University of Regina | Faculty of Engineering and Applied Science

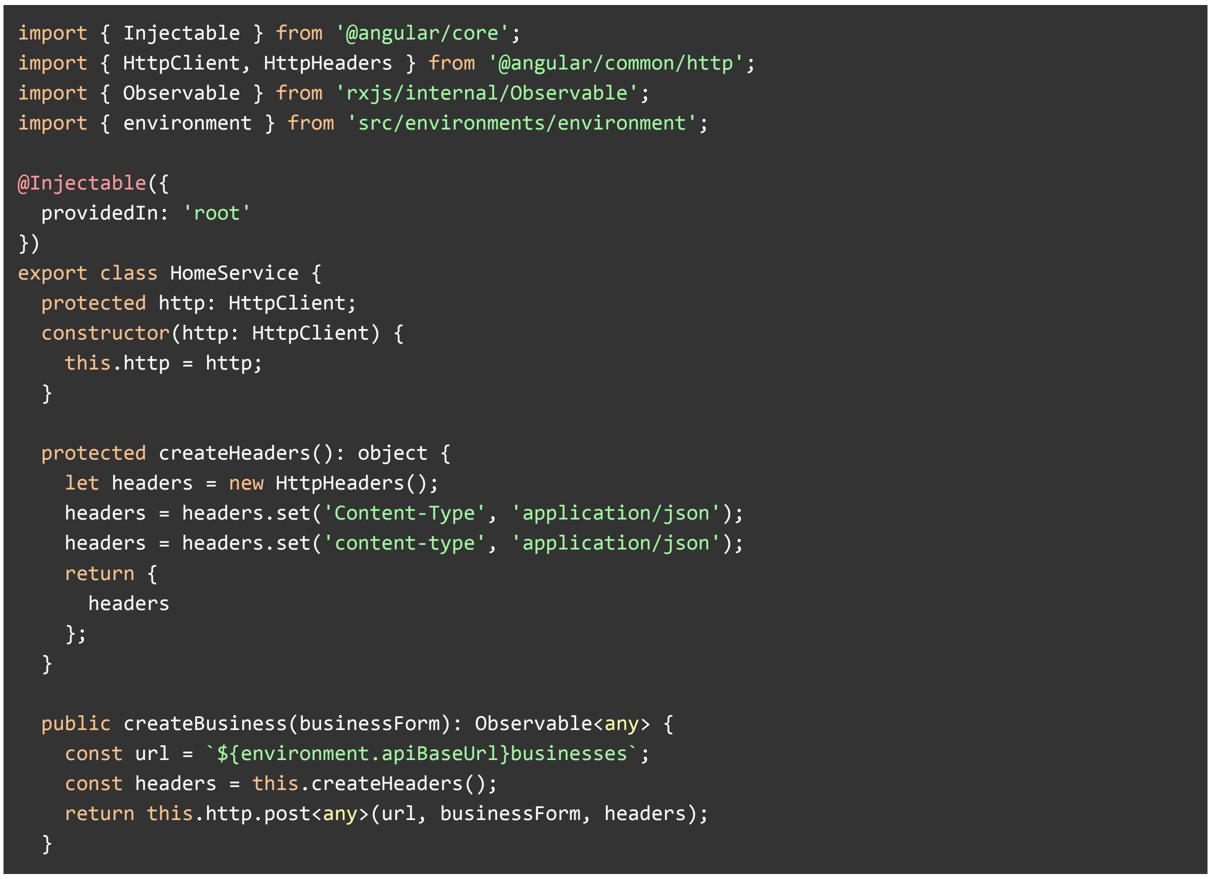
Software Systems Engineering Code Quality Review Report

Pencil.

Paul Hewitt & Ian Quach
200343079 & 200367522

Table of Contents

1. Code Formatting



```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs/internal/Observable';
import { environment } from 'src/environments/environment';

@Injectable({
  providedIn: 'root'
})
export class HomeService {
  protected http: HttpClient;
  constructor(http: HttpClient) {
    this.http = http;
  }

  protected createHeaders(): object {
    let headers = new HttpHeaders();
    headers = headers.set('Content-Type', 'application/json');
    headers = headers.set('content-type', 'application/json');
    return {
      headers
    };
  }

  public createBusiness(businessForm): Observable<any> {
    const url = `${environment.apiBaseUrl}businesses`;
    const headers = this.createHeaders();
    return this.http.post<any>(url, businessForm, headers);
  }
```

Fig 1 - Code snippet from home.service.ts

Pencil adheres to strict formatting standards for our entire code base. Our frontend code is written in TypeScript, with our backend written in Go. With any code collaboration, it is extremely important that all contributing members are adhering to the same style, and syntax guidelines. This makes sure that all code looks and reads identical, regardless of who wrote it. Pencil uses a code linter to help ensure that its codebase is tidy, and syntactically correct.TSLint is used for the frontend TypeScript code, to ensure that Pencil's formatting standards are met. For example, Pencil uses tabs, not spaces. Proper indentation is crucial for readability, as it clearly defines when code blocks start and end. For import statements seen at the top of Fig 1, a single white space is inserted on either side of the variable being imported. These standards were defined before programming began on Pencil, and TSLint helped to maintain them. For our commits to GitHub, we followed commitizen's rules, ensuring each commit had a clear message indicating the type of change, and the scope of the change.

Pencil uses the camelCase naming convention for everything, including methods and variables. As seen in Fig 1, the methods createHeaders and createBusiness both adhere to this standard. Once again, this is to both maintain consistency, and readability. The exception to this is for modules, components, and decorators, in which we use PascalCase.

This is standard for Angular projects, and is what the official documentation recommends for styling your codebase. Ultimately, it is up to the developers to pick something that works, as long as it is consistent throughout the whole codebase.

Another feature of TSLint is the limiting of line length to 140 characters. This way, horizontal scrolling will never be necessary when reading Pencil's codebase. For those tricky lines that need to be longer than 140 characters, they can almost always still be compiled with a line break placed inside of them. Finally with TypeScript, you have the choice to include semicolons at the end of each line. Pencil uses strict linting to ensure that each line ends in a semicolon, and will not allow the code to compile if a semicolon is missing. These are just some of the ways Pencil ensures its whole codebase is formatted properly, and up to industry standards.

2. Architecture

2.1 Separation and Encapsulation

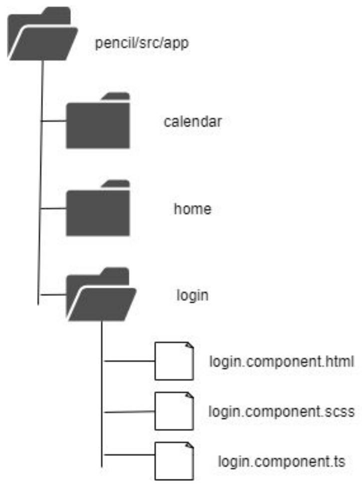


Fig 2 - Pencil folder structure

Due to the nature of Angular, everything is separated into components. This provides Pencil with separation of all files into their respective component folder. As seen in Fig 2, our HTML, TypeScript, and SCSS files are all separated, and they live in the proper parent folder. Pencil also separates into the traditional layers, such as Presentation, and Data layers. It does this using services. Components should not fetch or save data directly, they should just present data, and allow services to handle the data. These services are injected into each component as needed. This dramatically improves the performance, as well as the security of the overall web application. Pencil uses environment files to securely encapsulate away sensitive things such as API keys, and our REST API routes. These things should almost never be exposed, and by tucking them away in environment files, Pencil maintains application security, from the backend to the frontend.

2.2 Tech Stack

Pencil uses an industry standard tech stack. Not only does this prepare us for the real world, but it ensures best practices, and security. Deprecated tech typically has security vulnerabilities, so it is important to use what is current in industry. Pencil's frontend is built on Angular 8, which was the most current version available at the beginning of Capstone class in Fall 2019. The most recent version of TypeScript is also used, which is version 3.8.3 at the time of writing. The backend uses Go, which is Google's language similar to C. Go was the third most desired language in the Stack Overflow Developer Survey in 2019. Finally, our REST API is hosted on AWS using the Serverless Architecture. Again, this is industry standard, and AWS is one of the most powerful cloud computing platforms available. Pencil utilizes API Gateway, Lambda, DynamoDB, S3, and CloudWatch Logs on AWS.

2.3 Design Patterns

Pencil utilizes the MVVM design pattern. As discussed earlier, our services handle all the data, allowing them to be the Model of our MVVM design pattern. Pencil has four services, Business, Home, and User. As for the View, this is composed of the html and scss files of each component. These two files are what the user sees, and interacts with, making them the View. Our last files of each component are the TypeScript files. These make up the ViewModels. The TypeScript files provide methods and variables that are available to the View.

Our REST API uses the Services architecture. Each Lambda function handles the related data model. Each of these functions are mapped to a single route, with different endpoints representing the different CRUD operations, like GET, or POST. Using Services as opposed to MicroServices gives us less Lambda functions to worry about, and in turn increases Pencil's performance. This also allows us to deploy our Lambda Functions faster as opposed to MicroServices. The one drawback to using Services is it makes debugging slightly harder, as each Lambda function handles different functionality.

3. Non Functional Requirements

3.1 Maintainability & Reusability

Pencil takes pride in its readability. All methods and variables are named appropriately, and the purpose of almost all variables and methods can be identified by the name alone. The codebase itself has test files ready for each component, so testing is quite easy. Each component contains a .spec.ts file, where all tests will be written, ensuring each component is working as it should. The use of services is super handy too, as they almost double as interfaces. These services are extremely useful, and used by almost every component. Every method has been written for a reason, and Pencil tries to recycle methods and variables as often as possible. As Pencil is a web application, the best way to debug is using Developer Tools on your favorite browser. These tools are extremely powerful, and allow users to see the exact flow of control as the application executes.

3.2 Performance, Scalability, Security

We tried to keep Pencil as fast and snappy as possible. All assets that are served have been heavily compressed beforehand in order to minimize load times. Another way we keep Pencil fast is manipulating the DOM as little as possible. DOM bindings can be extremely expensive, so it is of the utmost importance to keep these bindings down. Thanks to RxJS Observables, Pencil is able to execute all of its HTTP calls asynchronously. This means the application will not hang up when it is waiting to hear back from the backend. As it waits for the observables to arrive, the application is free to spend its computing power elsewhere. Each one of our HTTP methods return an observable, which is then subscribed to by the calling component. All of the data Pencil sends and receives to the backend is in JSON, which is the industry standard in sending and receiving information over the web. This format is very lightweight, and is able to be parsed quite quickly, ensuring that any filtering done on the frontend can be completed in a timely manner.

Using AWS and the Serverless framework, Pencil's backend is able to scale very quickly under heavy load. Each one of our REST API's routes are actually an AWS Lambda function. Whenever an API call is made, the Lambda function quickly spins up, and connects to the required database. These Lambda functions will exist until they are not hit for roughly 30-45 minutes, in which they spin down. Each Lambda function is able to initially handle 1000 concurrent requests, with that limit increasing by 500 additional requests each minute. This means that if 1000 people all tried to create an account on Pencil at the exact same time, our backend would be able to handle it. Granted, we would be left with a fairly hefty AWS bill. The one potential choke point may be our database. If Pencil was operating at scale, it would have multiple databases for different regions, demographics, etc. The frontend should have no issue scaling at all, except for maybe applying too much pressure on the host.

As far as security is concerned, Pencil uses Facebook for user authentication, and AWS for data storage. Both of these platforms are very secure, and it would take something extraordinary to bring either of these titans of industry down. Anything sensitive that is currently on the frontend, such as API keys, or API routes, are located in an environment file. This is secure, but not impenetrable. In the future, it would be better to get these keys from the backend asynchronously, or utilize the browser's localStorage or sessionStorage. If Pencil was handling extremely sensitive data, the best option would be to utilize Docker and a Kubernetes cluster.

4. Usability

Pencil uses SASS for styling (similar to CSS), as well as Bootstrap. Bootstrap allows us to easily, and rapidly design responsive layouts that can work on screens of all sizes. As Pencil is a web application, it has the added benefit of working on any device, as long as it has a web browser. This way, our users are not limited by a platform like iOS or Android. However, there is some benefit to making dedicated apps for each platform, and that would be something exciting to explore in the future. We chose a fairly neutral color scheme, and tested it with grayscale filters to ensure that no readability was lost with the lack of color. The neutral colors help with readability, and makes sure that those that are colorblind can still differentiate the components of our application. Pencil makes use of certain affordances and signifiers in order to make call to action elements obvious. Our cards are slightly raised, and change the mouse pointer when you hover over them. This makes it more apparent that they are clickable. On our login screen, the login button is a different color from the background, and clearly says “Login with Facebook”, so users will not be alarmed when the Facebook modal opens up. In order to book an appointment, a user simply taps on the calendar in the time slot they may want, which feels natural and intuitive.

Unfortunately, our user testing was unable to fully be implemented due to the Covid-19 Pandemic. We were however, still able to get some of the people in our households to interact with, and give feedback on the application. One of the testers was quite tech illiterate, so she made the perfect user to test the application. She complained about the small font size, and she found the video background slightly distracting. Changes were made to remedy these issues. Font sizes were increased, and the video was trimmed, and slightly slowed down. It would have been nice to get users from all types of backgrounds to use, and interact with Pencil, but the extraordinary measures put in place due to Covid-19 prevented that.

5. Object-Oriented Analysis and Design Principles

Pencil aims to meet all the standards required for object oriented programming. We use different objects for all of our data sets. Businesses, Users, Appointments, etc. all have their own unique objects. These objects can be parsed to and from JSON to keep them lightweight when communicating with the backend. Just about all of our methods follow the Single Responsibility Principle (SRS). By doing this, we ensure reusability, and we avoid what is called the ‘God Function’, which is when a function/method is doing too much.

In following the open closed principle, new code was exactly that, new code. New functionality shouldn't compromise existing functionality or methods. The caveat to this is when changes to the backend were made. Some of the backend changes dramatically changed how the frontend had to send and receive data, and the required changes needed to be made. However, it is our understanding that is an appropriate violation of the open closed principle.

Interface segregation can be seen with our services. Each service is dedicated to a very specific task, and is named appropriately. For anything user related, all of the required methods can be found in the user service. This is applicable to all of the services. Finally, as discussed earlier, we inject each service into the required components, instead of hardcoding them. This ensures a couple of things. First, our code is clean, and more readable. Second, it keeps our code loosely coupled. You never want your codebase to be tightly coupled, otherwise it is not modular, and hard to add or remove functionality. Lastly, this keeps our code much more reusable, and in the process, making sure it is DRY.