ICS 427 Summer 2022

UHBadger Web App: Requirements and Design

INTRODUCTION

Team Name: Team UHBadger

Team Members: Feimei Chen, Hansen Cabanero, and Cheolhoon Choi

Application Title: UHBadger

Description:

UHBadger will track and analyze users' spending to help them optimize their spending. Users can record every spending and plan what they want to spend. UHBadger will show the balance and every expenditure to help assist users in spending more than they earn. In addition, UHBadger will calculate the percentage of each area of expenditure, such as food and cars. Users can enter an amount of money they want to spend in a specific period, and at the end of the period, UHBadger will calculate the total amount they have spent and then calculate how much they have saved or overspent since the start of the specified period. In general, UHBadger is a money management and planning application that will allow for user and admin accounts. Users are able to freely sign up with a user account. The Admin can remove and edit user accounts.

Function Requirement Specification:

Function Requirements

- Sign up
- Login and logout
- Create a planner
- Edit, Delete and add spendings

Development Tools

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

Type of Program: Web Application

REQUIREMENTS

Security and Privacy Requirements:

The system supports only one user and the data submitted by the user. It stores certain private user information such as email address, password, and budget usage. The private user data must not be accessible to anyone other than the logged in user. We want to create an application that can be used simply and conveniently when users visit the site and write down their income and expenses.

The application will run on a server that is accessible by the public internet, but user login data and personal information should be kept safe. Otherwise, other users can access it. Therefore, this sensitive data should be encrypted so that only the user can see it. We do not plan to track user data, and we think we need the ability to delete user data from the database if they no longer use the service.

Github will be used to keep track of security flaws that arise during the development and deployment of the app. By using Github, these types of issues can be assigned to specific members of the engineering team to be resolved. When a team member completes the task, they push their branch to the deployment branch to verify security compatibility. Team members can create and document new issues in the To-Do column if they find security flaws or bugs during project development. A severity level of important or critical will cause the development team to pause work on features in order to resolve the security vulnerability.

Quality Gates (or Bug Bars):

Privacy Bug Bars

Critical

- Lack of notice and consent
  - Example: storing sensitive information without notice
- Lack of user controls
  - Example: users do not have the ability to delete their user account and download collections.
- Lack of data protection
  - Example: sensitive information is not encrypted before being stored in collections.
- Improper use of cookies
  - Example: private information stored in a cookie is not encrypted

Moderate

- Lack of internal data management and control
  - Example: Admin cannot help users to recover their data after user delete accidentally, and we do not have a retention policy.

Low

- Lack of notice and consent
    - Example: sensitive information collected and stored locally as hidden metadata without notice. Sensitive information is not accessible by others.

<u>Security Bug Bars</u>

Critical(A security vulnerability that would be rated as having the highest potential for damage)

- Elevation of privilege
    - execution of arbitrary code

Important (A security vulnerability that would be rated as having significant potential for damage, but less than Critical)

- Information disclosure
    - An attacker can get any PII information such as income and email address that was not intended to be exposed.
- Spoofing
    - An attacker masquerades as a user to gain access to control the user account and get and modify any private information.

Moderate (A security vulnerability that would be rated as having moderate potential for damage, but less than Important)

- Denial of service
    - Server loss

Low (A security vulnerability that would be rated as having low potential for damage)

- Absence of encryption
    - Private data without encryption which makes sensitive data easily accessible.

Risk Assessment Plan for Security and Privacy:

How

UHBadger will store the user's email and password because it requires that information to make a user account. In addition, an email address as a username should be unique as a unique identifier(ID) for each user. User accounts only allow access when they enter a correct password. Users have to agree to our privacy policy to sign up successfully. In our Privacy policy page, we will tell users why we need that private information and how we use it, etc.... As we know, UHBadger is a money management and planning application, we will have to store the private information such as income and spendings, which are classified as sensitive information. We do not transfer and allow unauthorized access to sensitive information. Users do not have the ability to delete their account, but they can request an admin to delete their account through a verification process. However, the user can delete any information and even re-initialize the account to avoid their sensitive information being stored and known by us. Therefore, we cannot recover user information after sensitive information is deleted. In addition, The user can edit and remove their information. Users do not need to install new software, and we only store the information we need and are allowed by the user.

If UHBadger wants to add new features, we will discuss and evaluate new features. If users have questions about our web, feel free to contact us(our contact information is listed in footer) and we will reply as soon as possible to solve the problem and answer the questions or concerns. In general, we will do our best to keep users' sensitive information safe and make the user have a good experience.

Parts that require threat modeling

- Login feature
- Client interaction with database

DESIGN

Design Requirements:

UHBadger is an application focused on a very simple user interface and easy accessibility. Since UHBadger does not track user data, we will focus on security for many of the design requirements.

- Database

  User data can be stored in MongoDB to protect user data. Access to MongoDB is restricted to specific service codes on the server. All calls to access the database must go through this database service code. This is to minimize the number of places in server code that could open insecure connections to the database. Client access to the database shall run over protected API endpoints. Only authenticated requests shall be allowed to access data for a particular user.

- Project credential

  The credentials required to use a service such as a password should not be stored in the code file in Github. All user data must be encrypted. The credentials must use a trusted certificate and store the credentials in an environment file that is securely stored and referenced by the code.

- HTTPS

  To provide a secure environment without the application tracking user data, the application is implemented with HTTPS certificates. The application and server must only communicate via an encrypted connection such as TLS to prevent data from being intercepted.

Attack Surface Analysis and Reduction:

There are two privilege levels for our application, the levels show below, one for users and another one for the Admin. The Privileged user level where users need to create an account and sign in with a username which is email and the correct password. Admin can delete and edit user information after signing in.

01. Privilege Levels

a. User

    i. Send personal information/data through a form

    ii. Modify personal information/data through a form

b. Admin

    i. Modify information/data of a user

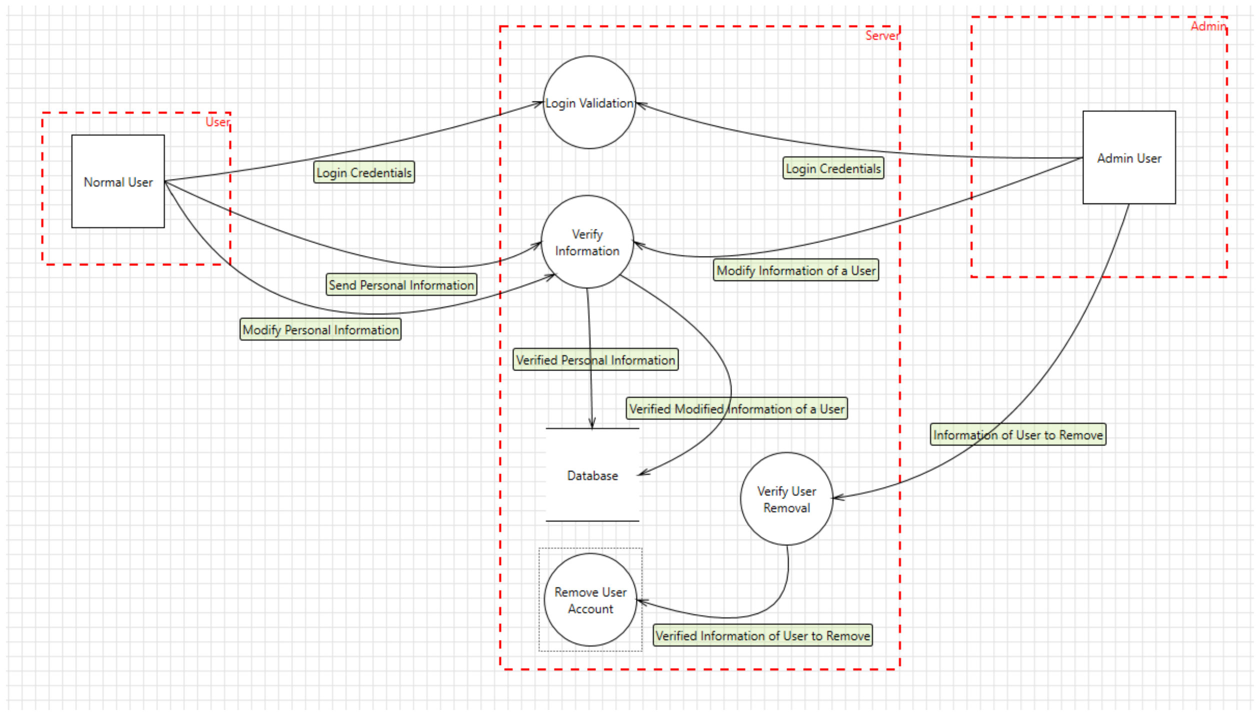
    ii. Remove a user account

There might be some hackers trying to attack this web application to get user private information.

Security vulnerabilities in Web applications can be as follows.

- Unprotected URL: user can specify URL to access page which does not apply ProtectedRoute to specify who allows access to the page.
- Authentication: hackers may pretend as a user to login in the application to acquire the user information, they may login in the application successfully by guessing the user passwords after a few times.
- Injections: hackers may find a flaw in the application where malicious code may be injected where data input is not very restrictive or not secure

Threat Modeling:


User
Normal User
Server
Login Validation
Verify Information
Database
Verify User Removal
Remove User Account
Admin
Admin User
Login Credentials
Login Credentials
Send Personal Information
Modify Personal Information
Modify Information of a User
Verified Personal Information
Verified Modified Information of a User
Information of User to Remove
Verified Information of User to Remove

Threats by Category

Spoofing:

- Spoofing of Destination Data Store Database
- Spoofing of Verify information Process
- Spoofing the login Validation Process
- Spoofing the Admin User External Entity
- Spoofing the Human User External Entity
- Spoofing the verify User Removal Process

Tampering:

- Potential Lack of Input Validation for Login Validation

- Potential Lack of Input Validation for Verify Information
- Potential Lack of Input Validation for Verify User Removal

Repudiation:

- Potential Data Repudiation by Login Validation
- Potential Data Repudiation by Verify Information
- Potential Data Repudiation by Verify User Removal

Information Disclosure:

- Weak Credential Storage
- Data Flow Sniffing

Denial of Service:

- Potential Excessive Resource Consumption for Verify Information or Database
- Potential Process Crash or Stop for Login Validation
- Data Flow Generic Data Flow Is Potentially Interrupted
- Potential Process Crash or Stop for Verify Information
- Data Flow Send Personal Information Is Potentially Interrupted
- Potential Excessive Resource Consumption for Verify Information or Database
- Potential Process Crash or Stop for Verify User Removal

Elevation of Privilege:

- Elevation Using impersonation
- Elevation by Changing the Execution Flow in Verify information
- Elevation by Changing the Execution Flow in Login Validation

- Login Validation May be Subject to Elevation of Privilege Using Remote Code Execution
- Verify Information May be Subject to Elevation of Privilege Using Remote Code Execution
- Elevation by Changing the Execution Flow in Verify User Removal
- Verify User Removal May be Subject to Elevation of Privilege Using Remote Code Execution

Implementation

Approved tools:

Tool	**Version**	**Comments**
React	17.0.2	
react-dom	17.0.2	
react-router	5.3.3	
react-router-dom	5.3.3	
semantic-UI-react	2.1.3	
Meteor	2.7.3	
Node	16.15.0	

Mongo	5.0.5	
IntelliJ IDEA	Most Recent Version	
ESLint	7.0.3	
sweetalert	2.1.2	
lodash	4.17.21	
uniforms	3.5.2	
prop-types	15.8.1	
simpl-schema	1.12.2	
Github Desktop	3.0.1	Optional

Deprecated/Unsafe Functions:

- semantic-UI-react
    - Ref

      Replaced by:

      React.forwardRef
- ESLint
    - Abstract equality

        - == && !=

      Replace by

        Strict equality (=== && !==)

- React-DOM
  - render

    Replaced by:

      createRoot

- Node
  - eval

    Replaced by:

      JSON.parse

- Mongo
  - Mongo.Collection#deny

    Replaced by:

      A custom method

  - Mongo.Collection#allow

    Replaced by:

      A custom method

  - db.collection.count

    Replaced by:

      db.collection.countDocuments or db.collection.estimatedDocumentCount

- db.collection.insert

  Replaced by:

  db.collection.insertOne or db.collection.insertMany

- db.collection.remove

  Replaced by:

  db.collection.deleteOne or db.collection.deleteMany

- db.collection.update

  Replaced by:

  db.collection.updateOne or db.collection.updateMany

- db.collection.mapReduce

  Replaced by:

  db.collection.aggregate

Static Analysis:

Tool	**Version**
ESLint	7.0.3
eslint-plugin-react	7.24.0
eslint-plugin-react-hooks	4.2.0
eslint-plugin-meteor	7.3.0

eslint-plugin-import	2.23.4
eslint-plugin-jsx-a11y	6.4.1

The static analysis tool we chose for this project is ESLint because we are familiar with it. It is the dominant tool for linting JavaScript code and is used to help programmers to maintain their code quality and do error catching. It analyzes the source code and identifies possible programming errors automatically in real-time when coding. It marks errors with the red wave underlines and warnings with yellow wave underlines. In addition, there are a couple of code style standards in ESLint, and ESLint rules can be customized. In other words, you can create your own custom ESLint rules. Plugins can expose additional rules(custom rules) for use in ESLint. Plugins are published as npm modules in the format of esling-plugin-<plugin name>. The examples are shown below:

- eslint-plugin-react
  - React specific linting rules for ESLint.
- eslint-plugin-react-hooks
  - Enforces the Rules of Hooks.
- eslint-plugin-meteor
  - Meteor specific linting rules for ESLint.
- eslint-plugin-import
  - Ensure import point to a file that can be solved.
  - Ensure named imports correspond to a name export in the remote file.
  - Esure a default export is present, give a default import.
  - Prevent misspelling of file paths and import names.

- eslint-plugin-jsx-ally
  - Static AST checker for accessibility rules on JSX elements.

Therefore, ESLint can help programmers to find style errors or problems while coding. It is very useful for team work because it helps us to keep a certain style in our code and thus avoiding confusing your teammates about your code.

<u>**Verification**</u>

Dynamic Analysis

We use TestCafe to perform dynamic analysis on the application in the form of end-to-end testing. TestCafe is an open-source testing framework designed to work with JavaScript applications and can be used on a variety of platforms. TestCafe has a built-in function to check if the application UI is stable and is implemented in the template we use, making it easy to integrate into UHBadger. Therefore, we can quickly test all the features within our application through TestCafe. After implementing a new feature or updating the code, run a test to ensure that the application is not experiencing errors. TestCafe runs automatically when one of the branches of the team members is merged into the main branch, which helps our application stay safe and error-free.

The advantage of end-to-end testing is that the entire application is tested, from the server to the front-end UI. TestCafe works by opening a web browser (e.g., Google Chrome) and remotely controlling it to simulate clicks and determine the shape of a specific node in the DOM. With TestCafe, we can ensure that applications continue to work correctly after changes and that servers and clients continue to work together. One disadvantage of implementing TestCafe is that it requires all DOM elements to have a unique 'id' tag, which is an additional step for the front-end developer to perform. Additionally, the time taken to test is also limited by the speed at which the browser can render the DOM. On the other hand, an integration test does not have a bottleneck of waiting for the DOM to render, but cannot confirm that the UI elements have been rendered to the page. Therefore, end-to-end testing from TestCafe provides important coverage in that regard. The next step in our dynamic analysis coverage will be the addition of integration tests. We hope to have close to 100% coverage of our code.

Attack Surface Review

There have been no changes, updates, patches to the approved tools since the submission of Assignment 2. In addition, no vulnerabilities were found. In general, nothing has changed and no vulnerability was reported this week.

Approved tools

Tools	**Version**	**Changes/Updates, Patches..etc**	**Vulnerabilities Found? Y/N**	**Comments**
React	17.0.2	NO	N	N/A
react-dom	17.0.2	NO	N	N/A
react-router	5.3.3	NO	N	N/A
react-router-dom	5.3.3	NO	N	N/A
semantic-UI-react	2.1.3	NO	N	N/A
Meteor	2.7.3	NO	N	N/A
Node	16.15.0	NO	N	N/A

Mongo	5.0.5	NO	N	N/A
IntelliJ IDEA	Most Recent Version	NO	N	N/A
ESLint	7.0.3	NO	N	N/A
sweetalert	2.1.2	NO	N	N/A
lodash	4.17.21	NO	N	N/A
uniforms	3.5.2	NO	N	N/A
prop-types	15.8.1	NO	N	N/A
simpl-schema	1.12.2	NO	N	N/A
Github Desktop	3.0.1	NO	N	Optional

Fuzz Testing

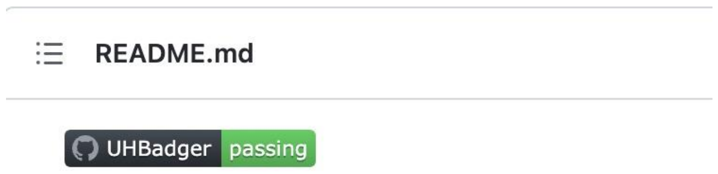
One of the common places to attack a website that has accounts is the login page. We attempted to break into the website with the method of creating a new account with an email that has been used before. The purpose of going through this method is to see if the website would allow multiple accounts with the same email as well as see if it would make it so that the multiple accounts would have access to the same information. What was done was that we first went to the sign up page and tried to make a new account with an email we know has already been used. In doing so, the website itself has shown confirmation of the creation of the account, but the account was not actually made within the website, only within the collection of user profiles. Even though a new entry of a user profile was added, you are not able to login with the “new” account. Why this is happening from our understanding is that we never added a proper error checking for account creation for the sign up page. To help explain, Meteor, a framework that is used for this website, has its own way of keeping track of accounts for the website. This is separate from our collection of user accounts that we keep track of with our defined code. So, when the account creation process went through, Meteor did not accept the new account and failed to make a new entry, but our side of the code still went through even though Meteor rejected the creation and generated its own account entry. This could have implications for a different type of vulnerability or bug in relation to the accounts. So, it should be planned for us to work on error checking when going through account creation on the sign up page.

Another method of attacking the login page is through brute force. As the name of the method may imply, the method has the user try to get access to an account through trial and error of different combinations of emails and passwords, potentially taking a long time to break into. Eventually, an account can be broken into if the person trying to break into an account puts enough time and resources. When we performed brute force, the rate and speed of the success fell solely on password strength and other features that can slow down the method. Since our password requirements are loose, a user can have a password as simple as a single letter or number. This could make finding the correct password for an account very easy if someone was using a tool that goes through a dictionary of passwords. Besides the weak password requirements, the website's way of slowing down brute force attacks are not great. While brute forcing an account, you will eventually hit a cooldown timer when attempting to login too many times and the login fails. Currently, the timer is 10 seconds when there are multiple login attempts that failed. Even though this does slow down the attacker from continuously trying to brute force the credentials, this may not be enough of a deterrent for some. What can be done to improve the security of the website from brute force attacks is to employ more intensive password requirements as well as add more to the penalty of multiple failed login attempts. For example, adding a password length requirement to 8 characters minimum can greatly reduce the speed a brute force attack can figure out your password. As for the penalty, increasing the timer to 30 minutes can be a much bigger deterrent to the brute force attack. Not only increasing the timer, but when an account gets set to a timer for failed login attempts, it can be flagged and notify the owner of the account through an email that there were multiple failed login attempts and can ask for a password reset or temporarily lock your account.

Besides attacking the login page, another method of breaking the website is through attempted access of URLs that are not protected. The purpose of performing this test is to see if someone can potentially access another user's information. How this test was done was by logging into one user account and trying to edit their spending information. When accessing the page to edit their spending information, the URL is unique with the specific spending information you are editing. Someone can possibly get that unique URL and try to access it with a different account to get access to the information. In an attempt to try to access the unique URL when not logged in with the appropriate user, the access fails with a blank page. This shows that the website has enough security to not reveal information to outside users that do not own the information. Why this is secure is because information related to a user can only be accessed when they are currently logged in on the active client. If the user is not logged in, then the information will not be published unless you are an admin. To get into more details, when accessing unique URLs for editing spending information, the page will try to subscribe to the collection of all spending information for the currently logged in user. So, if you are a different user, when trying to access the page, you will only be subscribing to spending information that is connected to your account only, not other users.

Static Analysis Review

Our team has been using ESLint to do static analysis. It will show red wave lines when we have ESLint errors, and it can help us to fix it. As we know, static analysis is the testing and evaluation of an application by examining the code without running the application. Thus, we can easily find errors with ESLint while coding. If we find an error, we will first stop coding and fix it rather than continue coding. In addition, we run the linter to test code by using **meteor npm run lint** command in the terminal to double check that we don't have any errors before pushing the commit to our Github repository, which is called UHBadger. If we pass static analysis and dynamic analysis, we can see that our project status is shown as passing in the README file. So far, we do not have any errors, and our project looks good from our static analysis review.


README.md

UHBadger passing

Dynamic Analysis Review

We are using TestCafe for dynamic analysis and have expanded the test range. We have verified that testing for basic pages such as landing, registration, login, and logout works properly. We also added testing for the budget planning and budget spending pages required for our application. We have added functions that allow users to manage their budgets and we are using TestCafe to make sure that the updated functions are working properly. There are also plans to implement a page where administrators can manage user accounts and create tests for them. We took some time to increase the test coverage for the code to 100%, and so far, there have been no problems with TestCafe. As a result of running a test on our project, we found that our project is working as intended because it passed all the tests.

Release

Incident Response Plan

A Privacy Escalation Team:

- Escalation Manager(Hansen Cabanero)
    - Know the validity of the incident and situation
    - Analyze the issue and decide if we need to fix it immediately
    - Make a plan to solve the issues
    - Assign tasks to our teammates
    - Make a summary of the known facts
- Legal Representative(Feimei Chen)

    - Responsible for consistently helping resolve any legal issues throughout the process

- Public Relations Representative(Cheolhoon Choi)
    - Listen to users complaints and comments
    - Report unsolved problems to the management
    - Teach users how to use our applications in-person or through media
- Security Engineer(Feimei Chen)
    - Find and solve vulnerabilities
    - Respond to security incidents
    - Implement and test security features

Emergency Contact Email:

- Email address A(choi4879@hawaii.edu)
    - Contact with Cheolhoon Choi
- Email address B(feimei@hawaii.edu)
    - Contact with Feimei Chen
- Email address C(hansenca@hawaii.edu)
    - Contact with Hansen Cabanero

Please don't hesitate to contact us if you have any questions or concerns.

Procedures: (A set of procedures that our team will go through each time there is some kind of incident)

1. Once the issue is identified and involves sensitive user information, our public relations representatives will let all users know that the issue is occurring and

remind them to protect their information by changing their passwords or even deleting their user accounts if necessary, and we will resolve the issue immediately.

2. The escalation manager will analyze the issue and make a plan to solve the issue. If there are more than one issue happening or reported at the same time, the escalation manager will decide which issue needs to be fixed first based on the severity of the issue, then assign tasks to each of our teammates.
3. Each teammate completes their assigned tasks according to the plan. For example, the security engineer needs to fix the vulnerabilities and do static and dynamic analysis to ensure that the issue is solved.
4. If we allow user information to be attacked successfully, our legal representative will communicate with the victims and the court on behalf of our organization.
5. If the issue is solved, our public relations representatives will make our users know this good news immediately to release their concerns.
6. The escalation manager will summarize the known facts.

Final Security Review

Looking back at the initial threat model that was made at the design section of this document, it has not changed since then as it still reflects what our software should have its interactions be doing. As for the static analysis, running the **meteor npm run lint** command shows no important errors coming from our code and ensures that the quality of our code is up to standard. Since the last review of our dynamic analysis, extra tests have been implemented for making sure the pages and components of our website are working properly. The result of running the test on our project shows that everything is working properly.

Going over the quality gates that were established in the requirements section of the document, there is one critical problem in regards to privacy for our website. The problem is the lack of user control, where the user is not able to delete their account or download their collection as an example. The ability for the user to delete their account is available, but the ability to download their own collection is currently not an option. Besides the critical problem for the privacy of our users, there is also one important security problem. Spoofing is a problem for the security of our users, in particular the brute force approach in trying to gain access to an account. Currently, the measures to deal with a brute force approach are not that significant like the penalty for multiple failed login attempts being only ten seconds or no password restrictions.

As seen going through the final security review of our website, we determined that the grade should be a **Passed FSR with exceptions**. The decision to give the website this grade instead of a **Passed FSR** is because of the problems that were explained in the review like the lack of user control for downloading collections or the spoofing problem with the brute force method. These problems are the exceptions of the grade and are to be logged and fixed before the next release. Every other aspect of the SDL requirements our team has laid out has been met and this led to the **Passed FSR** portion of our grade.

Certified Release & Archive Report

The released version of UHBadger is available here:
https://github.com/UHBadger/UHBadger/releases/tag/v1.0.0

UHBadger is a money management and planning application that allows users and administrator accounts. It tracks and analyzes user spending to help optimize user spending. Users can record all expenditures and plan their desired expenditures. UHBadger helps users spend more than they earn by displaying their budget and all expenditures. UHBadger also calculates the proportion of each expenditure area, such as food and transportation. A user can freely sign up with the user account and the administrator can remove user accounts. This is version 1.0.0 of our application and we would like to add the ability to search for specific spending events so that users can easily search for them in their plans in a future version.

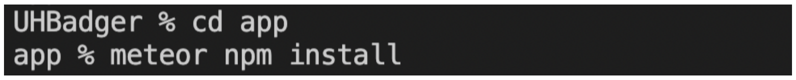
Technical Notes

Installation

First, install Meteor.

Second, download UHBadger and request permission to gain access to UHBadger.

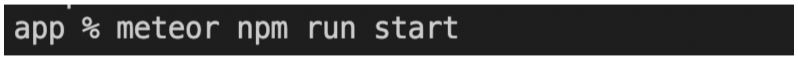
Third, cd into the app/ directory and install required libraries: meteor:



```
UHBadger % cd app
app % meteor npm install
```

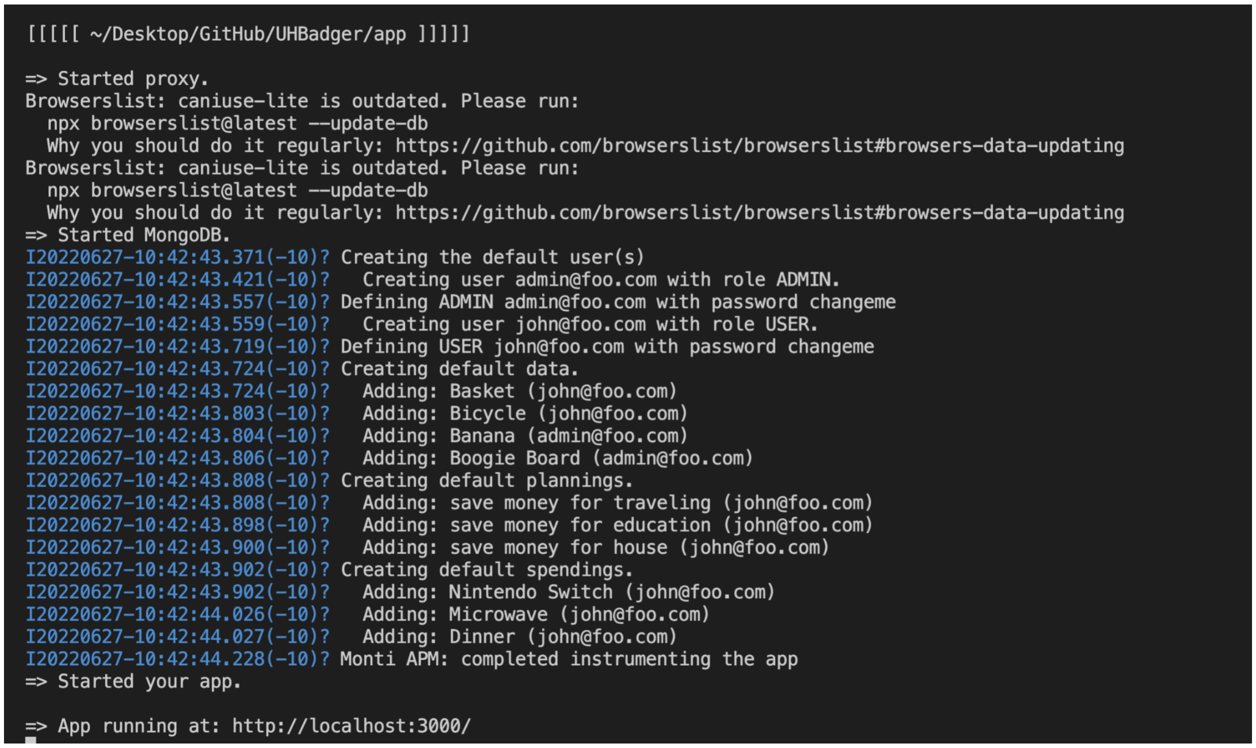
Running the Application

After installation, you can run the application by typing:



```
app % meteor npm run start
```

Running the application for the first time will add default users:



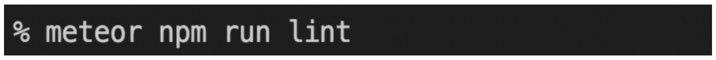
```
[[[[[ ~/Desktop/GitHub/UHBadger/app ]]]]]

=> Started proxy.
Browserslist: caniuse-lite is outdated. Please run:
  npx browserslist@latest --update-db
  Why you should do it regularly: https://github.com/browserslist/browserslist#browsers-data-updating
Browserslist: caniuse-lite is outdated. Please run:
  npx browserslist@latest --update-db
  Why you should do it regularly: https://github.com/browserslist/browserslist#browsers-data-updating
=> Started MongoDB.
I20220627-10:42:43.371(-10)? Creating the default user(s)
I20220627-10:42:43.421(-10)?   Creating user admin@foo.com with role ADMIN.
I20220627-10:42:43.557(-10)? Defining ADMIN admin@foo.com with password changeme
I20220627-10:42:43.559(-10)?   Creating user john@foo.com with role USER.
I20220627-10:42:43.719(-10)? Defining USER john@foo.com with password changeme
I20220627-10:42:43.724(-10)? Creating default data.
I20220627-10:42:43.724(-10)?   Adding: Basket (john@foo.com)
I20220627-10:42:43.803(-10)?   Adding: Bicycle (john@foo.com)
I20220627-10:42:43.804(-10)?   Adding: Banana (admin@foo.com)
I20220627-10:42:43.806(-10)?   Adding: Boogie Board (admin@foo.com)
I20220627-10:42:43.808(-10)? Creating default plannings.
I20220627-10:42:43.808(-10)?   Adding: save money for traveling (john@foo.com)
I20220627-10:42:43.898(-10)?   Adding: save money for education (john@foo.com)
I20220627-10:42:43.900(-10)?   Adding: save money for house (john@foo.com)
I20220627-10:42:43.902(-10)? Creating default spendings.
I20220627-10:42:43.902(-10)?   Adding: Nintendo Switch (john@foo.com)
I20220627-10:42:44.026(-10)?   Adding: Microwave (john@foo.com)
I20220627-10:42:44.027(-10)?   Adding: Dinner (john@foo.com)
I20220627-10:42:44.228(-10)? Monti APM: completed instrumenting the app
=> Started your app.

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

Viewing the running app locally

If everything goes well, the template application appears at http://localhost:3000.

ES Lint

We can run ESLint to verify that our code complies with coding standards.



```
% meteor npm run lint
```

Links

- GitHub Repository (https://github.com/UHBadger/UHBadger)
- README (https://github.com/UHBadger/UHBadger/blob/main/README.md)
- Wiki Page (https://github.com/UHBadger/UHBadger/wiki)
- Release (https://github.com/UHBadger/UHBadger/releases/tag/v1.0.0)