Cash Companion

A. Introduction

Team Name: Byte Bandits

Team Members: Coltin-Kai Kaupiko, Cathy Kim, Gavyn Gostage, Steven Le

Github URL: https://github.com/Cash-Companion/Cash-Companion

App Title: Cash Companion

a. **Description:** A web application resembling a budget tracker that is capable of managing expenses. The user has the ability to input spendings. The app will then calculate monthly statistics on the user's finances. The app also will offer budgeting and categorical sorting.

b. **Functional Requirements:**

- Ability to input expenses
- Ability to set budget/expense limit
- Spending Categories
- Income
- Total Spending
- Sign-Up/Login Feature.

Type of Program: Web Based Application

Development Tools:

- JavaScript
- HTML
- CSS
- Meteor
- React
- MongoDB
- IntelliJ IDEA

B. Requirements

1. Security and Privacy Requirements

The main security requirement for our web based budget tracker will be the protection of personal financial user data. This includes information such as income, expenses, budgeting categories/goals, and general PII which should be accessible to only the user with the exception of the administrator for development and data management. A user will be able to access their private financial information that they input in our tracker by logging in with an email address and password. Private user data should be not accessible to anyone other than the logged in user. Our goal is to create an application where users can log in and conveniently track/log their spending.

The privacy of user data is also a critical requirement of Cash Companion. It will be ensured that financial information is not shared or distributed amongst third parties without explicit user consent. The application will be aimed to be transparent and clear regarding how user data is collected, stored, and used. Additionally, users will be provided with the ability to delete their data from our database in the case that they no longer want to use Cash Companion.

To maintain these security and privacy requirements, GitHub will be used during the developmental and deployment phases of our application. This will ensure that bugs or security flaws are resolved as soon as possible by our team to minimize any vulnerabilities. It will also allow the Cash Companion team to manage tasks and add future software features to the application through continuous integration. Any and all security vulnerabilities will be prioritized and fixed before any further development.

2. Quality Gates (or Bug Bars)

Privacy Bug Bars

Critical

- Lack of notice and consent
  Example: Transfer of personal information without prominent notice.
- Lack of user control
  Example: Users do not have the ability to stop subsequent collection and transfer of personal information.
- Lack of data protection

Example: Users cannot access and correct stored personal information in a general database.

- Improper use of cookies

  Example: Personal information stored in a cookie is not encrypted.

Moderate

- Data minimization

  Example: Personal information transmitted to a third party.

Low

- Lack of notice and consent

  Example: Personal information is collected and stored in a hidden database. PII is not accessible by others.

Security Bug Bars

Critical

- Elevation of privilege (ability to either execute arbitrary code or obtain more privilege)

  Example: Execution of arbitrary code.

Important

- Information disclosure

  Example: Disclosure of personal information (email addresses, incomes, etc.)

- Tampering (modification of any user data)

  Example: Modification of OS/application settings without user consent, and modification of user data.

Moderate

- Denial of service (Causes Blue Screen/Bug Check)

  Example: Service failure.

Low

- Information disclosure (untargeted)

  Example: Leak of random heap memory.

3. Risk Assessment Plan for Security and Privacy

a. *Identify the Risks*
   i. Unauthorized access to user data
   ii. Data breaches and loss of sensitive information
   iii. Inadequate data protection and encryption measures

b. *Assess Likelihood of and Impact of Risks*
   i. Unauthorized access: High likelihood, high impact
   ii. Data breaches: Medium likelihood, high impact
   iii. Inadequate data protection: High likelihood, medium impact

c. *Develop Risk Mitigation*
   i. Implement robust authentication and access controls to limit unauthorized access
   ii. Use encryption for all data storage and transmission to protect against data breaches
   iii. Comply with industry-standard data protection regulations and guidelines

d. *Monitor and Review*
   i. Continuously monitor the system for suspicious activity and respond promptly to any security incidents
   ii. Regularly review the risk assessment plan and update it as needed to reflect changes in the system and emerging threats

e. *Train the User*
   i. Provide training and resources to help users understand the importance of security and how to protect their sensitive information
   ii. Encourage them to use strong and unique passwords, and to monitor their account for any suspicious activity.

f. *Threat Modeling (detailed more on pg. ___)*
   i. Identify possible threats to privacy and security
   ii. Identify parts that require more security than others

C. Design

1. **Design Requirements**

   Data Management Policy

- User data is stored in MongoDB to prevent data from being lost and protected from malicious actors. Access to MongoDB is restricted by service code and calls to access the MongoDB must be performed through the service code. Only authenticated requests are allowed to access data.
- Application is implemented with HTTPS certificates to provide a secure environment without tracking the user data and ensure encryption for user data. The application and server only communicate by an encrypted connection to prevent data from being intercepted.

  Password Management Policy

- All passwords are to be treated as sensitive, and must never be stored in readable format or in code files. All passwords must be encrypted. Passwords must never be accessible to unauthorized individuals.

2. **Attack Surface Analysis and Reduction**

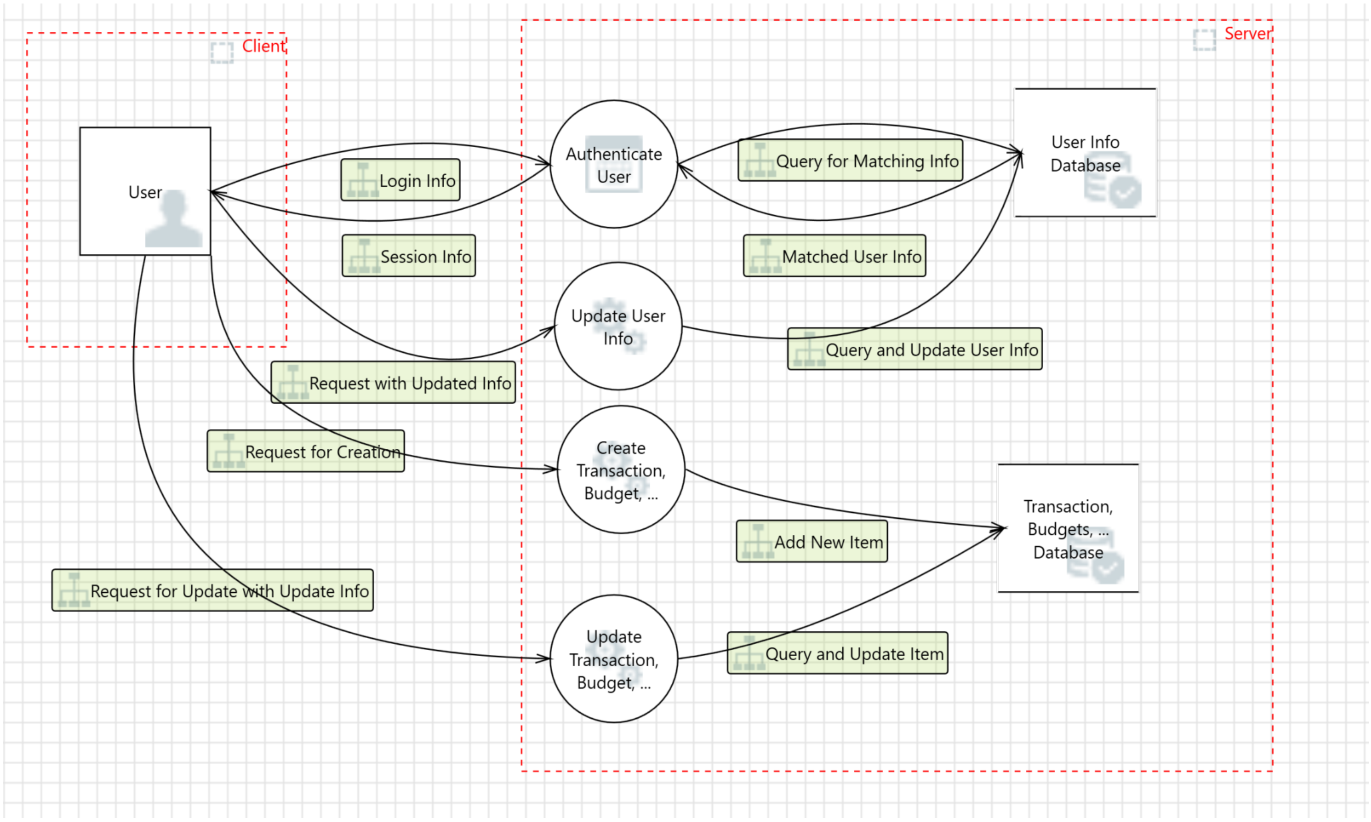
Privilege Level	**Features/Permissions**
Anonymous Users	● Log-In, using Username and Password ● Sign-Up, using Personal Information
Authorized Users	● Inputting Transactions, Making Budgets, etc ● Edit their (Personal) Account Information. This may include name and email.
Authorized Admin Users	● Edit their (Personal) Account Information ● Access to Admin Interface ○ Can Remove User Accounts ○ Access to User Personal Information

Anonymous Users are those that have not provided a username and password to request a session. They have the least permissions, but can use the log-in or sign-up features. Authorized Users are those who have successfully logged-in and are provided the appropriate session. They have access to the ability to make transactions and other features meant for consumers. Authorized Admin Users have the highest level of privilege, being able to manage other user accounts.

Potential Vulnerabilities:

- **Invalid Access**
    - Missing Function Level Access Control - The failure of an authentication check on a user, allowing them to access unauthorized information and features
    - Insecure Direct Object Reference - exposure of reference to supposedly secure objects
    - Directory/Path Traversal
    - Session Hijacking
    - Unrestricted URL Access
- **Authentication**
    - Weak Passwords and Empty String Password
    - Lack of/Weak Hashing of Passwords
    - Brute Force Attack
- **Injections**
    - Code Injections
    - CRLF (Carriage Return Line Feed) Injection
    - SQL Injection

3. Threat Modeling


Client
Server
User
Login Info
Session Info
Authenticate User
Query for Matching Info
Matched User Info
User Info Database
Update User Info
Query and Update User Info
Request with Updated Info
Request for Creation
Create Transaction, Budget, ...
Add New Item
Transaction, Budgets, ... Database
Request for Update with Update Info
Update Transaction, Budget, ...
Query and Update Item

Spoofing	• Man-in-the-Middle Attack between Client and Server • Phishing of login information of User
Tampering	• Injection by malicious User through entry points ○ Code Injection ○ CRLF Injection ○ SQL Injection
Repudiation	• Lack of checking authenticity of User • Web Parameter Tampering to change info being sent from Client to Server
Information DIsclosure	• Misplaced sensitive data on web App that allows Users to reach • Weak measures to protect Databases from unwarranted queries
Denial of Service	• Weak measures and recovery to protect Server from DoS Attacks caused via high traffic such as Distributed DoS • Application Layer Attacks that can abuse requests to overwhelm the Server • Abuse of the unrestricted limit on resources stored on Database and

	Server, taking away available memory
Elevation of Privilege	• Manipulation of parameters of sent information from Client to Server that allows access of higher privileged data • Lack of authorization checking on higher privilege features and functions • Lack of preventive measures to stop URL Traversal that bypasses authorization checking

Implementation

Approved Tools

Tool	**Version**	**Comments**
React	18.2.0	Front-end JS library based on UI components
React-bootstrap	2.7.0	Component based library
React-dom	18.2.0	Package that provides methods to manage elements of web page
React-router	6.7.0	Library for routing in React
Node	12.20.10	Server environment for running web applications
Prop-types	15.7.5	Mechanism that ensures correct datatype
Mongo	3.0.1	Document oriented database tool

Meteor	2.2.0	JavaScript web framework
IntelliJ IDEA	Recent Version	IDE
Eslint	8.32.0	Tool to identifying patterns found in code to avoiding bugs
Testcafe	2.2.0	Open-source test runner
Simple Schema	3.4.1	Package that supports direct validation of MongoDB
Sweetalert	2.1.2	JavaScript's popup boxes

Deprecated/Unused Functions

- React
  - *componentWillMount*
    - Instead use: *useEffect*
  - *componentWillUpdate*
    - Instead use: *getDerivedStateFromProps*
  - *React.createClass*
    - Instead we now use ES6 classes or functional components with hooks
- MongoDB
  - *db.collection.insert*
    - Instead use: custom method (defineMethod)
  - *db.collection.remove*
    - Instead use: custom method (removeItMethod)
  - *db.update*
    - Instead use: custom method (updateMethod)

- React-router
  - <Route> component now has deprecated children & render props. it is no longer possible to pass router props to components.
    - Instead use: useNavigate/useLocation/useParams hooks
  - withRouter component for getting router props deprecated
    - Instead use: React Hooks
  - Switch components
    - Instead use: Routes
- Node
  - *timers.setTimeout()*
    - Instead use: *setTimeout()*
    - This works better because it operates in the global scope.

Static Analysis

We chose to use ESLint for the static analysis tool. ESLint is a popular open-source project that offers a suite of static analysis tools for JavaScript and JSX. This tool scrutinizes code to uncover syntax errors and anti-patterns, helping to establish uniform code styles. It also detects errors in code and provides an explanation of each error, enabling team members to ensure consistent and accurate code.

One of the key advantages of using ESLint is that it promotes consistency across coding conventions. Additionally, it streamlines the debugging process by identifying errors and providing clear descriptions of what went wrong. By enforcing consistent coding styles, ESLint can also enhance the readability of code written by other team members, making collaboration more efficient and effective. Overall, ESLint is an extremely valuable static analysis tool for any team looking to maintain consistency and improve code quality.

One potential disadvantage of using ESLint is that customizing it to suit the needs of a specific project can be time-consuming. It's worth noting that if we fail to set up and maintain our rules in ESLint correctly, it may run the risk of overlooking important issues or spending unnecessary time reviewing false positives. Therefore, it's essential to ensure the rules are configured correctly and regularly updated to align with our project's needs. By taking the time to establish and maintain the rules effectively, we can help ensure that our team benefits fully from using ESLint as a static analysis tool.

A few examples of the ESLint rules that our team is using:

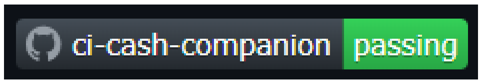
Rules	Description
prefer-const	Require const declarations for variables that are never reassigned after declared
no-duplicate-imports	Disallow duplicate module imports
no-unused-vars	Disallow unused variables

Implementation & Verification

Dynamic Analysis

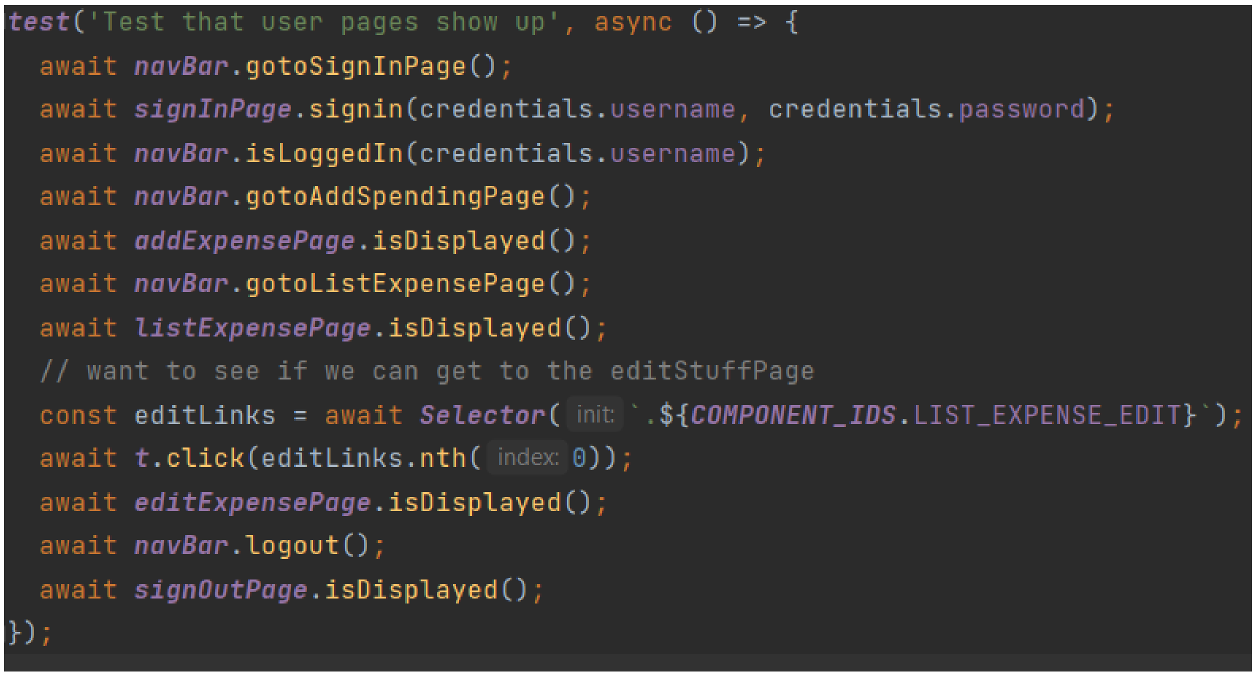
We use TestCafe to perform dynamic analysis of our application through comprehensive end-to-end testing. TestCafe is an open-source testing framework that is highly adaptable and compatible with JavaScript applications, running effectively on various platforms. Its remarkable feature of verifying application UI stability is integrated into our template, making it highly convenient to use. With TestCafe, we can quickly test all the features of our application to ensure that there are no errors after implementing new features or updating the code.

The beauty of it all is that TestCafe automatically runs once a team member's branch is merged into the main branch, which is an excellent way of ensuring that our application operates smoothly without any glitches. This allows us to stay ahead of the curve and avoid any unforeseen issues that may arise during development.

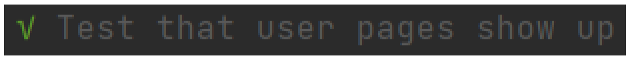


One of the significant advantages of utilizing end-to-end testing through TestCafe is that it encompasses the entire application, including the server and front-end UI. TestCafe operates by controlling a web browser, such as Google Chrome, from a remote location, allowing for simulated clicks and assessments of particular nodes in the Document Object Model (DOM). By using TestCafe, we can ensure that any alterations to the application do not impede its proper functioning and that both the server and clients remain compatible.

However, implementing TestCafe does require the front-end developer to assign a unique 'id' tag to all DOM elements, which can be a time-consuming task. Additionally, the speed of the browser's DOM rendering process limits the time taken for testing. Although integration testing lacks the bottleneck of waiting for DOM rendering, it is unable to confirm that UI elements are rendered on the page. Thus, end-to-end testing from TestCafe offers crucial coverage in this aspect.



```
test('Test that user pages show up', async () => {
  await navBar.gotoSignInPage();
  await signInPage.signin(credentials.username, credentials.password);
  await navBar.isLoggedIn(credentials.username);
  await navBar.gotoAddSpendingPage();
  await addExpensePage.isDisplayed();
  await navBar.gotoListExpensePage();
  await listExpensePage.isDisplayed();
  // want to see if we can get to the editStuffPage
  const editLinks = await Selector( init: `.${COMPONENT_IDS.LIST_EXPENSE_EDIT}`);
  await t.click(editLinks.nth( index: 0));
  await editExpensePage.isDisplayed();
  await navBar.logout();
  await signOutPage.isDisplayed();
});
```



```
√ Test that user pages show up
```

Moving forward, we plan to supplement our dynamic analysis with integration tests, aiming for near-perfect code coverage. This approach will allow us to catch any issues early on and ensure that our application is of the highest quality. By utilizing TestCafe and integration testing, we can maintain our application's stability and ensure that it runs seamlessly on various platforms.

Attack Surface Review

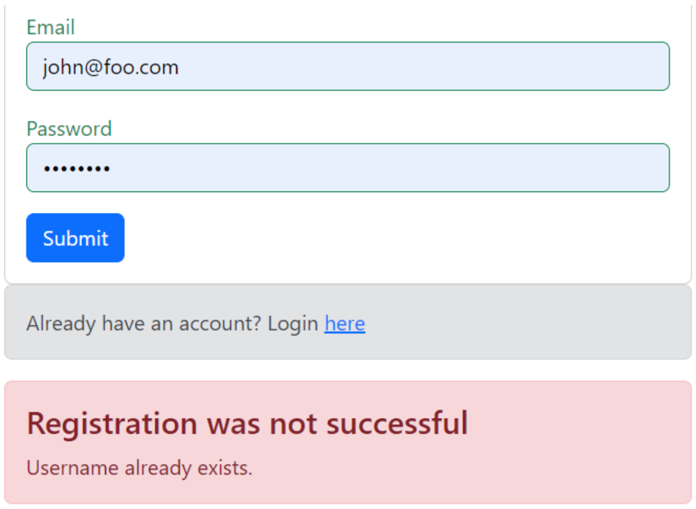
Since the submission of Assignment 2, there have been some updates made to the system. Despite these updates, there haven't been any vulnerabilities found so far.

Tool	Version	Changes, update, patches (Y/N)	Vulnerabilities
React	18.2.0	N	N/A
React-bootstrap	2.7.0	2.7.2	N/A
React-dom	18.2.0	N	N/A
React-router	6.7.0	6.9.0	N/A
Node	12.20.10	19.7.0	N/A
Prop-types	15.7.5	15.8.1	N/A
Mongo	3.0.1	6.0.1	N/A
Meteor	2.2.0	2.11.0	N/A
IntelliJ IDEA	Recent Version	N	N/A
Eslint	8.32.0	8.36.0	N/A
Testcafe	2.2.0	2.4.0	N/A
Simple Schema	3.4.1	N	N/A
Sweetalert	2.1.2	N	N/A

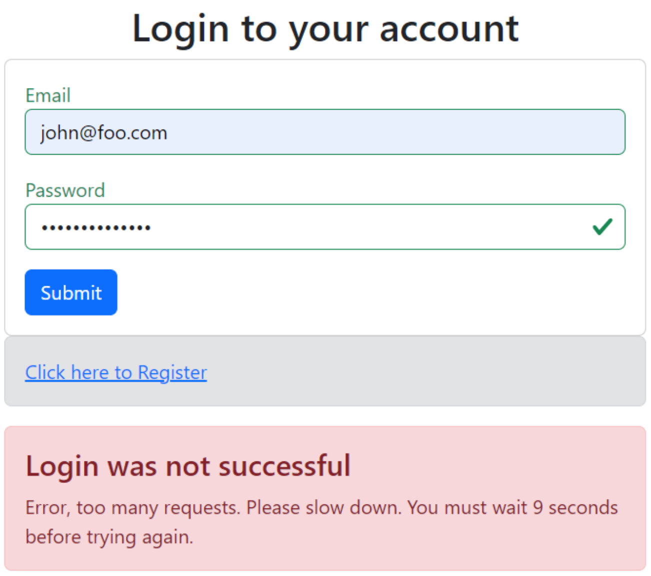
Verification

Fuzz Testing

When it comes to websites that require user accounts, one of the most vulnerable areas is the login page. To see just how secure a website was, we attempted to breach it by creating a new account with an email address that had already been registered. Our aim was twofold. First, we wanted to see if the website allowed multiple accounts with the same email. Second, we wanted to determine if multiple accounts would have access to the same information. So we went to the signup page and tried to create a new account with an email we knew had already been used. The website prevented us from doing so, displaying an error message that read "Username already exists." This refusal was due to proper error checking that had been added for account creation on the signup page using `*Meteor.loginWithPassword*`. Meteor is a framework used by this website for managing user accounts. It did not accept the new account and failed to create a new entry.


Email
john@foo.com
Password
••••••••
Submit
Already have an account? Login here
Registration was not successful
Username already exists.

Another way that attackers can try to gain unauthorized access to our website is through brute force attacks. This technique involves trying multiple combinations of usernames and passwords until the correct one is found. However, since our website doesn't have any password requirements, users can choose simple passwords that are easily guessed by hackers using common password cracking tools. Moreover, the website's current method of slowing down brute force attacks by imposing a 10 second cooldown after multiple failed login attempts isn't enough to deter determined attackers. To improve our website's security against brute force attacks, we should consider implementing stricter password requirements, such as setting a minimum password length. Additionally, we could increase the penalty for multiple failed login attempts, for example, by imposing a longer cooldown timer, or sending account owners an email notification if their account is flagged for multiple failed login attempts. By taking these steps, we can significantly reduce the risk of brute force attacks and enhance the overall security of our website.


Login to your account

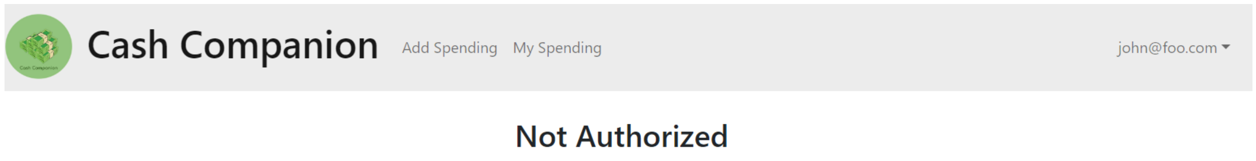
Email: john@foo.com

Password: ••••••••••••••

Login was not successful

Error, too many requests. Please slow down. You must wait 9 seconds before trying again.

Besides attacking the login page, another common way that attackers can try to break into a website is by attempting to access URLs that are not properly protected. For example, an attacker may try to access a page that they should not have access to, by guessing the URL or using a tool to automatically generate URLs. This type of attack is often referred to as forced browsing or URL guessing, and can be particularly effective if the website's URLs are easy to guess or not properly protected. Fortunately, our website uses unique IDs for each account, which makes it difficult for attackers to access pages that they do not own. This means that even if an attacker can guess the URL for a specific page, they will be denied access because they do not have the correct account ID. Additionally, we have implemented protected routes that prevent regular users from accessing admin pages or other sensitive areas of the website. These routes are designed to refuse access to users who do not have the proper credentials, adding an extra layer of security to our website. By implementing these measures, we can greatly reduce the risk of forced browsing attacks and protect our website from unauthorized access.



Not Authorized

Static Analysis Review

Using Eslint for our static analysis, we have minimal to no egregious errors without our project’s code. A majority of the errors outputted by Eslint involve violation of simple coding standard rules. This includes things like spacing and the lack of use of defined variables/classes. Simple refactoring can help clean up some of these errors. Additionally, we can ensure the quality of our code by executing the `*meteor npm run lint*` command in the terminal. This allows us to thoroughly examine our code for any errors or inconsistencies before we push the commit to our Github repository. Our use of the linter ensures that our code is properly structured and formatted, reducing the risk of any issues down the line.

Dynamic Review

Our website's dynamic analysis is being performed using TestCafe. We have thoroughly tested basic pages such as landing, sign up, sign in, and sign out pages, and confirmed that they are functioning correctly. Going forward, we plan to expand our testing range to include additional pages such as the adding expense page, my spending page, and managing account page. This will ensure that updated functions are working as intended. So far, TestCafe has proven to be a reliable testing tool, as we have encountered no issues during our testing efforts. Through these testing efforts, we have successfully confirmed that our website is functioning as intended, as all tests have been passed. We attribute this success to the effective use of TestCafe in verifying the accuracy of our website's functions. Moving forward, we will continue to utilize TestCafe to maintain the functionality of our website and ensure that it performs optimally for our users.

Release

Incident Response Plan

Privacy Escalation Team

- Escalation manager: Coltin-Kai Kaupiko
    - Organizes, and prioritizes activities related to gathering evidence, examining it, and containing incidents during the incident response process. Evaluate the severity and type of incident and determine appropriate actions accordingly.
- Legal representative: Steven Le
    - Offers legal guidance and recommendations to the team during the incident response. Provides representation to the team in case the incident leads to criminal charges, ensuring legal rights and interests are protected throughout the legal proceedings.
- Public relations representative: Gavyn Gostage
    - Responsible for communication with the general public. All statements must be approved before being disseminated, and media inquiries should be directed to public relations representative for responses.
- Security engineer: Cathy Kim
    - Offers expert technical knowledge, abilities, and support to help with identifying, containing, and resolving incidents. Responsible for maintaining the incident response plan and adjusting security measures as required.

Contact Email

- Coltin-Kai Kaupiko
    - coltin@hawaii.edu
- Cathy Kim
    - ckim60@hawaii.edu
- Gavyn Gostage
    - ggostage@hawaii.edu
- Steven Le
    - stevenle@hawaii.edu

If you have any queries or apprehensions, please feel free to contact us at any time.

Incident Procedure

1. In the event that sensitive user information is involved, our public relations representative will notify all users of the issue and advise them to protect their information, such as by changing their passwords or deleting their user accounts if necessary. We will work to resolve the issue as soon as possible.
2. The escalation manager will assess the issue and develop a plan to address it. If multiple issues arise simultaneously, the escalation manager will prioritize and assign tasks based on the severity of each issue.
3. Each team member will complete their assigned tasks according to the plan. For instance, the security engineer will focus on fixing vulnerabilities and performing static and dynamic analyses to ensure a complete resolution.
4. If user information is successfully attacked, our legal representative will communicate with the victims.
5. Once the issue is resolved, our public relations representative will inform users of the positive news to alleviate their concerns.
6. Finally, the escalation manager will summarize all known facts related to the incident.

Final Security Review

We have found that the threat model remains unchanged and continues to reflect the expected interactions of our software. In terms of static analysis, we have run the `*meteor npm run lint*` command, which has not yielded any significant errors, indicating that our code quality meets the necessary standards. Additionally, since our last dynamic analysis review, we have implemented extra tests to ensure that our website pages and components are functioning as intended. As a result, our project has passed all tests, confirming that everything is working properly.

After reviewing the quality gates established in the requirements section of the document, we have identified two significant issues that require our attention. The first issue pertains to user privacy, specifically the lack of control given to users regarding their account. While users have the ability to delete their account, this action can only be performed by admins. Moreover, users are currently unable to download their own collection of data, representing a critical privacy concern that must be addressed. The second issue relates to security, specifically the risk of spoofing and brute force attacks that could compromise user accounts. Our current measures to address these risks are inadequate, with penalties for multiple failed login attempts being too lenient at just ten seconds. Additionally, there are no significant password restrictions in place, further exposing our users to potential security breaches. It is essential that we take concrete steps to enhance our security measures and protect our users' personal information.

Certified Release & Archive Report

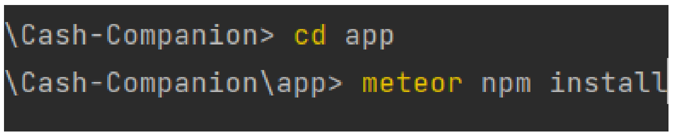
The released version of Cash Companion is available here:
https://github.com/Cash-Companion/Cash-Companion/releases/tag/1.0

Cash Companion is an intuitive web-based application designed to help users keep track of their expenses and manage their budget more effectively. With Cash Companion, users can easily input their spending and categorize it according to their needs. The app then automatically calculates their total spending and generates a comprehensive pie chart showing the breakdown of expenses by category. By providing users with a clear and concise overview of their spending habits, Cash Companion empowers them to make informed decisions about their finances and achieve their financial goals more efficiently. Whether you're trying to save money, pay off debt, or simply keep track of your expenses, Cash Companion is the perfect tool to help you stay on top of your finances.

Technical Notes

Installation

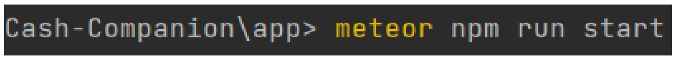
1. First, install Meteor.
2. Second, go to https://github.com/Cash-Companion/Cash-Companion, and click the "Code" button.
3. Third, click the "Clone or download" button to download Cash Companion to your local file system. Using GitHub Desktop is a great choice if you use MacOS or Windows.
4. Fourth, cd into the app/ directory and install required libraries



```
\Cash-Companion> cd app
\Cash-Companion\app> meteor npm install
```

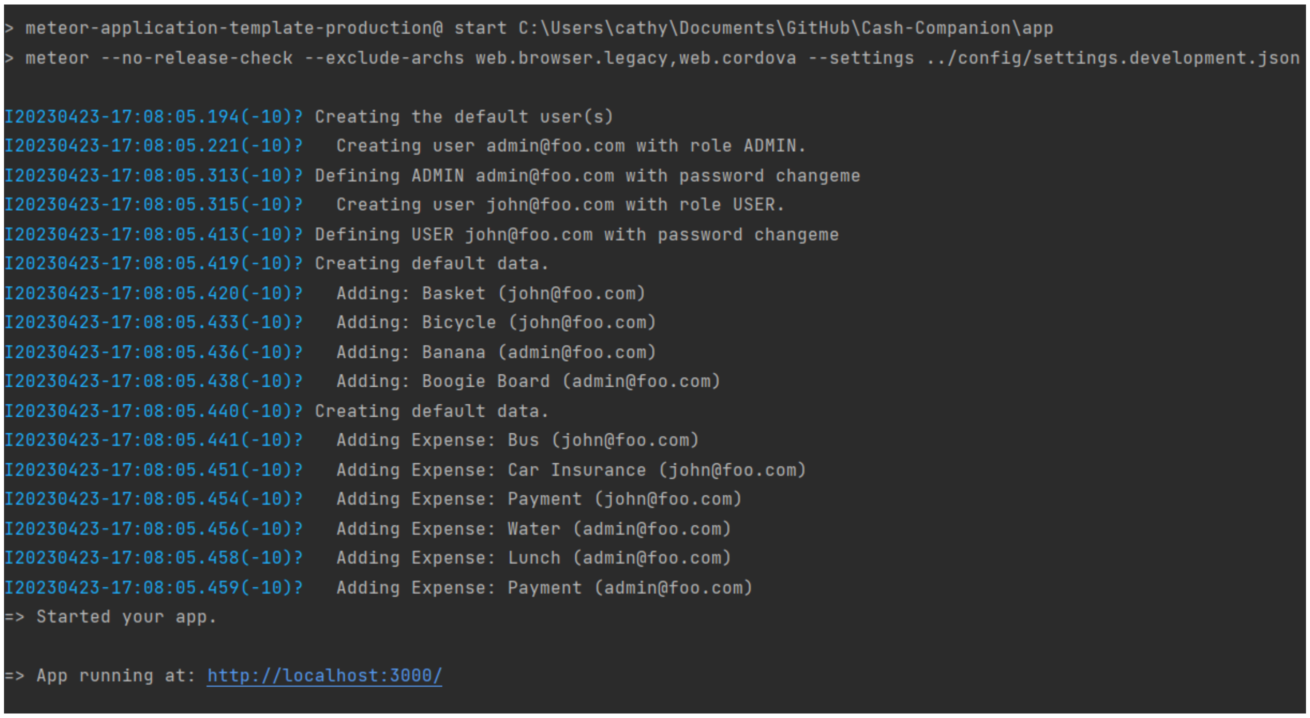
Running App

1. After installation, you can run the application by typing `*meteor npm run start*`.



```
Cash-Companion\app> meteor npm run start
```

2. Running the application for the first time will add default users / default admin / default data

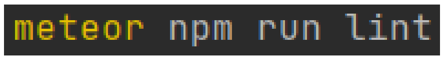


```
> meteor-application-template-production@ start C:\Users\cathy\Documents\GitHub\Cash-Companion\app
> meteor --no-release-check --exclude-archs web.browser.legacy,web.cordova --settings ../config/settings.development.json

I20230423-17:08:05.194(-10)? Creating the default user(s)
I20230423-17:08:05.221(-10)?   Creating user admin@foo.com with role ADMIN.
I20230423-17:08:05.313(-10)? Defining ADMIN admin@foo.com with password changeme
I20230423-17:08:05.315(-10)?   Creating user john@foo.com with role USER.
I20230423-17:08:05.413(-10)? Defining USER john@foo.com with password changeme
I20230423-17:08:05.419(-10)? Creating default data.
I20230423-17:08:05.420(-10)?   Adding: Basket (john@foo.com)
I20230423-17:08:05.433(-10)?   Adding: Bicycle (john@foo.com)
I20230423-17:08:05.436(-10)?   Adding: Banana (admin@foo.com)
I20230423-17:08:05.438(-10)?   Adding: Boogie Board (admin@foo.com)
I20230423-17:08:05.440(-10)? Creating default data.
I20230423-17:08:05.441(-10)?   Adding Expense: Bus (john@foo.com)
I20230423-17:08:05.451(-10)?   Adding Expense: Car Insurance (john@foo.com)
I20230423-17:08:05.454(-10)?   Adding Expense: Payment (john@foo.com)
I20230423-17:08:05.456(-10)?   Adding Expense: Water (admin@foo.com)
I20230423-17:08:05.458(-10)?   Adding Expense: Lunch (admin@foo.com)
I20230423-17:08:05.459(-10)?   Adding Expense: Payment (admin@foo.com)
=> Started your app.

=> App running at: http://localhost:3000/
```

3. The Cash Companion application will appear at http://localhost:3000

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
```

Links

1. Github
   - https://github.com/Cash-Companion/Cash-Companion

2. Release
    - https://github.com/Cash-Companion/Cash-Companion/releases/tag/1.0
3. README
    - https://github.com/Cash-Companion/Cash-Companion/blob/main/README.md
4. Wiki
    - https://github.com/Cash-Companion/Cash-Companion/wiki