Simple Note-Taking API (December 2017)

Cade P. Gillem

***Abstract*—This simple note-taking app is a demonstration of a REST API in action. It was implemented in the Go programming language, which includes a built-in HTTP library. The frontend is implemented in HTML and JavaScript. These two portions of the application work together to form an interactive application that stores simple text-based notes.**

I. Used Technologies

A. Go and Libraries

Go [1] is a systems language created by Google. It was designed to be efficient, compiling quickly while still performing garbage collection. Its syntax is unique, as well. This combination of factors made it a tempting language to put to use for this project, and it succeeded.

The following libraries were used in this project, aside from what comes with a standard Go installation:

- mux: A router package from the Gorilla toolkit. [2]
- jwt-go: An implementation of JSON Web Tokens used for authentication. [3]
- go-jwt-middleware: A package that helps with authentication on protected routes. [4]
- negroni: A middleware library used in tandem with go-jwt-middleware. [5]
- go-sql-driver/mysql: An implementation of a database driver for use with MySQL. [6]

B. MySQL

MySQL [7] is a widely used relational database management system. It was used to store the two main data tables for this project, "users" and "notes." It is accessed through the go-sql-driver/mysql package.

C. JQuery

JQuery [8] is a JavaScript library that streamlines frontend code. Through CSS-style queries, the frontend may manipulate the document in order to display changes to resources immediately. In addition, JQuery provides functionality for sending HTTP requests to the backend. This means that notes can be created, opened, updated, and deleted without navigating away from the same page in the browser.

D. Bootstrap

The frontend of the app was designed with Bootstrap [9], a CSS framework. This framework provides effortless styling and responsiveness, allowing it to perform well on mobile browsers as well as desktop ones.

II. Software Architecture

A. Database Interface

To begin with, the project uses a separate application within its main package to manage the database schema. This is the "db" subdirectory. Upon building the entire project, a binary will be generated within this directory that allows for simple command-line manipulation of the database. Its core functions are database setup, teardown, and seeding.

Next, a file called "db_helpers.go" is provided for the rest of the program in order to set up, tear down, or seed the database. The "db" subdirectory uses this, as well as the tests. Functions within this file make direct queries to the database. In addition to the core functionality, there are other helper functions for purposes such as checking whether a stored password hash matches with a given password.

The "resource.go" file allows for a higher-level interface to the database by generalizing the properties of resources such as users and notes. A user, a note, or any other resource added in the future of development may make use of the base Resource struct within. This struct provides methods for selecting data from the database, syncing it to the database, and deleting data from it.

Finally, there are model structs that allow database interaction on the highest level. Found in "user.go" and "note.go," these structs encapsulate a Resource struct within them and hold the data fields of the individual models as well. They make calls to their underlying Resource struct when data needs to be changed in the database.

B. API Backend

The API itself is divided into the files "note_handlers.go" and "user_handlers.go," and HTTP requests are directed to them through "routes.go." All API routes are behind the "/api/" sub-URL. The functions within these two files return handler functions. Those handler functions deal with all the error handling, existence checking, and authenticating, before manipulating or returning any data.

Another file that is necessary for the API is "json_response.go." This contains a struct which stores various fields of data that the client will be able to use, such as the models retrieved, input validation messages, and error messages. When filled, this struct offers a method which converts it to a JSON string and writes it to the given response writer. The response writer is a Go struct used for sending HTTP responses.

C. Responsive Frontend

The frontend of the application, which serves as a client to the REST API, is stored entirely within the "public" directory. The Go backend serves this directory as a file server, meaning its contents are available to client browsers. The chief files within this directory are "index.html" and "app.js."

In "app.js," which is run as a script at the end of the main webpage, all unnecessary HTML elements are hidden and a list of notes is shown to the user, provided they are logged in. This list of notes is not included with the webpage; the script loads this data in after the document is loaded. In this process, action buttons are appended to each note's table row. These buttons get their own unique event handlers for managing their

assigned notes.

The frontend provides two other views, which are a note editor and a note viewer. They are initially hidden, but can be navigated to by clicking an action button on a note row. Upon navigation, they are cleared and refilled with any particular note's information, namely its title and content. The note editor has a dual purpose, depending on whether the user wished to edit an existing note or create a new one. Data validation is performed in either case and error messages are displayed below the form controls.

D. Authentication

Authentication is done through JSON Web Tokens, or JWTs [10]. These tokens are JavaScript object strings that store a user's basic data and a signature. This signature is checked by the backend every time the user navigates to a protected route. The protected routes within this application are prefixed by "/api/." To access these routes, the client needs to include one of these JWTs in the "Authorization" header of its HTTP requests.

III. Testing Suite

Tests for this project are written using Go's default testing framework. It is straightforward, and only a few features were needed in addition to its core. A few extra functions are found in "test_helpers.go." These functions streamline assertions necessary for comparing expected data with test results, as well as database setup and teardown.

The tests themselves are found in "resource_test.go" and "models_test.go." They provide coverage of the database interaction layer within the project. A mock model struct which makes use of the Resource struct is used in order to provide as much isolation as possible. This mock model is not the same as a user or note model, therefore testing code for the Resource struct itself does not need to concern itself with differences between the user and note structs.

IV. Security

The decision to use JSON Web Tokens was made in order to comply with REST standards as much as possible. The philosophy behind the decision is statelessness: the desire to avoid storing any more data on the server than necessary [11].

An important note on security must be made, however. Due to the nature of JWTs, SSL encryption is necessary on any server this project would be deployed on. Without encrypting the network traffic, packet sniffers such as Wireshark would conceivably be able to look at the authorization header of an HTTP request and copy the token along with its signature.

References

[1] 'The Go Programming Language'. [Online]. Available: http://golang.org. [Accessed: 6- Dec- 2017].
[2] 'mux - Gorilla Web Toolkit'. [Online]. Available: http://www.gorillatoolkit.org/pkg/mux. [Accessed: 6- Dec- 2017].
[3] 'dgrijalva/jwt-go: Golang implementation of JSON Web Tokens (JWT)', 2017. [Online]. Available: https://github.com/dgrijalva/jwt-go. [Accessed: 6- Dec- 2017].
[4] 'auth0/go-jwt-middleware: A Middleware for Go Programming Language to check for JWTs on HTTP requests', 2017. [Online]. Available: https://github.com/auth0/go-jwt-middleware. [Accessed: 6- Dec- 2017].
[5] 'urfave/negroni: Idiomatic HTTP Middleware for Golang', 2017. [Online]. Available: https://github.com/urfave/negroni. [Accessed: 6- Dec- 2017].
[6] 'go-sql-driver/mysql: Go MySQL Driver is a MySQL driver for Go's (golang) database/sql package', 2017. [Online]. Available: https://github.com/Go-SQL-Driver/MySQL/. [Accessed: 6- Dec- 2017].
[7] Oracle Corporation and/or its affiliates, 'MySQL', 2017. [Online]. Available: http://mysql.com. [Accessed: 6- Dec- 2017].
[8] The JQuery Foundation, 'JQuery', 2017. [Online]. Available: http://jquery.com. [Accessed: 6- Dec- 2017].
[9] 'Bootstrap'. [Online]. Available: http://getbootstrap.com. [Accessed: 6- Dec- 2017].
[10] 'JSON Web Tokens'. [Online]. Available: http://jwt.io. [Accessed: 6- Dec- 2017].
[11] RESTfulAPI.net, 'What is REST'. [Online]. Available: https://restfulapi.net/. [Accessed: 6- Dec- 2017].