JavaScript App Development Platform ● Check version using "**meteor --version**" command in project directory → not /app directory ● If not 1.8.1, run "**meteor update**" command in project directory → not /app directory ● Meteor Documentation
MongoDB Database	4.0.6	● Meteor uses a "NoSQL Database" ● Check version using "**meteor mongo**" command in /app directory while running the application in another terminal ● Meteor/Mongo Documentation
IntelliJ IDEA Ultimate	2019.1.3	● Integrated Development Environment (IDE)
ESLint	5.16.0	● Static Code Analysis Tool ● Run "**meteor npm install eslint**" command in /app directory to install/update ESLint

Node Package Manager (NPM)	6.9.0	● Installation of Javascript Packages into web application ● Check version using "**npm -v**" or "**npm --version**" command in /app directory. ● Run "**meteor npm install**" command in /app directory.
GitHub GitHub Desktop	N/A (Website) 2.0.3	● Version Control, Source Code Organization, and Project Task Organization

2. Deprecated/Unsafe Functions

Table 7: List of Deprecated Javascript functions

Deprecated Javascript	Reason	Alternative	Example
dateObj.getYear()	Year 2000 Problem	getFullYear() → Date Object Method	dateObj.getFullYear()
dateObj.setYear(value)	Year 2000 problem	setFullYear() → Date Object Method	dateObj.setFullYear(1992)
Number.toInteger(value)	Lack of Browser Compatibility	Math.round(value)	Math.round(3.14)

for each … in statement	No longer Supported	for … of statement	for(let i of array) {
			console.log(i);
			}

for … in	No longer Supported	for … of statement	for(let i of array) {
			console.log(i);
			}

Table 8: List of Deprecated Meteor packages

Deprecated/Not Recommended Meteor Packages/Functions	Reason	Alternative
Session	Session variables are in a global namespace	Use “reactive-dict” package
Meteor.uuid	No validation on server side to check the validity of the “_id” being passed is a random value	Use “random” package → Random.id()

3. Static Analysis

Static Analysis tools allow developers to identify possible issues in their source code and rectify those issues without the need to run the application. Majority of our application is composed of JavaScript files. Since our application uses JavaScript, which does not have a compiler, most errors will be identified during application execution. Therefore, it must be required for us to use these tools during application development because it ensures consistent coding standards, promote high-quality code, and reduces the possibility of having security vulnerabilities.

During application development, we have decided to use ESLint, a JavaScript linting tool, to improve our application quality. ESLint contains hundreds of built-in rules that is used to identify and notify the developers of any inconsistent coding standards and possible security issues that are present, such as unsanitized input. ESLint can be incorporated into the IntelliJ IDEA Ultimate using one of the three following methods: “npm install eslint -g” (for global installation), “npm install eslint ---save-dev” (to install ESLint as a development dependency) or as a plugin.

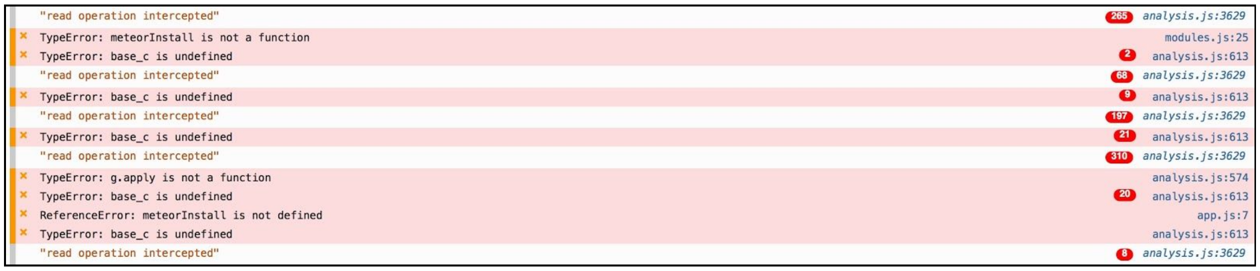
So far the experience using ESLint in the IntelliJ IDEA has been very helpful. It is extremely easy to tell general errors such as syntax errors, and coding style errors since it gives the specific line within the code that has the error and the reason for the errors. By strictly following the coding style rules using ESLint, it forces our code to stay consistent no matter who wrote it, which helps us write a more secure code.

E. Verification

In the Verification phase of the Security Development Lifecycle, we will briefly describe a dynamic analysis tool called Jalangi and explain our plans of how we will be using the tool to help identify any issues during application execution. Dynamic analysis tools allows you to identify complex vulnerabilities during runtime, such as user privilege issues, SQL injection and buffer overflows, that were not detected by static analysis tools. We will also conduct an attack surface review, in which we will report any patches/updates to any of the approved tools that were listed in Table 6. Conducting an attack surface review is important because we will be able to identify new vulnerabilities that may exist as a result of patches/changes to our approved tools. In conclusion, our application will be able to run correctly and securely if we follow the guidelines discussed in the following subsections.

1. <u>Dynamic Analysis</u>

Jalangi, which is a JavaScript dynamic analysis tool, will be used during our application development. The tool is available as a Firefox extension. The browser extension will be able to capture and monitor every operation in the JavaScript code. This will allow us to determine if there are any issues/vulnerabilities that may exist in our code. Besides the default tests that are provided, Jalangi also allows custom tests. However, we will use the default tests to test our application. The tool is proving to be useful as seen in Figure 2. If we are able to fix the issues that are being discovered by Jalangi, the quality of our application will better.



```
  "read operation intercepted"                      265 analysis.js:3629
× TypeError: meteorInstall is not a function             modules.js:25
× TypeError: base_c is undefined                      2  analysis.js:613
  "read operation intercepted"                       68 analysis.js:3629
× TypeError: base_c is undefined                      9  analysis.js:613
  "read operation intercepted"                      197 analysis.js:3629
× TypeError: base_c is undefined                     21  analysis.js:613
  "read operation intercepted"                      310 analysis.js:3629
× TypeError: g.apply is not a function                   analysis.js:574
× TypeError: base_c is undefined                     20  analysis.js:613
× ReferenceError: meteorInstall is not defined               app.js:7
× TypeError: base_c is undefined                         analysis.js:613
  "read operation intercepted"                        8 analysis.js:3629
```

Figure 2: Jalangi was able to discover several issues found in our JavaScript code.

2. Attack Surface Review

Table 7: List of Approved Tools Needed for EarnIt. Patches and Updates will be indicated in "red" (Updated June 30, 2019).

Development Tool	Optimal Version (Recommended)	Comments
Meteor	1.8.1	● No Updates/Patches
MongoDB Database	4.0.6	● No Updates/Patches
IntelliJ IDEA Ultimate	2019.1.3	● Next release will be 2019.2.0, which will be released in July 2019
ESLint	5.16.0	● Next release will be 6.0.0. (Possibly be released during our application development)
Node Package Manager (NPM)	6.9.2	● Updated from 6.9.0 to 6.9.2
GitHub GitHub Desktop	N/A (Website) 2.0.4	● GitHub Desktop Version from 2.0.3 to 2.0.4

3. Fuzz Testing

1) **NoSQL Injection** - MongoDB is a NoSQL database, which means that the data is stored as "documents" in the form of BSON (Binary JSON). Each document has its own unique structure. Although the SQL language is not used in MongoDB, a different type of attack is possible, NoSQL injection. By default, all Meteor projects have the insecure package installed, which allows anyone to access and alter the MongoDB database from their console. For example, we were able to write anything to the profile field of my Meteor.users document when logged in. To fix this, we added a server deny function that prevents users from updating their own document. This stops users from potentially stored malicious content in the database or altering information that we may store in the profile field in the future.

2) **Sensitive Data Exposure** - While logged in, we attempted to access another user's information by searching the database from the console for documents associated with their email. However, an "access denied" error was returned. This is because Meteor implements a publish/subscribe mechanism for its data collections. When a user logs in, the app searches the database and only publishes documents associated with the account. Then, when a user accesses a page that uses any of that information, a subscription to the published data makes it available to the user. This prevents users from accessing others' information.
3) **Cross-Site Request Forgery** - We attempted to make an HTTP request that might change a user's password if they unknowingly clicked on a link that executed the action. However, cross-site request forgery is impossible on Meteor apps because they use localStorage tokens instead of session cookies. This makes the app more secure because it prevents attackers from discreetly sending users links that may result in the execution of an HTTP request that results in malicious activity.

4. Updated Static & Dynamic Analysis

Our static and dynamic analyses have produced results similar to our previous examinations. We have continued to use IntelliJ IDEA and ESLint for our static analysis to maintain high quality code. Likewise, we still use Jalangi for our dynamic analysis, and although some of the errors have persisted since our last check-in, we are continuing to work to resolve them so as to produce the most secure app.

F. Release

1. Incident Response Plan

Privacy Escalation Team

Role	Role Description	Name
Escalation Manager	The escalation manager will be responsible for determining the severity of the incident and determine the right course of action.	Mai Abe
Legal Representative	The legal representative will handle any legal issues related to the application.	Julie Chai
Public Relations Representative	The public relations representative will handle any questions or concerns about the application. Also, will provide the public with all the incident information (cause, possible consequences, and solutions).	Mai Abe
Security Engineer	The security engineer will continue to monitor and identify any security vulnerabilities of the application after the release. When any security vulnerabilities occurs, the patches or application updates will be created.	Jun Okabe

Our Privacy Escalation Response process will be the following:

1) The Privacy Escalation Response team is notified of an incident. The Escalation manager will be the first to be notified of the situation and will determine the correct course of action.
2) Determine the source of the escalation and determine the impact that it will have.
3) Determine the validity of the incident and allocate resources to the incident. If valid, create a summary of all known facts on the incident to help understand it.
4) Create a timeline of expectations or tasks, which includes a security patch (by the Security Engineer), prepare for any concerns/questions and provide information on the

incident to the public (by the Public Relations representative), and prepare for any legal affairs regarding the incident (by Legal Representative).

5) Properly store the documentation of the incident for future references.

To reach our Privacy Escalation Team, please contact the team using the following email address: earnit.jmj@gmail.com.

2. Final Security Review

During the final security review, we must examine all the security activities performed before the release of our application. Using the threat model that was created (Figure 1), static analysis, dynamic analysis, and quality bug bars, the outcome of our final security review is “**Passed FSR with exceptions**”. “Passed FSR with exception” indicates that all security and privacy issues have been mitigated during application development and we have recorded all remaining issues and will be corrected in the next release.

As a result of the final security review, some of the vulnerabilities we confirmed that were able to prevent at our satisfaction were Injection, Sensitive Data Exposure, Cross-Site Request Forgery, along with Broken Authentication and Insecure Deserialization. For Injection, Sensitive Data Exposure and Cross-Site Request Forgery, please refer to the document's "Fuzz Testing" section as the results were the same.

- For Broken authentication, we confirmed a satisfactory prevention because whenever the user needs access to their data, corresponding page only allows access to the data with correct subscription. Meteor.subscribe() method takes care of authenticating the user to access to the correct data.
- For Insecure deserialization, we confirmed a satisfactory prevention because Meteor uses hashed passwords to store user passwords. Though this does not guarantee a completely secure authentication method, we ensured that passwords were not transmitted in plain-text format.

In our next version, we will need to remove all insert, update, and remove collection calls and replace them with appropriate server Meteor functions. This will prevent users from being able to alter the database from their console. Furthermore, we will need to implement admin pages in order to improve the access control mechanism for the app. Implementing admin pages will allow the app to have more control over data.

3. Certified Release & Archive Report

Installation Instructions:

1) Install Meteor
2) Download a copy of EarnIt project or clone the project using Git.
3) To update packages used in the EarnIt project, go to the **app** directory of the project using the **cd** command.
4) In the **app** directory, enter the following command **meteor npm install**.
5) To run the project, enter the following command **meteor npm run start**.
6) The application will be available for use in http://localhost:3000.
7) If you would like to enable emails for the forgot password feature, you will need to follow additional instructions detailed here.

Application Feature List for Version 1.0:

- For first-time users, the new user would need to create an account using the **Sign-up** Page. In the **Sign-up** page, the user will input an email address, which will be used as the username, a password that needs to contain one number and be more than 6 characters long and agree to the privacy agreement.
- For existing users, the user will need to login using the **Login** Page. If the user has forgotten their password, the user may use the "Forgot your password" feature to send a link to reset their password.
- At the **Overview** page, the user or a visitor may be able to get an "at-a-glance" overview of the application.
- For first time users, the user will be asked to create a profile. The profile will store the name of the user and the amount of money the user wants to save for each month. For existing users, the profile information can be updated in the **Profile** page.

- At the **Profile** page, the user is able to enter their earnings/income. The history of their income and earnings will be available at the bottom of the page. The user will be able to update or delete each entry in the income table.
- At the **Expense** page, the user is able to create categories for organization. For each category, the user is able to add expenses or view their expense history for a category. In the expense history table, the user will be able to edit or delete their expense.
- At the **Userhome** page, the user is able to visually see all their income and expenses for the current month and current year. This page will also allow the user to check if they have reached their monthly savings goal.

Future Development Plans:

- Create a better visual design of the application
- Verify email address after user sign-up
- Two-Factor Authentication using user's email address
- Session Timeout after 30 minutes of inactivity by the user

The EarnIt project repository is available on GitHub at https://github.com/team-jmj/EarnIt.

Version 1.0 of the application is available at https://github.com/team-jmj/EarnIt/releases/tag/v1.0.

The EarnIt GitHub Wiki page, which contains a user guide, is available at https://github.com/team-jmj/EarnIt/wiki.