ECE 563 – Spring 2017

Project Report - Knowt

Alexander Tseng(at200), Jay Wang (zw48)

April 23, 2017

Contents

1 Overview

1.1 Motivation

"In order to KNOW everything, we need to take notes."

-Alexander Tseng and Jay

For most people, it is impossible to remember every single details in your life. These information can range from sophisticated equations taught in lectures to simple reminders such as To-Do lists. However, regardless of the information one might write down, we can all agree that taking notes undoubtedly plays an important role in our daily lives.

In the past, people tend to take notes on portable notebooks using pens or use sticky notes. As technology advances, people begin to use to their mobile phones more and more often. As a result, numerous note taking applications are created. In the modern society, smartphones and laptops have become inseparable with us. With the technology of Cloud computing, the art of note taking has been brought to the next level: we can now access our notes anywhere, anytime, and without the worry of losing our notes.

Surprisingly, we only found a few note-taking apps that are both lightweight enough to be easy-to-use, and able to be accessed anywhere. Also from time to time, we want to share our notes to our friends just so they know what we are up to. Take Apples Note app as an example - it is indeed a light app in iOS device and Mac OS device, but it fails to exist on a Windows machine. Thus, we want to build our own version of note-taking web app, which focuses on its lightness, simplicity, and user-friendliness.

1.2 Product

The final product can be accessed here: `knowt.pro`

Functionality overview:

- User Authentication
  - Login and regiseration (include username, email, password validitation)
  - Net ID login
- Note Taking
  - Notes are created with title and context
  - Supports markdown format
  - Draggable
  - Edit and Delete
- Sharing
  - Uses autocomplete to complete the textfield when the user types in a username
  - Notes can be shared with "view" or "edit" option

  - Notes can be unshared; if the owner deletes the note, the note will be deleted from other shared users

- View
  - Notes can be viewed in the following catagories: Notes created by user (my notes), notes shared with user (notes shared with me), and all notes.
  - The application is well routed (Landing page, login, registeration, and the app itself).

2 Tech Stack

The table below summarize the technologies we picked for this project.

Focus	**Technology**
VPS + DNS	DigitalOcean
Reverse Proxy	NGINX
Process Manager	PM2
Email Server	Mailgun
Front-end	React + Material-UI
Back-end	Node.js + Express
Database	MySQL
ORM	Sequelize
API	REST
Editor	Atom
Linter	ESLint

3 Approaches and Evaluations

In this section, we provide detailed explanations of the technical approaches we took to complete this project. We also included the evaluations of each approach and technique chosen.

3.1 Deployment

We actually took a very uncommon path this time - before we had any code written (except for the basic code to initialize a node server), we decided to focus on the deployment of the server in the beginning. Since neither of us had any experience on setting up a server, we believed it would be a great chance to learn something new and valuable, which turned out was indeed.

3.1.1 Host Server

After spending some time on several Virtual Host, including Duke VM, DigitalOcean and AWS, we decided to go with DigitalOcean. The reason we didn't choose Duke VM is that we actually wanted this project to go beyond the lifespan of this class. Thus to save the mess of migrating, we limited us to only DigitalOcean and AWS. This concept of 'Droplet' was what finally won us. With your credit card typed in and 5 dollars per month, it provided us with this ready-to-go 20G spaces and 512M virtual machine running Ubuntu 16.04. After a lot of beginners' guide and tutorial and Stackoverflow, we successfully made our node server up and running, able serve a simple GET request. We also configured NGINX as a reverse proxy to redirect the visit to our IP address http://67.205.162.151/ to the port 8080 which the server listens on.

3.1.2 MySQL Server

We had a discussion on whether we want to use a BaaS (Backend-as-a-Service) like Parse or Firebase which would for sure simplify the backend development and data storage, or to build our own storage layer and start the backend from scratch. It may sound very silly, but this was indeed what we thought - we CANNOT pay 5 dollars a month just to store our HTML and CSS! Also, we already had a server - what's the point of having a server if its purpose was solely to call another BaaS? Thus, we chose to write our own back-end, and host our own MySQL server on our VM.

3.1.3 Process Manager

We used PM2, which is an advanced process manager for Node.js application. It tremendously simplified the deployment, able to auto restart the server on crash or machine start-up.

3.1.4 Improvements

Looking back, I believe there are some improvements that we can do. Firstly as mentioned, our MySQL server is hosted on the same VM as our server. This is less than ideal - a separate MySQL server with backup (or even geo-replicated) would definitely be more preferable. Since our project is highly data centered, users would not want to lost their notes just because one server was toasted. Secondly, we are currently using HTTP. As will be mentioned again in Authentication section, this will pose major risk on users' first sign in, because under HTTP, username and password are sent without encryption. Also when client later performs API call with the auth token, the token is subject to man-in-the-middle attack and getting stolen. It is definitely important for us to look into the setup of HTTPS and the installation of SSL certificate if we are ever to publish our product to the public.

3.2 Development

3.2.1 Version Control

We used Git as our cersion control and created our own repository on GitHub. We have two main branches, master and dev. Master always represents a stable version of our product and is the branch we used for deployment, and dev should be a relatively stable version that, if tested to be good for a while, will be cut a release into master. Other features should be logically separated into each feature branch.In order for each feature branch to be merged into our product, we used Pull Requests which were reviewed by another person. We set our master and dev branches to be protected, so no code could be directly merged into them without a request.

3.2.2 Scheduling

We used milestones on GitHub with concrete due dates, along with fine-grained issues under milestones as our To-Do lists and to track our progress throughout the semester.

3.2.3 Code Style

For the first week or two, we did not use a style guide or linter system to ensure consistent code styling. We paid for this in many ways. Because of the nature of JavaScript, it is a common mistake that you override some variables already defined, or a undefined misspelled variable which gives you no error. Also, React PropTypes were defined vaguely and not enforced. For example, we did not specify what type of object a React prop was supposed to be, which made it difficult when using that particular prop. Furthermore, we had a mixture of ES6 and ES5 syntax which results in less readability and consistency.

After some research, we added a linter (ESLint) to our development process. We adapted AirBnb's Javscript style guide and now enforce strict stylistic conventions on our codebase. We invested some time in going back through our old code and refactoring it to adhere to our new style guide. It effectively transformed our Atom editor to an IDE-like environment, largely increased the code consistency and development speed, and decreased the syntax mistakes made.

3.3 REST API

We designed our app so that every operation is done using HTTP request with carefully defined REST API. The comprehensive API documentation can be found here: `https://quip.com/48tcAaSUKgha`. It specifies the HTTP action, endpoint, parameters, and JSON response schema for each REST API. We have been updating it in a timely manner, making sure the other person is aware of the API changes. We also used Postman as our API tester to test and debug.

The main reason we chose to define our API as RESTful is that it is an application-layer protocol on top of TCP. We do not have to worry about retransmission, congestion control, reordering packets by sequence number, coalescing fragmented packets, etc. We also get serialization for free. It is by definition exactly-once semantics so the system is safe under non-idempotent operations. It is also language and framework independent so if ever we decided to implement part of our services in a different framework, nothing on the API and above needs any change. For example, our current product is a web-app, but we can easily create the mobile version to interact with our server, with only the need of building the front-end.

3.4 Authentication

Since our product requires user sign-up and login, authentication is needed by its nature. After googling, we decided to use JSON Web Token (JWT) and bcrypt to facilitate our user authentication and single sign-on feature. When a user is created, the password is **salted** and **hashed** using bcrypt, and only the salt and hash are stored in our database. When a user successfully logs in using their credentials, a JWT will be generated and returned to client. The client could then use this token to prove its identity. The tokens are signed by the server's private key, so only the server is able to decrypt the token, verify its validity and read out the information of this logged-in user (in our case, userId). Since JWT is designed to be compact and URL-safe, it is light-weighted enough to be used for every request - it is required as an Authorization Header for the requests to all of our endpoints starting with **/api**.

Though using JWT has many advantages and facilitated the development of our authentication system, we do think there is a caveat. The expiration time is encoded within the JWT token, and for our product it is set to a week. However at the same time, this indicates that if a user's token was stolen by an attacker, the attacker would have essentially become the user, and can perform any operation just as the victim. Currently, our system does not have an efficient way to invalidate one single JWT token without invalidating all of the existing tokens.

3.5 Client

3.5.1 React/ Material UI

React was chosen as the frontend framework for the project. React focuses heavily on the V(view) of the MVC model. It is a framework that allows programmers to write pure Javascript that updates React Components, and React will update the DOM for the programmer. In otherwords, the data binding is not interweaved with the application. It is a frontend rich application that is extremely modular and component based. It is very easy to add and switch out components as well as UI mockup. Since it is open source, there are numerous plug ins out there such as the draggable functionality and markdown support.

In terms of the frontend UI library, we chose Google's Material UI. Combining Google's Material Design and React, Material UI is a frontend library that provides an extremely clean and sleek interface.

3.5.2 React Router

React Router is an npm package created by reacttraining.com. It is a collection of navigational components that compose declaratively with the application. Even though our application is a single page application, react router allows ease of rendering different components based on the specified route. However, the documentation is pretty poor and there were constant bugs. We actually spent a great deal of time just to find the right tutorial to set up the router because most tutorials online are outdated. First, they changed the name from react-router to react-router-dom. Then, they implemented new syntax such as HashRouter instead of Router. Nevertheless, after the router is set up, it is extremely easy to navigate through pages.

3.5.3 Main Components

We can break the application down to several components:

- app.js
  - Sets up the router that routes to Landing Page, Login Page, and Register Page.
  - It is the main container (entry point) for the whole application. When the user accessed the application, the routher will first automatically direct the user to the Landing page.
- LoginPage.js
  - Contains the user authentication logic. If the user is not logged in, then the application will render the page with the login form. If the user is logged in, then the application will render the app.
- KnowtApp.js
  - Contains the main application. The components within the app can be further broken down such as: header, left menu, note items, etc.

3.6 Accessor

Accessor, or access layer, is a client-side library we wrote for ourselves. It is used by View to facilitate the access to our back-end logic, abstracting out the actual API calls to the server. Every operation that involves the retrieving, creation, update or deletion of the data has one or more corresponding functions in the accessor. Under the hood, the accessor makes the asynchronous API calls, and return the result once it receives the response from the server. The result will always indicate the status as 'success' or 'error', along with the data or error information. A simple example would be the situation where the View needs to display all the notes belonging to the current user. By calling 'const response = await noteAccessor.getAllNotes()', the accessor will internally make a GET HTTP request to the endpoint configured and wait for the reply.

The reason we decided to write this client library is because that it exists as a layer of abstraction, making clear and less complicated the communication with our back-end. Exposed in a front-end friendly way, the accessor abstracts out the actual API calls to the endpoints and construct the API call parameters (path, header and payload) in a safe way known to the back-end developer. Two examples of accessor would be `createNoteAsync` and `shareNoteWithUserAsync(noteId, userId, permission)`. Corresponding accessor functions were written by the same developers who wrote the feature, which ensures the awareness of the expected API path, query parameters and payload format.

3.7 Email

Initially we did not think of adding email support. However, when we were implementing the sharing functionality, we felt the necessity to have email notification when a user receives a shared note - after all, users would certainly need to know when a note is shared with him/her.

After some research, we were faced with two choices - to build our own SMTP server, or to use existing email service. Lucky for us, we found the perfect choice - Mailgun. Mailgun is a transactional Email API Service which allows ten thousand emails per month for free. With its concise API and detailed documentations, we were able to add the email notification functionality to our app without too much trouble.

3.8 Storage

As mentioned above, we hosted our own MySQL server as our storage layer. Though Jay had some previous experience with SQL, we did not want to write raw SQL which is both messy and less-readable, and not safe and prone to SQL injection. After some research, we learned the idea of ORM, Object-relational mapping, which serves as an interface for SQL. There are honestly so many ORMs that we could choose from for Node.js environment, and we finally went with Sequelize which was one of the several choices that Google told us.

However in retrospective, we do wish we could have invested more time on choosing the ORM. The ORM we are using, Sequelize, is good in many aspects including its perfect support for creation, update, delete and more importantly, database transactions. However, it was later revealed to be not ideal for table JOIN. One application of JOIN operation is our sharing functionality - ideally, in order to get the notes that are shared with a user along with the sharer's information, we should join the sharing table (which simply pairs up the userId and the noteId) with the notes content table and the user table. I can imagine the cleanness of achieving this query using raw SQL - however, Sequelize did a very poor job on JOIN, and it turned out we need to perform some application layer, linearized manual join, which imaginably increases the latency and slows down the response speed.

4 Future Work

An application is never ”finished”. In fact, every application in the world right now is a work in progress because there is always room for improvement. One improvement we can make is to change from HTTP to HTTPS to make our website more secure.

In order to take advantage of the modularity of React, we can refactor our code to even smaller components, making it more modualized. Further functionalities can be added to the application. Here are a few examples: user settings, user appearance customization, notes variety such as to-do lists or reminders, allow keyboard shortcuts such as undo/redo, upload different kind of files (img, zip, docx, etc), and trash can. Last but not least, real time edit. Currently, after editing, sharing, and deleting, the user will need to refresh to update their notes. In the future, we hope that the notes can be updated in real time to save the hassle of constant refresh.

In the process of creating Knowt, we have learned new cloud and other technologies along the way. However, this project is just a small stepping stone for us towards the unlimiited potential of the cloud industry.