Travlr Getaways

Software Design Document

Version 1.0.2

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	05/21/24	Matthew Pool	Updated Table of Contents, Executive Summary, and Design Constraints, and added a UML Component Diagram
1.0.1	06/09/24	Matthew Pool	Created UML Sequence Diagram and UML Class Diagram and incorporated API Endpoints
1.0.2	06/19/24	Matthew Pool	Completed User Interface section

Executive Summary

Travlr Getaways needs a web application to provide its customers with an easy and efficient way to learn and plan their next vacation. The *Travlr Getaways* app is designed to do just that! The software architecture will use a customer-facing side accessed via a proposed URL from any browser and a single page application (SPA) for the administrator account(s). Images, texts, and links will be used throughout the application to provide an enjoyable user experience.

Software development will include the **Model-View-Controller** (MVC) design pattern in conjunction with the **MEAN stack** (MongoDB, Express.js, Angular, Node.js). MVC provides a separation of concerns to provide a secure and robust, easy to maintain and reuse, architecture. The MEAN stack can be used to develop the entire web application and only utilizes the JavaScript (JS) (and TypeScript) language, providing easier development. **JSON** (**JS Object Notation**) is a standardized format for data interchange and will be used in our web app. **Node.js** can power real-time applications and manage many simultaneous connections to provide greater scalability, as the number of users grows. **MongoDB** is a database system offering high flexibility and scalability and is suitable for data-intensive applications like this one. **Express.js** is a flexible Node.js web app framework providing a robust set of features for web and mobile apps. **Angular** is a front-end framework by Google offering a powerful toolset for building web apps with rich user interfaces.

The **user landing page** of our web app will stand out by displaying a high-resolution, surreal photo image of a beach or other breath-taking imagery. The page will give crucial information, as well as blog post links and testimonials from other satisfied customers. Using Angular, the user site will utilize **two-way data binding**, which puts HTML together based on provided data and immediately updates HTML if the HTML or data changes. This dynamic data provides the user with a user interface (UI) that reacts to any changes the user may make. The header and footer will include (redundant) links as follows:

- **Home**: landing page providing various pieces of information and imagery
- **Travel**: displays popular or suggested trip locations, including a title, description, and image for each trip
- **Rooms**: shows available rooms to rent, including pricing, title, description, and image for each
- **Meals**: displays different eating options, including title, description, and image for each item
- **News**: various relevant information, including testimonials, etc.
- **About**: various information about *Travlr Getaways*
- **Contact**: *Travlr Getaways* address, phone/fax number, and contact form

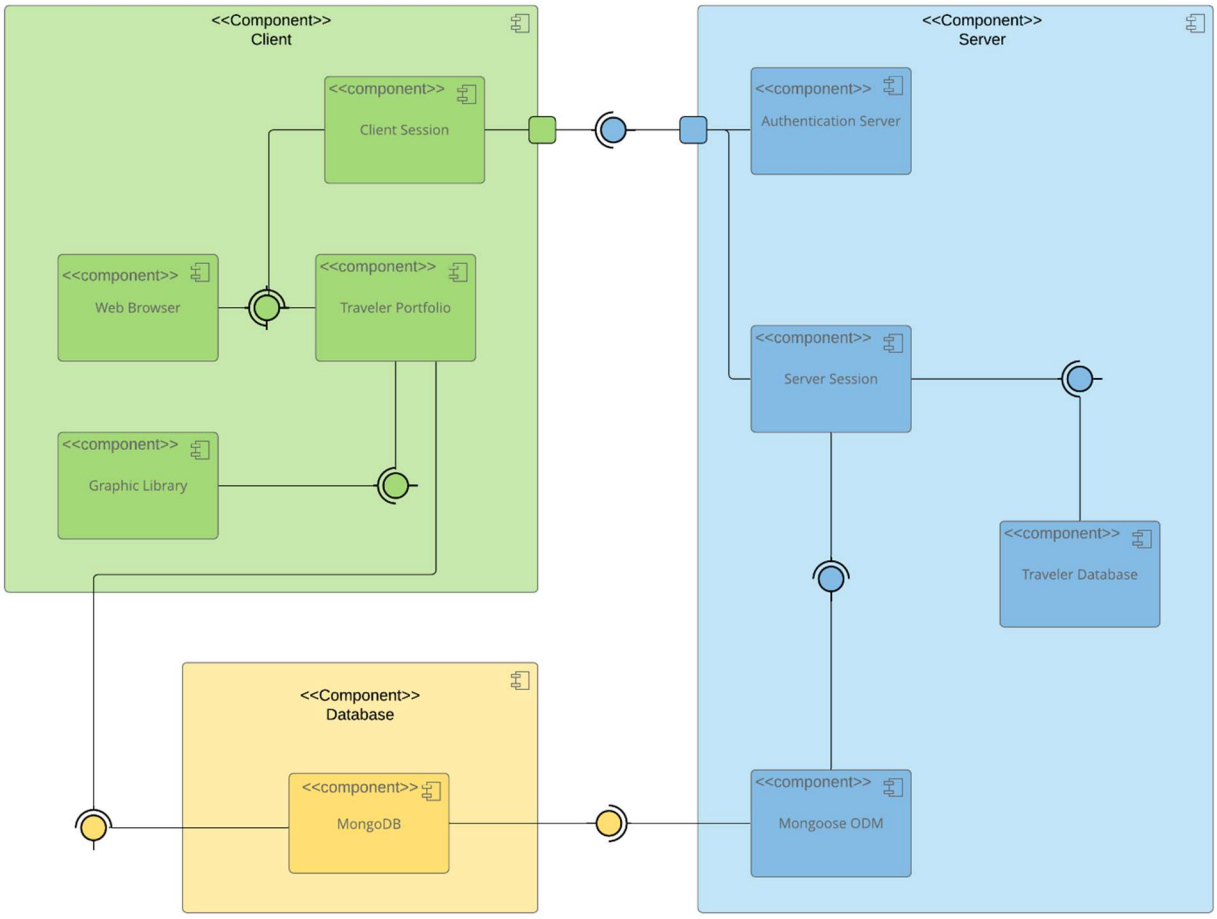
The **administrator page** will consist of a **single page application** (**SPA**) to provide the admin with an easy-to-use user interface to manage user and site data. The SPA runs in the admin's web browser and never fully reloads the page, providing a fast and efficient UI. All app logic, data processing, user flow, and template delivery can be managed. The server only has to pass data from the database and serve static files. These technologies and frameworks will result in an optimized, maintainable, scalable, secure, and robust *Travlr Getaways* web application.

Design Constraints

- **scalability**: must support increasing number of users and data
    - MongoDB (scalable database), Node.js (handle numerous simultaneous connections), and Angular (responsive UI)
- **performance**: fast and responsive interactions for users, minimum latency, caching/CDNs (content delivery networks), Angular two-way binding
- **security**: user data protection, unauthorized/unauthenticated access prevention, modern data encryption and password hashing, secure communication (HTTPS), input validation and sanitization, static/dynamic testing, safe error-handling
- **user experience (UX)**: intuitive, visually-appealing, efficient, responsive UI (Angular for dynamic components), localized static content for users
- **maintainability**: easy to maintain, debug, and update (MVC/MEAN), documentation
- **compatibility***:* must be compatible across various web browsers and devices/platforms
- **availability**: minimum downtime, redundancy, load balancing, strategic maintenance/updates
- **resources**: must work within budget and time limitations and use an Agile methodology with Scrum and XP (extreme programming) frameworks for quick, iterative, test-driven development to ensure the main functionality/requirements are met within the resource constraints provided
- **compliancy**: application must comply with relevant legal and regulatory requirements and follow industry best standards practice such as the following:
    - GDPR (General Data Protection Regulation)
    - PCI DSS (Payment Card Industry Data Security Standard)
    - ADA (Americans with Disabilities Act), FTC (Federal Trade Commission)
    - WCAG (Web Content Accessibility Guidelines)
    - SOX (Sarbanes-Oxley Act)
    - ISO (International Organization for Standardization) 2700
    - NIST (National Institute of Standards and Technology)
    - OWASP (Open Web Application Security Project)
    - FISMA (Federal Information Security Management Act)

System Architecture View

Component Diagram



The **Client component** consists of the following parts:

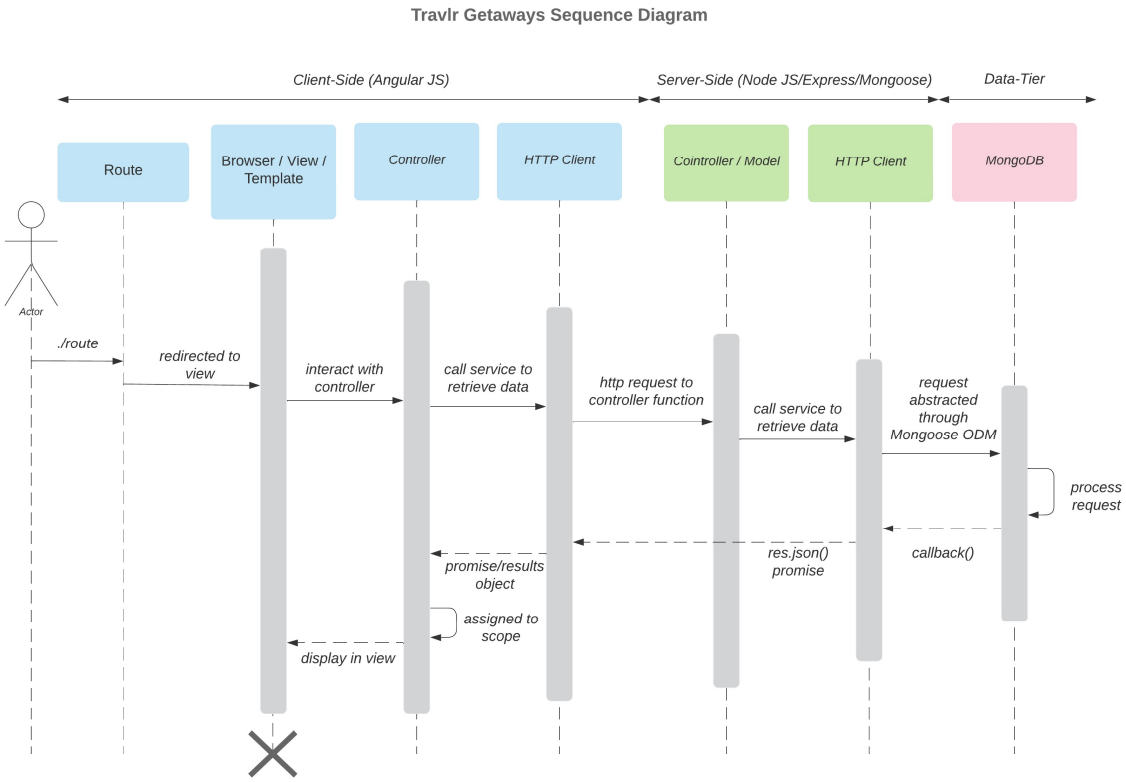
- **Client Session**: manages user session on client side (requires interface from Authentication Server for a distinct interaction (port) and interface from Web Browser/Traveler Portfolio)
- **Traveler Portfolio**: user data (requires MongoDB interface)
- **Web Browser**: interface for user to interact with app (requires Traveler Portfolio interface)
- **Graphic Library**: graphical functionality (requires Traveler Portfolio interface)

The **Server component** will provide much of the app's logic and functionality and will include the following parts:

- **Authentication Server**: manages user authentication and authorization
- **Server Session**: manages user session on the server side (requires Traveler Database and Mongoose ODM)
- **Traveler Database**: handles data storage and retrieval
- **Mongoose ODM**: facilitates interactions between server and MongoDB database in the

The **Database component** is used for data storage/retrieval (requires MongoDB interface)

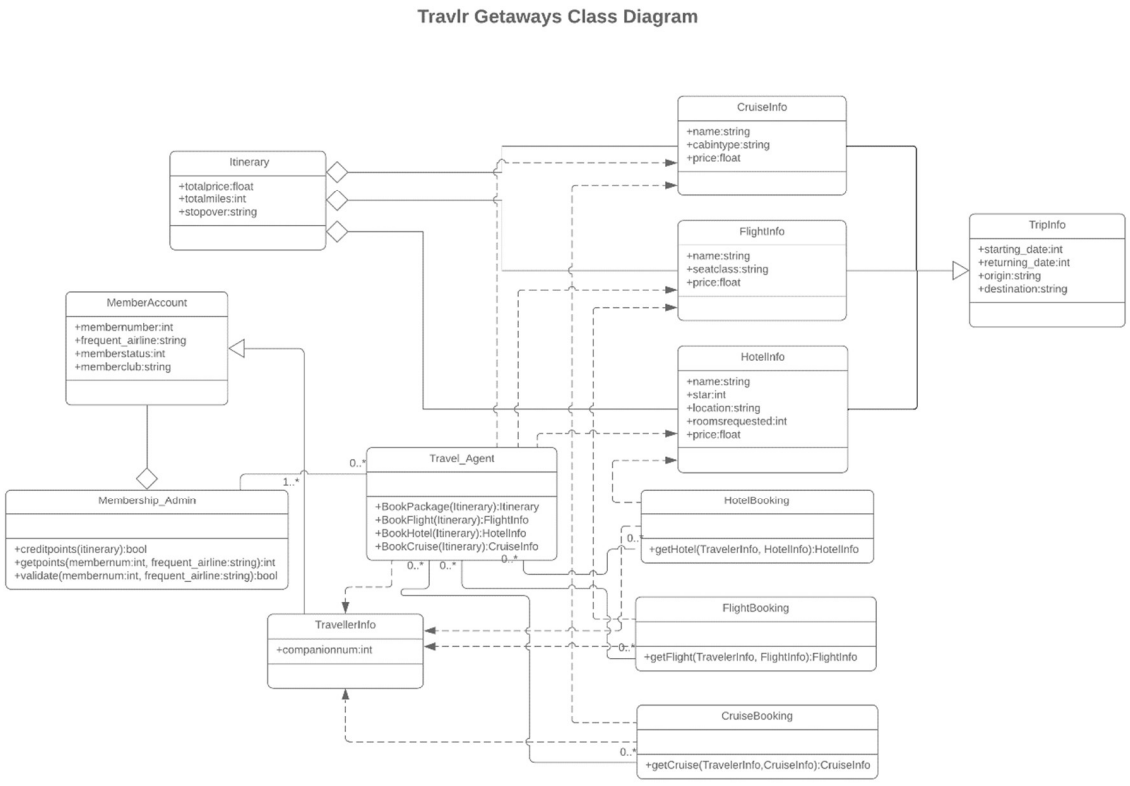
Sequence Diagram



A user (actor) kicks the sequence off by using a web browser to navigate using a specified route. The router redirects the request to the appropriate view (using Handlebars templating) and displays the view inside the user's browser. User interaction with the displayed view triggers a function in the AngularJS controller. The API controller calls a service that uses the HTTP client and fetch() function to make an HTTP request to the server-side (Node.js and Express.js). The server-side controller/model processes the request and calls a service using Mongoose ODM to query the MongoDB database. MongoDB processes the request and returns the data via a callback (.then) function and responds back to the server-side HTTP client with a res.json() promise. This is then resolved by the AngularJS service and assigned to a scope variable in the client-side controller. The controller then updates the view with the returned data, and the user sees the update. When the browser is closed, the sequence is ended.

For example, a customer uses their web browser to navigate to *Travlr Getaways* web site, and the router redirects the browser's request to show the appropriate home page view. The user then clicks the 'Travel' tab in the header, and this interaction triggers a function in the AngularJS (client-side) controller, which ultimately uses the server-side controller/model to request JSON (BSON) data from MongoDB. Finally, after sending this data back to the client-side controller, the controller updates the view, and the user sees the updated page in their web browser.

Class Diagram



According to the UML Class Diagram, there may be zero-to-many Travel_Agent instances for each Membership_Admin instance, and there may be one-to-many Membership_Admin instances for each Travel_Agent instance. The Membership_Admin class provides a way to manage member points for each MemberAccount instance, which contains membership attributes only. The Travel_Agent class contains methods to book trips and packages by passing an Itinerary-type parameter and using the companionnum attribute from the TravelerInfo class. A Travel_Agent instance may book a hotel, flight, cruise, or any combination of these, using the corresponding classes and associated methods. The Itinerary class contains the trip's total price, total miles, and stopover string. The 'Booking' classes use the corresponding 'Info' classes, which contain properties like name and price, along with the TravelerInfo class, which shows the start and stop times/locations. Lines with a hollow diamond indicate an aggregation association representing a part and the whole (diamond end) in which the part can exist independently of the whole. Lines with a hollow arrowhead indicate a child class inheriting attributes/methods of the parent class (arrow end). The '+' symbol is an access modifier that indicates public access to the associated attributes and methods.

API Endpoints

Method	Purpose	URL	Notes
POST	Register new account	/api/register	Returns success/failure status code
POST	Logs user into account	/api/login	Returns session token
GET	Retrieves a list of trips	/api/trips	Returns all active trips
POST	Creates a new trip	/api/trips	Returns new tripcode
GET	Retrieves a specific trip	/api/trips/:tripCode	Returns a single trip instance (identified by the tripCode passed on the request URL)
PUT	Updates a specific trip	/api/trips/:tripCode	Returns success/failure status code (identified by the tripCode passed on the request URL)
DELETE	Deletes a specific trip	/api/trips/:tripCode	Returns success/failure status code (identified by the tripCode passed on the request URL)
GET	Retrieves a list of meals	/api/meals	Returns all available meals
GET	Retrieves a specific meal	/api/meals/:mealCode	Returns a single trip instance (identified by the mealCode passed on the request URL)
GET	Retrieves a list of news	/api/meals	Returns all available meals
GET	Retrieves a specific news article	/api/meals/:mealCode	Returns a single trip instance (identified by the mealCode passed on the request URL)
GET	Retrieves a list of rooms	/api/rooms	Returns all available rooms
GET	Retrieves a specific room	/api/meals/:roomCode	Returns a single room instance (identified by the roomCode passed on the request URL)
POST	Creates (sends) new contact message from user	/api/contact	Returns success/failure status code

The User Interface

Travlr Getaways Admin!

Add Trip

Gale Reef

Emerald Bay, 3 stars

4 nights / 5 days only $799.00 per person

Gale Reef: Sed et augue lorem. In sit amet placerat arcu. Mauris volutpat ipsum ac justo mollis vel vestibulum orci gravida. Vestibulum sit amet porttitor odio. Nulla facilisi. Fusce at pretium felis.

Sed consequat libero ut turpis venenatis ut aliquam risus semper. Etiam convallis mi vel risus pretium sodales. Etiam nunc lorem ullamcorper vitae laoreet.

Edit Trip

Dawson's Reef

Blue Lagoon, 4 stars

4 nights / 5 days only $1,199.00 per person

Dawson's Reef: Integer magna leo, posuere et dignissim vitae, porttitor at odio. Pellentesque a metus nec magna placerat volutpat. Nunc nisi mi, elementum sit amet aliquet quis, tristique quis nisl. Curabitur odio lacus, blandit ut hendrerit

vulputate, vulputate at est. Morbi aliquet viverra metus eu consectetur. In lorem dui, elementum sit amet convallis ac, tincidunt vel sapien.

Edit Trip

Clair's Reef

Coral Sands, 5 stars

4 nights / 5 days only $1,999.00 per person

Clair's Reef: Donec sed felis risus. Nulla facilisi. Donec a orci tellus, et auctor odio. Fusce ac orci nibh, quis semper arcu. Cras orci neque, euismod et accumsan ac, sagittis molestie lorem. Proin odio sapien, elementum at tempor non.

Vulputate eget libero. In hac habitasse platea dictumst. Integer purus justo, egestas eu consectetur eu, cursus in tortor. Quisque nec nunc ac mi ultrices iaculis.

Edit Trip

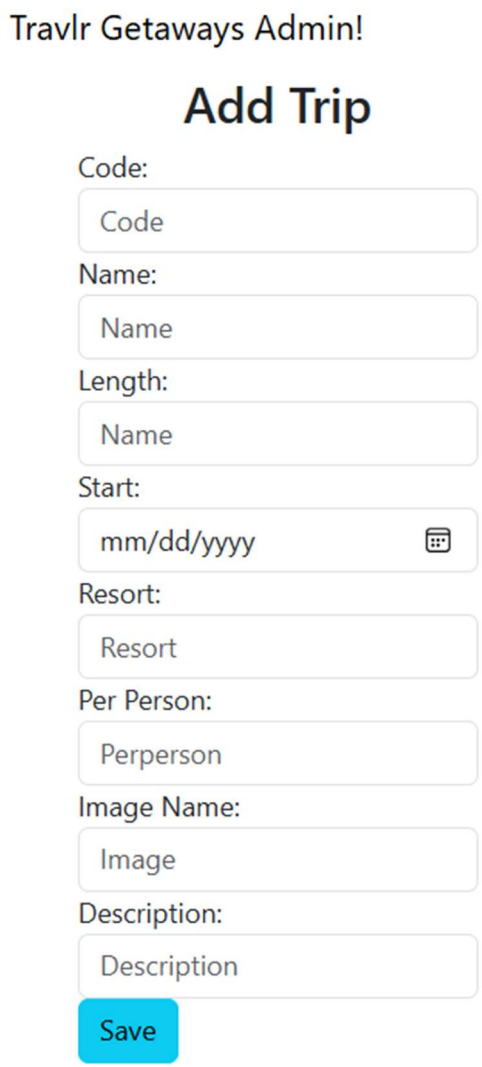
Most Bodacious Mega Reef

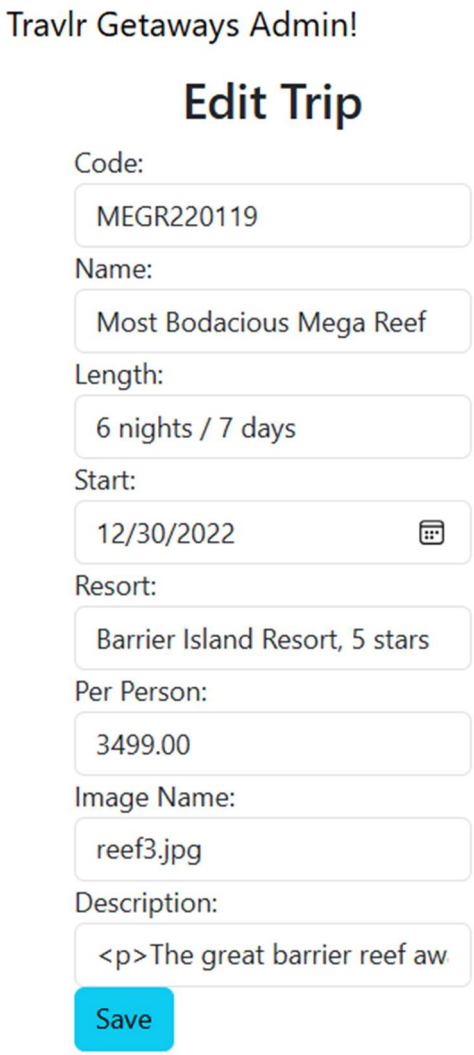
Barrier Island Resort, 5 stars

6 nights / 7 days only $3,499.00 per person

The great barrier reef awaits!

Edit Trip





Travlr Getaways Web App Architecture

The admin site for *Travlr Getaways* consists of a client-based single-page application (SPA) web page that uses *Bootstrap CSS* and *Angular*, which provides reusable UI components and logic. The architecture, functionality, and testing of the SPA site can be better understood by the following information.

Angular vs Express Framework

The customer-facing site consists of Node.js used in conjunction with the *Express* framework, which consists of HTML pages served from (and rendered in) the server, which utilizes JavaScript. The *Express* router manages server-side routing, while client-side routing is managed by *RouterModule*. Middleware is used for request handling, HTTP responses, and other functionality.

The *Angular*-based admin-facing website consists of a modular architecture that includes routes, modules, services, and components. Components consist of HTML, CSS, and TypeScript files. TypeScript is a JavaScript superset that includes type safety to help catch data type errors at compile time (Baeldung, 2021).

Single-Page Application (SPA)

A SPA is a type of web page that loads a single document, and then updates the body content via JS APIs like *Fetch* (MDN Web Docs, n.d.). SPAs provide fast and responsive pages due to client-side caching and the fact that the page doesn't need fully-reloaded. This results in increased performance and better user experience. However, since the page renders client-side, it is challenging to optimize for search engines (SEO). Since the entire app is downloaded to the user's device, there may be an increased initial load time.

Testing the SPA

SPAs, like most other web applications, have CRUD (Create, Read, Update, Delete) functionality to interact with a database, like *MongoDB* (accessed via *Mongoose*). *Postman* is an app commonly used to test RESTful APIs and web pages using HTTP methods like POST, GET, PUT, and DELETE (Postman, n.d.). After starting your web server and opening *Postman*, you can select the HTTP GET (Read) method type from the drop-down, input the API endpoint (URL) of the resource, and click the "Send" button. *Postman* displays the returned status code and (JSON, HTML, etc.) data retrieved from the (*MongoDB*) database. To test the PUT (Create) method, follow the same steps as the GET method requires, after entering in any number of key-value pairs into the "Query Params" section. If there are errors, a status code indicating an error, as well as debugging text, will appear instead. *Postman* also shows the time and size of the HTTP response. You may also create variables and JavaScript test scripts, if desired. You can verify the database documents by using a program like *MongoDB Compass*, without loading any web pages.