



**ASHESI UNIVERSITY**

**BUILDING A BACKEND FRAMEWORK ON THE DENO JS RUNTIME**

**APPLIED PROJECT**

B.Sc. Management Information Systems

**Isaac Kumi**

**2021**

# **ASHESI UNIVERSITY**

**Fly-Deno: A backend web framework built on the deno Js runtime.**

## **Applied Project**

Applied Project submitted to the Department of Computer Science, Ashesi University in partial fulfilment of the requirements for the award of Bachelor of Science degree in Management Information System

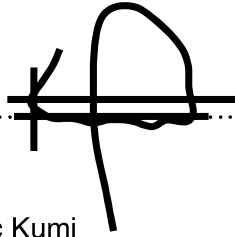
**Isaac Kumi**

**April 2021**

## Declaration

I hereby declare that this applied project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:



Candidate's Name:

Isaac Kumi

Date: 5-10-21

I hereby declare that preparation and presentation of this applied project were supervised in accordance with the guidelines on supervision of applied project laid down by Ashesi University.

Supervisor's Signature:



Supervisor's Name:

Todd Warren

Date: 5-10-21

## **Acknowledgement**

I want to thank the almighty God for seeing me through my four years of stay at Ashesi University. I also want to express my profound gratitude to my supervisor, Professor Todd Warren, for his guidance, comments and the recourses he provided me. To Reverend Divine Asem and the entire campus Gem family, I say God bless you for your support both in prayers and in kind. Lastly, to my family for always being there for me each time I needed their support.

## **Abstract**

This paper presents a minimal, opinionated and security-oriented backend framework built on the Deno Js runtime. I have had the opportunity to work with some popular backend frameworks like express Js, Adonis Js, and what have you. These frameworks are very powerful and get the work done within a short period. However, they compromise on user's security. They can access confidential information like environment variables, network, read and write access etc., without the user's knowledge.

Aside from the problems as mentioned earlier, there is a steep learning curve. Newbies have to spend a lot of time to understand some of the internal workings of the framework before they can get along.

This project presents developers, especially newbies, to build powerful but secured applications by writing less codes and leveraging their basic knowledge in programming to build applications. Also, developers can make use of existing libraries and modules in the framework as well. The framework is heavy on the model-view-controller architectural pattern.

**Keywords:** web framework, security, Deno Js

## Table of Contents

<b><i>Declaration</i></b> .....	<b>3</b>
<b><i>Acknowledgement</i></b> .....	<b>4</b>
<b><i>Abstract</i></b> .....	<b>5</b>
<b><i>Table of Figures</i></b> .....	<b>8</b>
<b><i>Chapter 1: Introduction</i></b> .....	<b>9</b>
<b><i>Chapter 2: Related Works</i></b> .....	<b>10</b>
<b><i>Chapter 3: Requirement Analysis and System Architecture</i></b> .....	<b>12</b>
<b>3.1 Requirement Design Overview</b> .....	<b>12</b>
<b>3.2 Introduction to MVC Architecture</b> .....	<b>12</b>
<b>3.3 Fly-Deno Framework core part</b> .....	<b>14</b>
<b>3.4 System Description</b> .....	<b>14</b>
3.4.1 System Features .....	15
<b>3.5 Supported Operating System/Environment</b> .....	<b>17</b>
<b>3.6 Functional and Non-Functional requirements</b> .....	<b>17</b>
3.6.1 Non-Functional Requirements .....	18
3.6.1.1 Security .....	18
3.6.1.2 Reliability .....	19
3.6.1.3 Maintainability .....	19
<b><i>Chapter 4: Implementation</i></b> .....	<b>19</b>
<b>4.1 The Core Parts</b> .....	<b>21</b>
4.1.1 The Core Parts.....	21

4.1.1.1 The Entry Point .....	22
4.1.1.2 The src folder .....	23
4.1.1.3 The public folder.....	24
4.1.1.4 The application folder .....	24
4.1.2 Implementation Process .....	28
4.1.2 Technologies and Tools .....	34
4.1.2.1 Languages .....	34
4.1.2.2 Libraries .....	34
4.1.2.3 Tools .....	35
<b><i>Chapter 5: System Testing</i></b> .....	<b>37</b>
<b>5.1 Unit Testing with Deno Test runner:</b> .....	<b>37</b>
<b><i>Chapter 6: Conclusion and Recommendation</i></b> .....	<b>39</b>
<b>6.1 Conclusion:</b> .....	<b>39</b>
<b>6.2 Limitation:</b> .....	<b>39</b>
<b>6.3 Recommendation</b> .....	<b>39</b>
<b><i>REFERENCES</i></b> .....	<b>41</b>

## Table of Figures

Figure 1 JavaScript frameworks survey.....	10
Figure 2 Model-View-Controller Architectural Pattern Overview.....	13
Figure 3 sample route in fly-deno framework .....	17
Figure 4 Functional requirement of a developer/user .....	18
Figure 5 Root directory of the fly-deno framework.....	22
Figure 6 Entry point of the fly-deno framework.....	<b>Error! Bookmark not defined.</b>
Figure 7 sample command to start the app server.....	23
Figure 8 Structure of the src folder .....	24
Figure 9 Structure of the app folder.....	26
Figure 10. content of the .env file in the app folder.....	26
Figure 11 The run method in the Application class.....	<b>Error! Bookmark not defined.</b>
Figure 12 GetRoutes method in fly-deno.....	30
Figure 13 The read routes function in deno-fly .....	30
Figure 14 . method for extracting handler .....	31
Figure 15 Controller class responsible for rendering view and communicating with data source .....	32
Figure 16 Enviroment variables initialized through the App constructor.....	32
Figure 17 Snippet of code for reading and parsing the env file.....	33
Figure 18 Code for setting read env variables .....	33
Figure 19 allowing network access in fly-deno framework.....	35
Figure 20 Unit test results .....	37
Figure 21 snippet of code for unit test .....	38
Figure 22 Snippet of code for testing home controller .....	38



## **Chapter 1: Introduction**

### **1.1 Introduction**

Over the past few decades, web application development has seen quite a number of improvements. These improvements are usually libraries and frameworks that seek to improve the efficiency of development by providing boilerplate code, Application Programming Interfaces (API's) and abstractions which takes a lot of time and resources to build. Also, the issue of security in a web application has been the major concern of most companies. Among the many attacks, the most common is Structured Query Language (SQL) injection and cross-site scripting (XSS) attacks [1]. This paper introduces a backend application framework built on the deno Js runtime. The framework will be built using the Model-View-Controller architectural pattern. An architectural pattern that is used by most software.

### **1.2 Background**

I took an online course on web application security during my sophomore year in Ashesi. And ever since, I have been so concerned about building a secured application without compromising on user's privacy. Since the birth of web application frameworks like Laravel-PHP, Adonis, etc., we have seen most security problems creep up over the years, yet none of these runtimes and frameworks has addressed such an issue. It will interest you to know that an application is given the exact same permission and privilege as the person executing the script. This, in the long run, will expose user's confidential information stored on the computer to web applications. In mobile applications, the user has some control over what kind of information the application can access. We normally see this during the very first stage of installation. The users are required to allow or deny access to certain features like camera, gallery, microphone and many more. This is not the case for the web application.

## Chapter 2: Related Works

There are many frameworks for a web application that uses the architecture of the model view controller [7]—Adonis, Express, Vue Js, Electron, etc., for example. In 2015, SitePoint, one of the most popular websites providing only tutorials for web and software-linked developers, carried out a global survey to find valuable insights into the various frameworks of JavaScript. One of the most important questions is why developers prefer one framework over another. Figure 1 shows the developer graph using different manufacturing frameworks.

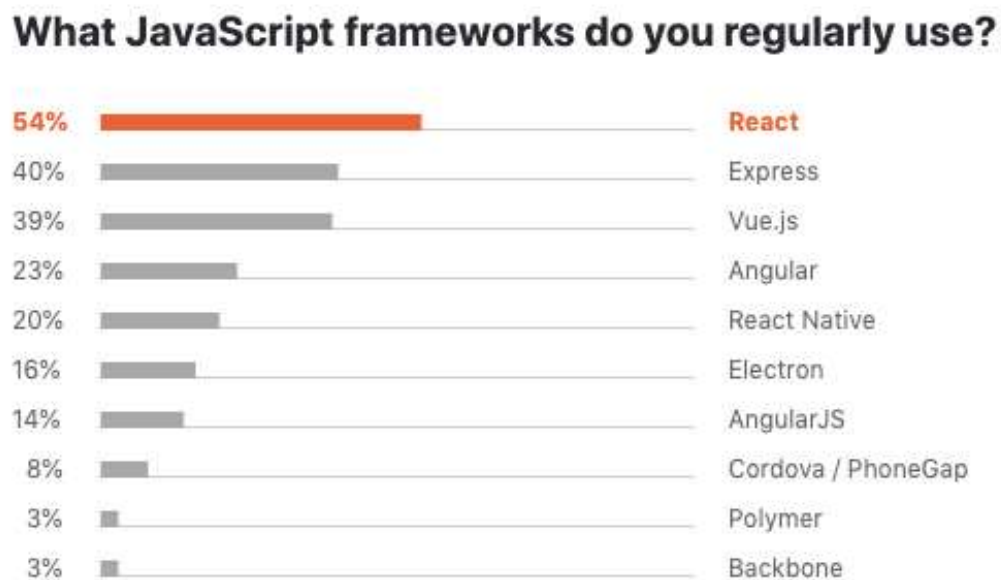


Figure 1 JavaScript frameworks survey

### Express Js vs Adonis Js

ExpressJS is a web framework that is minimalist. It allows you to work freely with any database, ORM or folder structure.

AdonisJS is an MVC framework, on the other hand. The model, views and controllers stand for MVC.

Each project of AdonisJS ships a number of files and folders. To work with it, we must first understand the structure of the folder. But a good project structure is important for a scalable Web application. So AdonisJS ensures that you're right.

With a solid routing system, we can quickly develop an application. Both frameworks have an appropriate routing system.

This is applicable to both static and dynamic routes.

The express router is very simple, but it gets the job done.

Adonis allows us to connect pathways to controllers, group pathways and create REST resources.

The problem Identified with these frameworks is that not only do they have steep learning curves, but newbies are also required to understand certain core features of the framework before they can build with them. This makes development difficult and time consuming. In this proposed solution, the aforesaid problems will be dealt with.

## **Chapter 3: Requirement Analysis and System Architecture**

### **3.1 Requirement Design Overview**

When developing software, defining requirements before starting the development phase saves much time and sometimes reduces financial costs. Gathering software requirements helps to clearly define everything that the software will accomplish and serves as the basis for designing and the development process. This chapter provides an overview of the design methods and architectural pattern used in coming out with this framework.

### **3.2 Introduction to Fly-deno Architecture**

MVC, was first described in 1979 by Trivet Reinking, while working on Smalltalk at Xerox PARC[10]. The Model-View-Controller pattern is the most used pattern for today's world web applications because it has been proven as an effective way to develop applications. The MVC pattern separates an application into three separate layers: model, view and controller that work separately to produce the same result. The controller handles the model and view layers to enable them to work together. The controller is responsible for receiving a request from the client, and invokes the model, responsible for interacting with a data source, to perform the requested operations and presents the data to the view. which is responsible for rendering web pages.

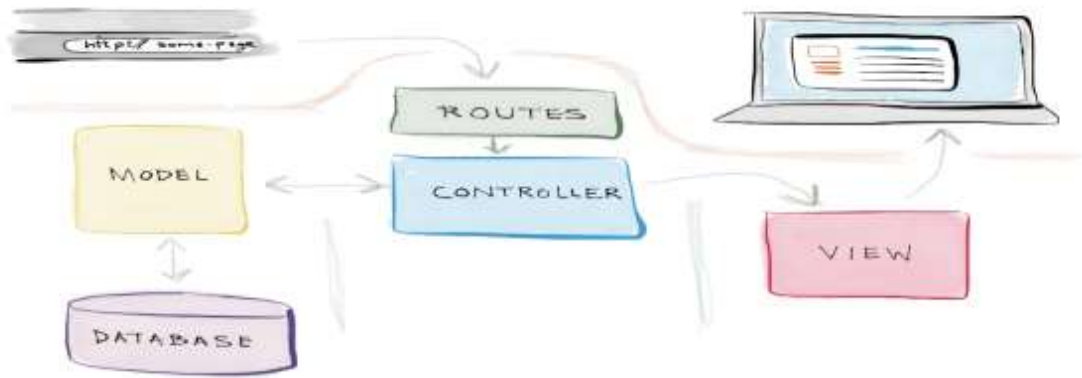


Figure 2 Model-View-Controller Architectural Pattern Overview

The diagram above shows the MVC architectural pattern and how data flows through it. Here, a client hits an endpoint to request a resource. Under the hood, *fly-deno* will take the endpoint and search through a predefined route based on the HTTP verb (i.e., it directs GET requests to *get.json* route whereas POST request is directed to *post.json*). The route contains a path, handler (controller class and method connected to the path) and an optional name parameter. When the endpoint entered in the URL matches a path in the route, the handler(controller) is then evoked. The controller, which is connected to the data source, then makes a request to the database for resource and then passes in the retrieved data to the view. The view then presents the resource/page requested by the client.

### 3.2.1 System Architecture

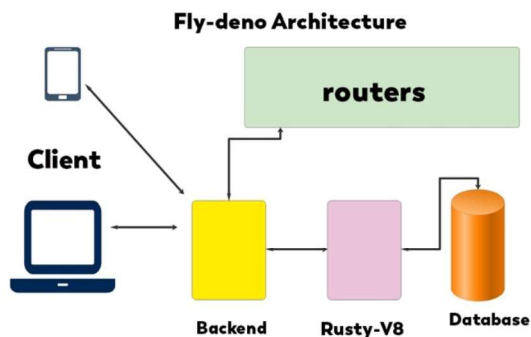


Figure 3. Fly-deno system Architecture

The diagram above shows the system architecture of fly-deno framework. Over here a client makes request to the backend via an endpoint. The backend searches through the routers and fires request to the rusty-v8 engine (the main engine that process requests in Deno). The rusty-v8 engine is also able to communicate with databases.

### **3.3 Fly-Deno Framework core part**

This section is devoted to the workflow of the fly-deno backend framework. Fly-deno is a minimalist framework built with security and MVC architectural pattern in mind. Fly-deno is written in typescript (a superset and a statically typed of JavaScript) with Deno Js as its runtime. The framework's folder structure is similar to that of Angular Js, Laravel and the popular frameworks out there with some small modification to make newbies navigate the framework. In the root directory of the application, there are two files, namely *.env* and *.env.example* similar to that of Laravel-PHP. The former is used to store information that are confidential to the application and therefore needs to be hidden from the public eye. It contains information such as application name, Application Programming Interfaces (API) keys, database connection information, application secrete key etc. The latter stores' similar information as the former but it is meant for storing production information like live-server URL, database connection strings etc. Fly-deno also ships with an in-built handlebars template engine. This will allow backend data to be easily interpolated into views and vice-versa.

### **3.4 System Description**

Fly-deno backend framework is a security-oriented, lightweight and an MVC focused framework. Unlike other frameworks like Adonis Js where newbies have to devote more time to learning the core features like migration, page routing etc. Sometimes developers have to drop their knowledge on certain things they know just to understand the bit and bytes of frameworks. For instance, a developer with Structured Query Language (SQL) knowledge may end up not applying any of this skill in Adonis because the framework does not require developers to write raw SQL queries. These frameworks mostly ship with their own migration libraries, which developers and newbies have to understand before they can fully use the framework. Fly-deno takes a new approach. It was designed with beginners and security in mind. It provides developers with a JSON-like route that is easy to understand [5]. Also, with MVC at its core, developers will be able to separate their application logic from the business logic hence making it easier to maintain and scale the codebase. Developers willing to build complex application can do so by importing third party and standard libraries provided by denoland.

### **3.4.1 System Features**

Fly-deno has brought a shift in the way applications are built. Its features include:

No Node package manager (NPM), MVC support, security, typescript as a first-class citizen, JSON route etc

#### **i. No NPM**

The fly-deno framework runs on the deno JS runtime and therefore does not have any package manager as seen in Node Js, Angular Js, Adonis Js etc. Decan Posited, there had been an increase in the number of vulnerabilities found in the application that uses npm [1].

This is very alarming, but thanks to deno, which has made it possible to build an application without relying on such insecure packages.

ii.      Typescript supports out of the box

Developers around the globe understands how hectic it is to configure typescript compilers in application. Typescript adds statically typed feature on top of JavaScript due to that it helps developers to catch errors at runtime and fix them as soon as possible before rolling their applications out. With the fly-deno framework, developers do not need to go through the pain of configuring typescript before they can use them. It ships with deno.js which is the runtime the fly-deno framework runs on.

iii.     Model-View-Controller support

With this, developers will be able to know where to do what in the application. For example, if the developer wants to work with a controller, he just need to locate the controller folder and write his code.

iv.      Preconfigured router class

Another key feature of the fly-deno framework is the fact that routers are preconfigured. So developers do not need to understand the underlying workings of how it works.



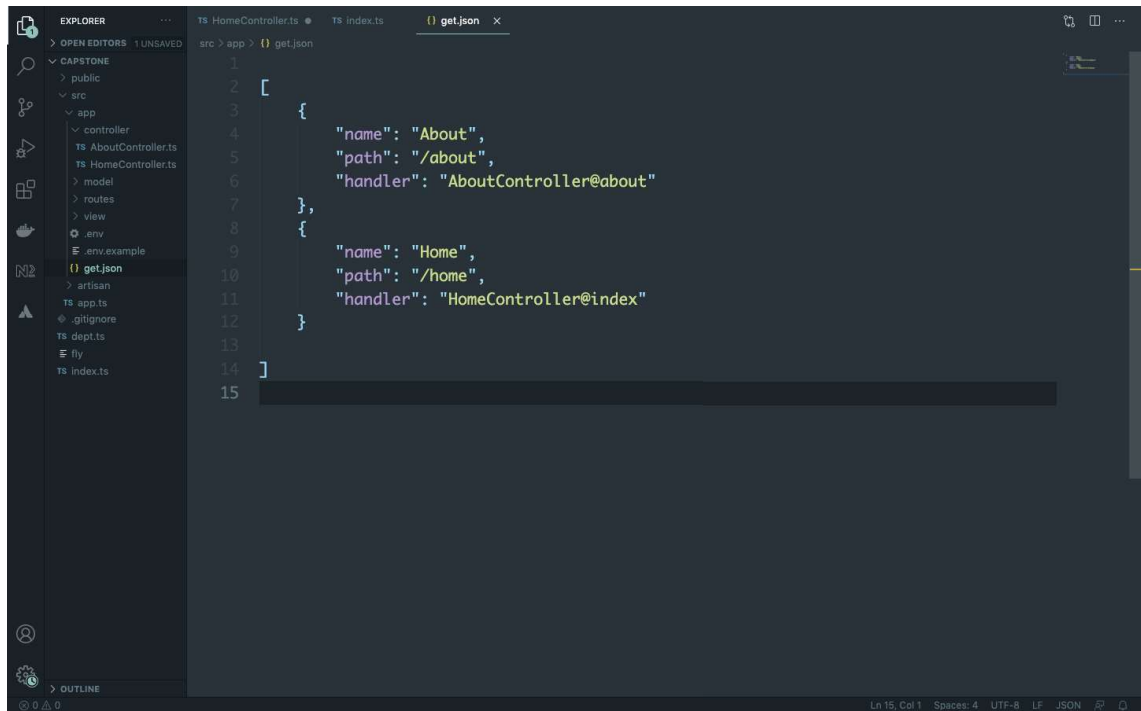


Figure 4 sample route in fly-deno framework

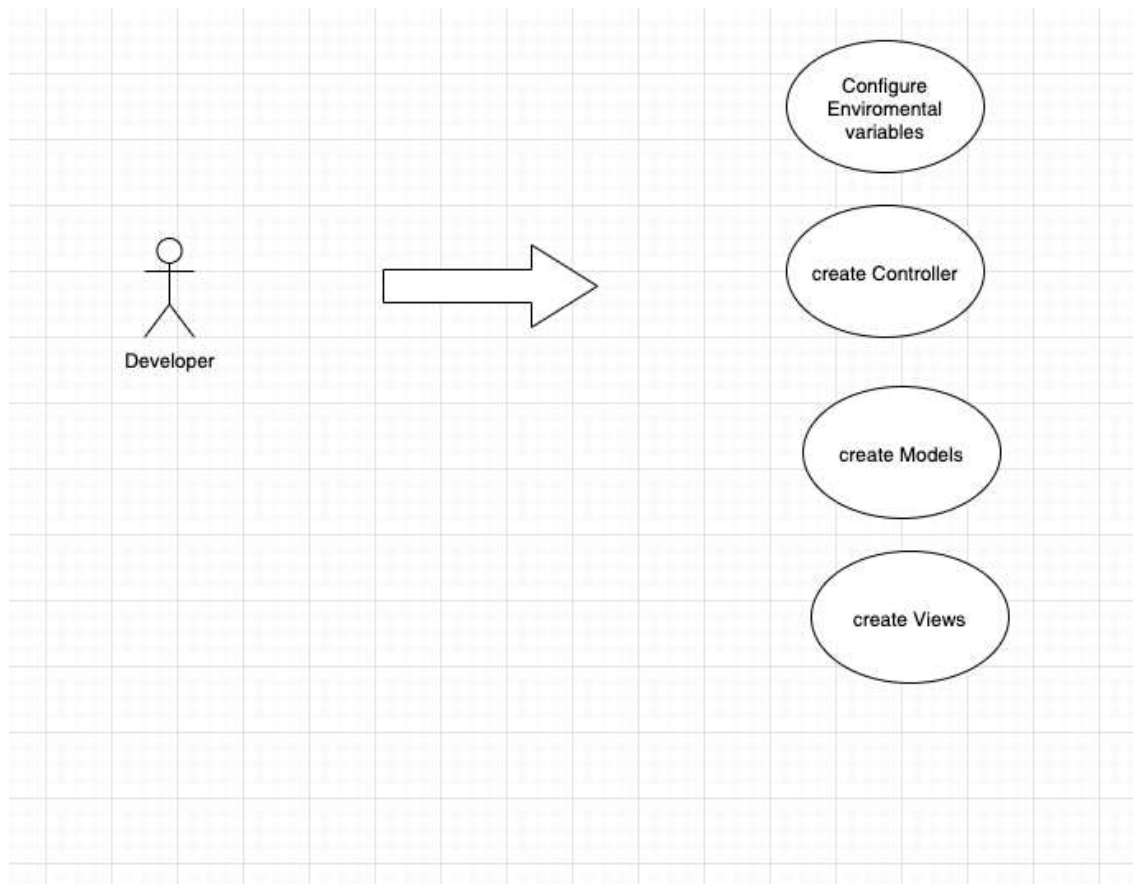
### 3.5 Supported Operating System/Environment

The fly-deno framework can work in any operating system (OS) provided the OS has deno runtime installed. There is no need to install web server software like Apache or nginx to run

### 3.6 Functional and Non-Functional requirements

Functional requirements provide the programmer with a snapshot of how to use the framework.

It starts from setting up the project to deployment.



*Figure 5 Functional requirement of a developer/user*

### **3.6.1 Non-Functional Requirements**

Non-functional requirements are requirements of the framework that will ensure reliability and security in any application the framework will be used for in order to attain the desired result. Some of the non-functional requirements of fly-deno are security, performance, reliability, maintainability, scalability and usability.

#### **3.6.1.1 Security**

The framework limits the developer to certain confidential information unless explicitly defined. This helps ensure that applications are secured and are not given exact permission as the root user. Also, the developer is at liberty to handle requests in their own convenient way without compromising on security.

### **3.6.1.2 Reliability**

Fly-deno should be able to catch errors and display them in a nicely formatted stack trace in a way that the developer can read and understand. With this, developers will find it easier to catch and debug errors with ease.

### **3.6.1.3 Maintainability**

Even though fly-deno does not use NPM as its package manager, it has a way of locking dependencies so that newer versions of dependencies will not conflict with older versions. This makes maintaining the codebase much easier since dependencies mismatch and environmental discrepancies are avoided [1].

## **Chapter 4: Implementation**

The chapter is devoted to how the fly-deno framework was built. It shows some of the inner workings, routing, environment variables etc are implemented.

## 4.0 Application Walkthrough

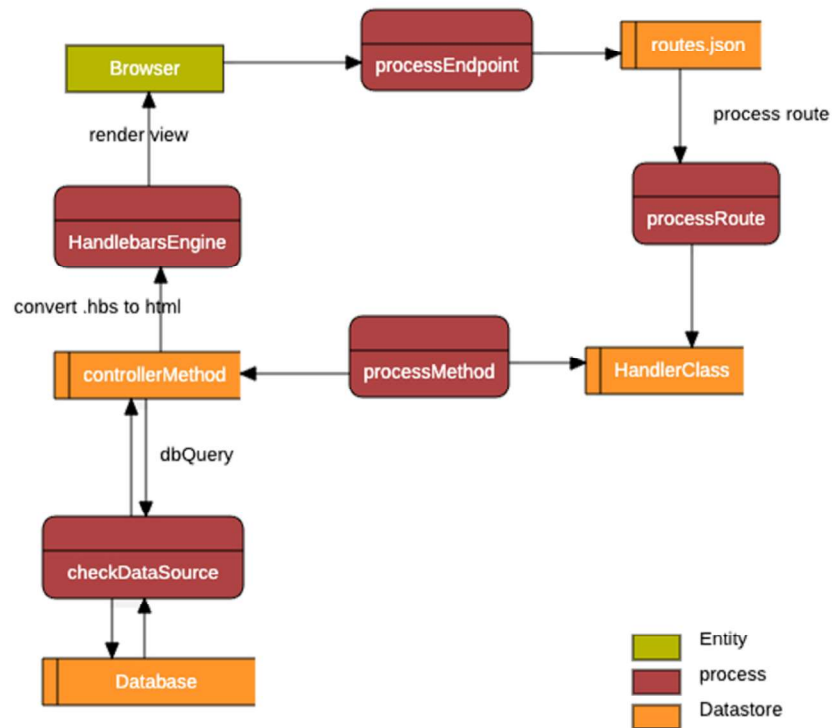


Figure 6. Application Walkthrough

Figure 6 shows a walkthrough of the fly-deno framework. First a client hit and endpoint in a browser, the framework will extract the path visited from the url and search through the routes.json file. Once it finds a match in the routes.json, it call the handler connected to the path. It then get the handler class and the controller from the found handler. The handler contains a class name and a method. The method now fires a request to the database for data. Once the data is retrieved from the database, the controller then passes the data to a handlebar template engine for processing. The handlebar template engine is responsible interpolating data into loaded pages. Once handlebar engine is done with the data processing, it then render the view with the data.

## 4.1 The Core Parts

The core part of this frameworks is:

- I. Environment variables initialization: This shows how environment variables are created.
- II. Folder structure: This shows the structure of the folders as well as how files/folders are organized. It gives the user an idea of where to do what in the framework.
- III. Routing: This shows how pages and are being connected to each other in the application.
- IV. Controllers: This forms a major part of the framework. It helps users to communicate between a data source and the view.
- V. Database: This shows how the database class is being created and used.
- VI. Custom Url library: The url library will be used to load binary files , css and javascript files into loaded pages.

### 4.1.1 The Core Parts

For easier navigation, the framework has two folders in its root directory; *public folder*, *src folder*, *index.ts* (entry point of the application) and *fly* ( bash file for running the application).

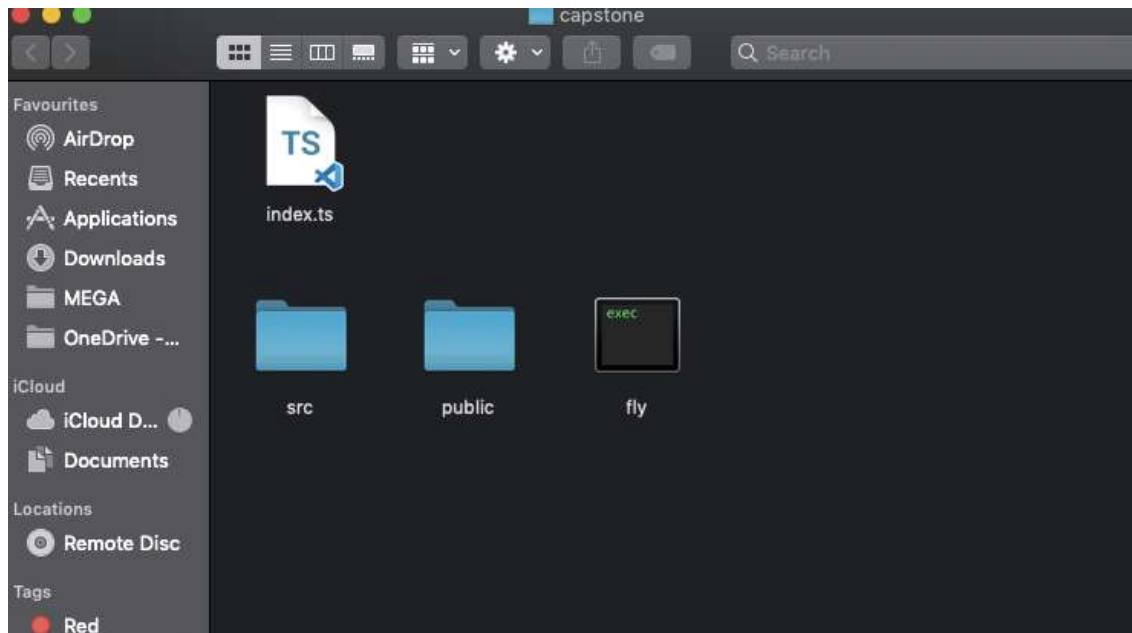


Figure 7 Root directory of the fly-deno framework

#### 4.1.1.1 The Entry Point

The applications entry point is the *index.ts*. This is where the server is being called before the application can run. This file contains the port on which the application will run on. The developer is required to set a port number for the application to use, else the application will use a default port (8000) to run the application. Fly-deno is aimed at making development fast, simple and fun. As a result, the render function has been hidden from the controller. It's get triggered in the entry point.

```

14 console.log(`🚀 Server is running on http://localhost:${PORT}`);
15
16 for await (const req of server) {
17
18   // CustomServer.setServer(req);
19   // CustomServer.setDir(ROOT_DIR);
20   // await render('index',{title: 'Isaac'});
21   CustomServer.setServer(req);
22   CustomServer.setDir(ROOT_DIR);
23   let res: any = await app.setup() => {
24     CustomServer.setServer(req);
25     CustomServer.setDir(ROOT_DIR);
26     console.log({ body: "Welcome to the Fly backend Framework" });
27   }).run(req);
28   CustomServer.persistData(res);
29   res = CustomServer.getRenderData();
30   // console.log(res);
31   if(res !== undefined && res !== null && typeof(res) === 'object') {
32     // console.log(res);
33     await render(res.template,res.data);
34   }

```

Figure 8 Entry point of the fly-deno framework

```

Isaacs-Air:capstone isaackumi$ sh fly
Welcome to fly
🚀 Server is running on http://localhost:8000

```

Figure 9 sample command to start the app server

#### 4.1.1.2 The src folder

The src folder contains the application logic and the backend logic. This folder structure was borrowed from Angular, ionic and react Js.

```

.
├── app
│   ├── controller
│   │   ├── AboutController.ts
│   │   └── HomeController.ts
│   ├── get.json
│   ├── model
│   ├── routes
│   │   └── get.json
│   └── view
│       └── index.hbs
├── app.ts
├── artisan
│   ├── cli
│   ├── include
│   │   ├── filesystem.ts
│   │   ├── function.js
│   │   ├── interfaces.ts
│   │   ├── lib.ts
│   │   └── test_env.ts
│   └── utility
│       ├── protocols
│       │   ├── Controller.ts
│       │   ├── Model.ts
│       │   ├── MySQL.ts
│       │   ├── View.ts
│       │   ├── usables.ts
│       │   └── view
│       │       └── index.hbs
│       └── routers
│           ├── Authentication.ts
│           └── Router.ts
└── 12 directories, 19 files
Isaacs-Air:src isaackumi$

```

Figure 10 Structure of the src folder

#### 4.1.1.3 The public folder

The public folder contains view related files such as handlebars, Hypertext markup Language (HTML), JavaScript and cascading style sheets (CSS). It also houses useful assets like images, favicons etc.

#### 4.1.1.4 The application folder

This is where the developer will write most of his code. Its folders are structured to conform to the MVC pattern. It contains four folders; the model, view, controller and the routes folder. Aside from that, there are also two hidden files, namely .env and .env.example.



- I. Model folder: The model folder contains database related queries. It is the “M” part of the MVC pattern.
- II. View folder: The view folder will contain frontend related codes. It is the “V” part of the MVC pattern.
- III. Controller folder: This contains the business logic. It is the “C” part of the MVC pattern.
- IV. Route folder: All routes for the application are defined in the routes folder.
- V. .env file: This file contains information that will be used throughout the application. It also houses confidential information.
- VI. Public folder: The public folder lives in the application root. It is a place where frontend files like html, handlebars, assets etc. live.
- VII. Page Not found folder: The fly-deno framework has a default file that get called when a page cannot be found. To make this more flexible, the developer can define his own 404 page, but it should leave inside the public folder and connected to a route as well.

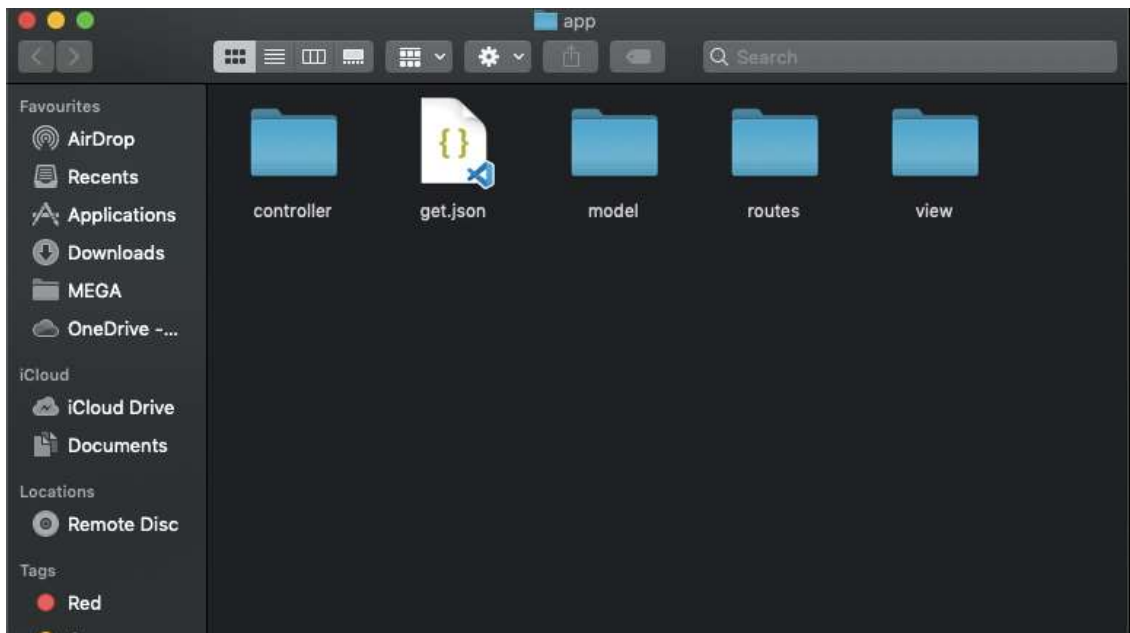


Figure 11 Structure of the app folder

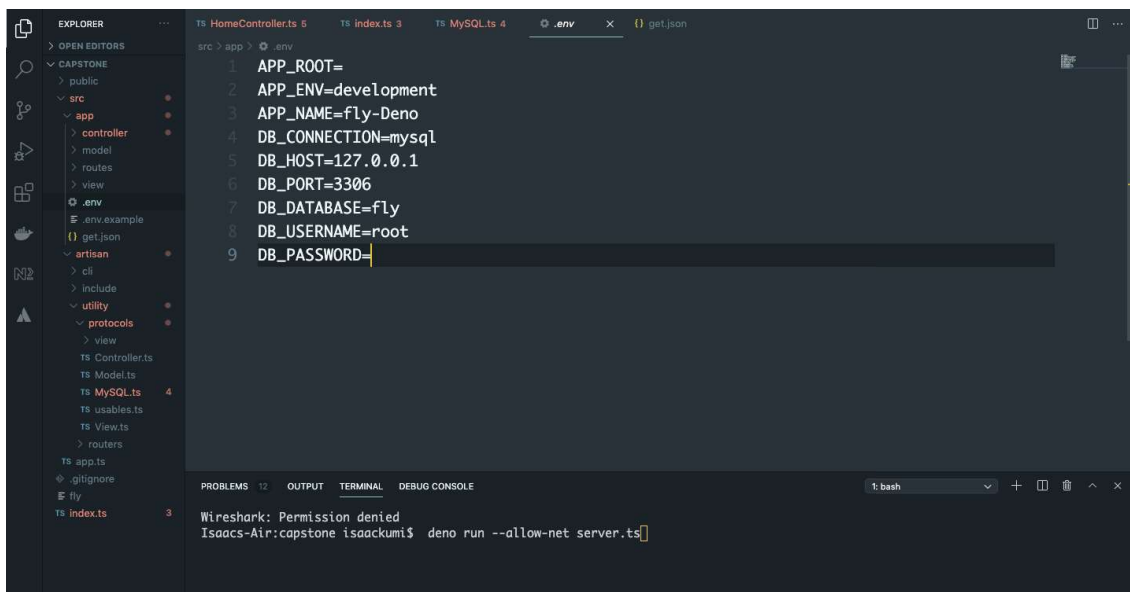


Figure 12. content of the .env file in the app folder

## Comparing Fly-deno to Express Js

Routing and Controllers:

```

7
8 export class HomeController {
9
10 public test(req: ServerRequest) {
11   return {
12     template: 'test',
13     data: { title: "Isaac Kumi" }
14   };
15 }
16
17 public index(req: ServerRequest) {
18
19   return {
20     template: 'index',
21     data: { app_name: "Fly-deno Framework" }
22   };
23
24 }
25

```

Figure 13 Fly-deno controllers

```

43
44 router.get('/service/:id', async (req, res) => {
45
46   const id = req.params.id
47
48   await Service.findById({_id:id})
49   .exec()
50   .then( data => {
51     res.send(data)
52     console.log(data)
53   })
54   .catch( () => {
55     res.json({message:"Service not found"});
56   })
57
58 });
59

```

Figure 14 Express Js combined routing and controller

Figure 13 shows how fly-deno does its controller. The controller returns a shape of data for the framework to use. With this type of controllers, newbies will not find it difficult to understand what is going on in the application. There is a hidden render function that process the data sent

by the controllers for processing. As seen, most the complexities have been hidden from the developer so he can focus more on building rather than trying to understand what function to call to do what in the application.

Figure 14 on the other hand shows how express Js does it routing and controllers. Its controllers and call-back functions inside the routers. This type of implementation does not favour beginners. This because newbies are mostly new to programming and framework in general, so they get overwhelmed when they are exposed to something they have no idea about.

Also, with express Js, when an endpoint is visited, it goes through all the routes which end up slowing the application. In fly-deno framework, once the path is found, the application doesn't go through the rest of the route, hence making the application faster. Also, fly-deno controller is very, simple and clean as compared to express Js. The developer only needs to define the shape of the data for the framework.

#### **4.1.2 Implementation Process**

The project started off by building a method that will be used to run the application with ease. It takes a parameter of type server request. server request is a method that handles the request, response in deno-http library.

```

18
19 public setup(callable: Function) {
20     this.callback = callable;
21     return this;
22 }
23
24 public async run(req: any) {
25     console.log(`method: ${req.method} - server: localhost - path: ${req.url}`)
26     ++Application.counter;
27     if(Application.counter % 2 == 0 || Application.counter > 2 ) {
28         Application.counter = 0;
29     }
30     if(Application.counter >= 1) {
31         this.callback(req);
32         switch (req.method.toLowerCase()) {
33             case "get":
34                 var router = new Router();
35
36                 return await this.getRoutes(req.url);
37

```

Figure 15 The run method in the application class

When the run method is called, it calls the getRoutes() method, which is responsible for reading through the route file based on the request method. That is, if the request is a GET, it will only search through the get.json routes for the path and handler connected to them.

```

public getRoutes(path: string) {
  const data = readRoutes(`${this.getApproot()}/src/app/get.json`);
  const parsed_data = JSON.parse(data);
  // console.log(parsed_data);
  var to_return = {};

  var self = this;

  parsed_data.forEach(
    function (element: { name?: string; path: string; handler: string }) {
      if (path == element.path) {
        // console.log(typeof element) object

        return self.process(element);
      } else {
        return function () {
          console.log("Invalid Path");
        };
      }
    },
    );
};

```

Figure 16 GetRoutes method in fly-deno

In the figure above, the routes file (get.json) is being read by a function called readRoutes(). readRoutes() is responsible for reading the route file and return its content as a string. It takes a file path as string to be able to read the file. Also, because deno is secured at its core, read and write permission is required to run this function. After the file content is returned, another method (process ()) is called to extract the handler and the controller.

```

76
77 export function readRoutes(filePath: string): string {
78   return new TextDecoder("utf-8").decode(Deno.readFileSync(filePath));
79 }
80
81 export default dotenv;
82

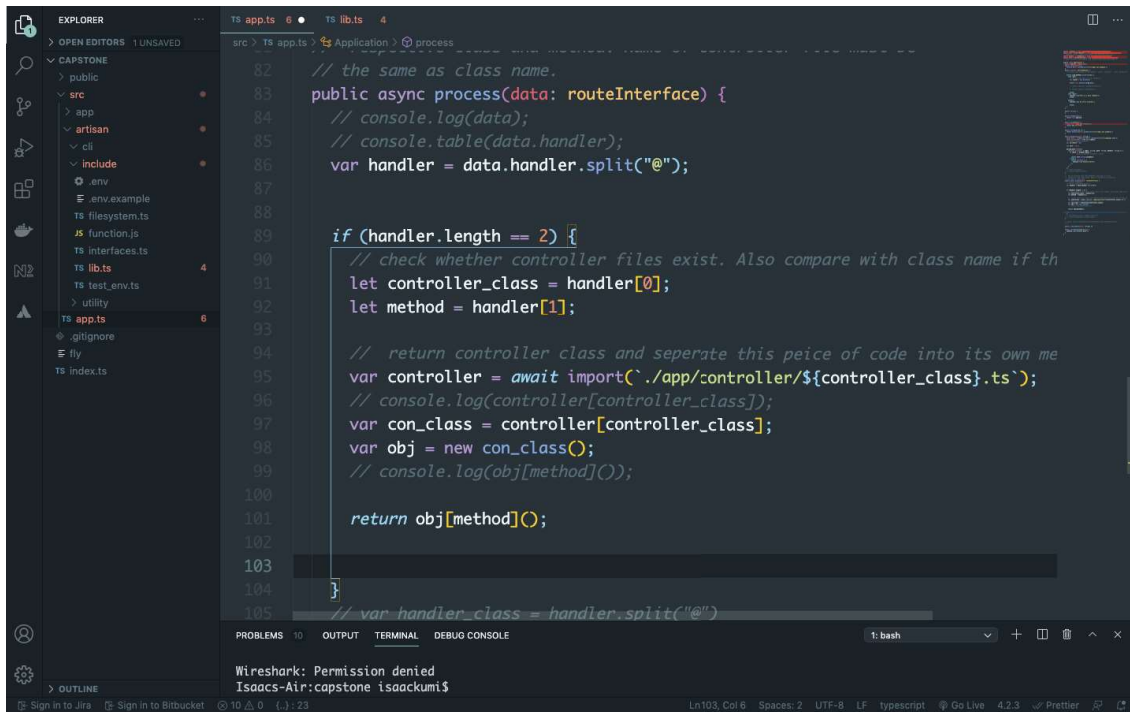
```

PROBLEMS 10 OUTPUT TERMINAL DEBUG CONSOLE

1: bash

Wireshark: Permission denied  
Isaacs-Air:capstone isaackumi\$

Figure 17 The read routes function in deno-fly



```
82 // the same as class name.  
83 public async process(data: routeInterface) {  
84     // console.log(data);  
85     // console.table(data.handler);  
86     var handler = data.handler.split("@");  
87  
88  
89     if (handler.length == 2) {  
90         // check whether controller files exist. Also compare with class name if th  
91         let controller_class = handler[0];  
92         let method = handler[1];  
93  
94         // return controller class and seperate this peice of code into its own me  
95         var controller = await import(`./app/controller/${controller_class}.ts`);  
96         // console.log(controller[controller_class]);  
97         var con_class = controller[controller_class];  
98         var obj = new con_class();  
99         // console.log(obj[method]());  
100  
101         return obj[method]();  
102     }  
103  
104 }  
105 // var handler_class = handler.split("@")
```

Figure 18 . method for extracting handler

In the process method, the handler returned by readRoute method is being split into an array of length two, with the first part being the class name of the controller and the second being the method in the controller class. In order to get the developer/users defined controllers in the application, a dynamic import statement is used to load controller classes for further processing in the framework.

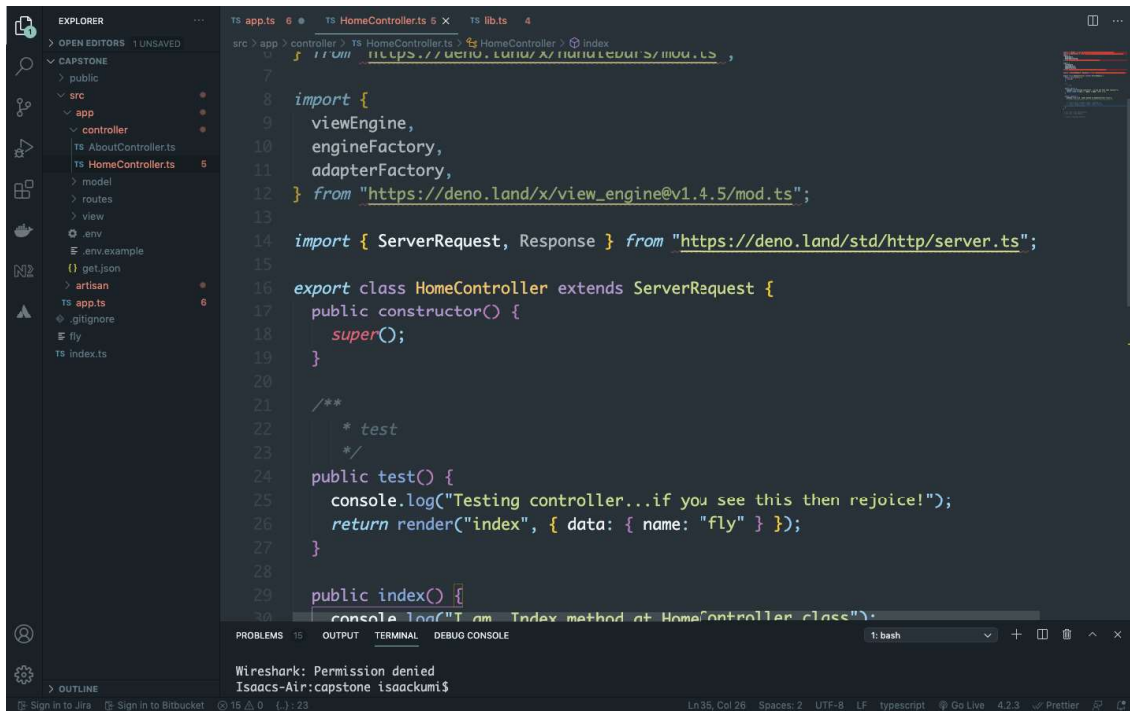


Figure 19 Controller class responsible for rendering view and communicating with data source

The framework borrows some concept from Laravel. For instance, in the Application class constructor, an environment variable is being initialized. This will help the developers/users to be able to use variables everywhere in the application. The env file follows laravel's environment file structure.



Figure 20 Enviroment variables initialized through the App constructor



```

21   * @return the .env file content as a string.
22   */
23  const readFileSync = (filePath: string): string => {
24    return new TextDecoder("utf-8").decode(Deno.readFileSync(filePath));
25  };
26
27   /**
28    * parse the .env file content from string to a two-dimensional array.
29    * @param decodedEnv - the '.env' file content.
30    * @return a two-dimensional array of type [[key, value], [key, value]]
31    */
32
33  const parse = (decodedEnv: string): string[][] => {
34     // convert the content of .env to an array of lines.
35    const envList = decodedEnv.split(LINE_BREAK);
36
37     // loop throw the lines and split them tp pairs [keys, value]
38    return envList.map((item) => {
39      const pair = item.split("=");
40      const size = Object.keys(pair).length;
41       // console.log(pair);
42       // console.log(typeof size);
43      return [pair[0].trim(), pair[1].trim()];
44    });

```

Figure 21 Snippet of code for reading and parsing the env file

```

53   /**
54    * set the new vars to the deno env object.
55    * @param parsedEnv - a two-dimensional array of type [[key, value], [key, value]]
56    */
57
58  const setEnv = (parsedEnv: string[][]): void => {
59    for (let i = 0; i < parsedEnv.length; i++) {
60      Deno.env.set(parsedEnv[i][0], parsedEnv[i][1]);
61    }
62  };
63
64   /**
65    * the main function.
66    * @return an abobject contains all the environment variables.
67    */
68
69  const dotenv = (envFilePath?: string) => {
70    setEnv(parse(readFileSync(envFilePath || ".env")));
71
72     // return an object that contains all the environment variables
73     // including the default variables
74    return Deno.env.toObject();
75  };
76

```

Figure 22 Code for setting read env variables

#### **4.1.2 Technologies and Tools**

This section describes the technologies, tools, and development stacks that were used to implement the proposed fly-deno framework.

##### **4.1.2.1 Languages**

###### **Typescript/JavaScript:**

TypeScript is an open-source language that adds static type definitions to JavaScript. Types allow you to describe the shape of an object, which improves documentation and allows TypeScript to validate that your code is working properly (*Typed JavaScript at Any Scale.*,2021). The TypeScript compiler or Babel converts TypeScript code to JavaScript code. This JavaScript is clean, simple code that can be used anywhere JavaScript is supported, such as in a browser, Node.JS [4], or your apps. Considering these features, I used typescript to write classes, interfaces and decorators in the application.

###### **MySQL:**

MySQL was used to implement the project's database layer. MySQL is a relational database management system that was chosen due to its ease of use, speed, and availability for free and open-source use [11].

##### **4.1.2.2 Libraries**

###### **Deno-Http:**

This is the library used to run the application server. It contains in built methods for sending and responding to requests. It also comes with a method that allows the application to listen on a specific port for changes in the application while running the application. Because deno is security oriented, this library will not be able to access network recourses unless access is

granted by the user running the application. To enable access, the user has to pass *–allow-net* flag.

Below is a sample command for allowing network access in fly-deno framework

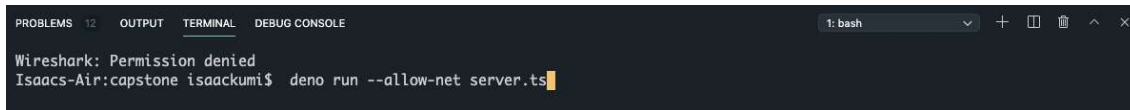
A screenshot of a terminal window within a code editor. The terminal has tabs for 'PROBLEMS', 'OUTPUT', 'TERMINAL', and 'DEBUG CONSOLE'. The 'TERMINAL' tab is active. It shows a message from Wireshark: 'Permission denied'. Below that, a command is entered in the terminal: 'Isaacs-Air:capstone isaackumi\$ deno run --allow-net server.ts'. The terminal title bar indicates it's a 'bash' shell.

Figure 23 allowing network access in fly-deno framework

### **Deno Path Library:**

I used the path library to navigate through the filesystem to find files and folders in the application's directory. To use this library the user must allow read and write permission by passing *–allow-read* and *–allow-write* flags when running the application.

### **MySQL Connect Driver:**

MySQL driver is a library that is used to connect deno application to mysql database [11]. I used this library to communicate with mysql in the application.

### **Deno- View-Engine:**

Just like any framework, fly-deno has support for handlebars template engine [3]. This was possible because of the view-engine library that ships with deno runtime.

## **4.1.2.3 Tools**

### **Postman:**

Postman is an API client. It is used to build, test, debug and document API's. This client was used to test endpoints in the framework.

### **Microsoft VSCode:**

Microsoft Visual Studio Code is a free source-code editor that works on Windows, Linux, and macOS. Among the features are debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

### **Phpmyadmin:**

PhpMyAdmin is a relational database management system for managing MySQL on the web. It is a simple lightweight web application that supports many MySQL and MariaDB database operations. It was used to regulate SQL operations.

### **Xampp:**

XAMPP is a web server solution package developed by Apache Friends for open-source and free, which includes the Apache HTTP Server, the MariaDB database and PHP and Perl scripts interpreters. I used Xampp to start mysql sever in other to communicate with the databases.

### **Git and GitHub:**

Tracking changes in code is a pain in the neck. Things get messy when the application begins to grow. In this framework, Git (a version control tool) was used to track changes in the project. GitHub which is a cloud storage version control tool works with Git. So, each time I make a change in the application I push it to GitHub through Git commands.

### **Bash:**

Bash is a command-line utility that takes commands and returns results based on the commands inputted [2]. I used bash to automate some part of the application. For instance, I realized that each time I run the code, I had to pass all the required flags for it to work, so I wrote a command-line script with bash as the interpreter to run the server each time I call it.

## Chapter 5: System Testing

The techniques and processes involved in fly-deno Framework software testing are highlighted in this chapter. It is essential to verify the functionality and functionality of the tests, as described in Chapter 2. To accomplish this, the functional or integration testing procedure is used instead of testing each component separately for the whole application. A detailed report of test cases, test results and analysis of test results are also included in the following section of the chapter.

### 7.1 Unit Testing with Deno Test runner:

Unit testing is a technique for testing individual system components like a function, method, or class [8]. Unit testing verifies the functionality of individual objects and fly-deno functions to ensure that user operations on each object produce the expected results when given valid [9], invalid, and null parameters. The Deno test runner was used for unit testing.

A screenshot of a terminal window with a dark background. The terminal shows the command 'deno test --allow-read' being executed. The output indicates that three tests were run and all passed successfully. The tests are: 'Testing run method', 'Testing process method', and 'Testing index-HomeController'. Each test is followed by '... ok' and a duration in parentheses. The final summary line states 'test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out'. The prompt 'Isaacs-Air:test isaackumi\$' is visible at the bottom.

```
Isaacs-Air:test isaackumi$ deno test --allow-read
Check file:///Users/isaackumi/Desktop/capstone/src/app/test/.deno.test.ts
running 3 tests
test Testing run method ... ok (22ms)
test Testing process method ... ok (2ms)
test Testing index-HomeController ... ok (3ms)

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out (35ms)

Isaacs-Air:test isaackumi$
```

Figure 24 Unit test results

```
1 import { ServerRequest } from 'https://deno.land/std/http/server.ts';
2 import { Application } from '../app.ts';
3 import { assertEquals } from 'https://deno.land/std@0.95.0/testing/asserts.ts';
4
5
6
7 const app = new Application();
8
9 Deno.test("Testing run method", async (s : ServerRequest) => {
10
11     assertEquals( await app.run(s),"Welcome to fly");
12
13 });
14
15 Deno.test("Testing process method", () => {
16     assertEquals( await app.process("/home"),"/Users/isaackumi/Desktop/capstone/s
17 });
```

PROBLEMS 28 OUTPUT TERMINAL DEBUG CONSOLE 1: bash

Isaacs-Air:test isaackumi\$ deno test --allow-read  
Check file:///Users/isaackumi/Desktop/capstone/src/app/test/.deno.test.ts  
running 3 tests  
test Testing run method ... ok (22ms)  
test Testing process method ... ok (2ms)  
test Testing index-HomeController ... ok (3ms)  
  
test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out (35ms)  
  
Isaacs-Air:test isaackumi\$ pwd  
/Users/isaackumi/Desktop/capstone/src/app/test  
Isaacs-Air:test isaackumi\$

Figure 25 snippet of code for unit test

```
Deno.test("Testing index-HomeController", () => {  
  
    assertEquals( await app.getRoutes("/home"),"/Users/isaackumi/Desktop/capstone  
  
});
```

Figure 26 Snippet of code for testing home controller

## **Chapter 6: Conclusion and Recommendation**

### **6.1 Conclusion:**

This document aims to use methodologies and principles of software engineering to develop a minimal but safe framework for developers. This approach allows any developer to take the fly-deno framework and create an application or microservices on it, which will enable them to be flexible to use all available libraries for this task as more complexity is demanded.

### **6.2 Limitation:**

The framework is not yet a matured framework like Adonis, Express, Oak and others. There are a lot of complexities that must be catered for. Currently, the framework does not have a session manager. The deno runtime is new and still growing, so most of the core libraries that will be used to develop such complex functionalities are now being developed by the Deno community.

Also, most of the standard libraries are not in their stable version so it ends up blowing up your project when they are added.

### **6.3 Recommendation**

In the future, the following features can be developed to improve system efficiency.

- I. Add a session manager to be able to track users across web pages.
- II. Popular frontend frameworks and libraries like React, Angular and Vue should be made pluggable into the framework.
- III. A command line tool for generating controller classes and migration scripts.

- IV. The framework should be able to work with multiple database instances asynchronously.
- V. The framework should have a bash script to bundle and deploy the application to production with ease.
- VI. It should support testing framework like mocha, jest and supertest.
- VII. A command line tool to generate CRUD (create, read, update, delete) scripts.



## REFERENCES

- [1] Alexandre Decan, Tom Mens, and Eleni Constantinou. 2018. *On the impact of security vulnerabilities in the npm package dependency network*. In *Proceedings of the 15th International Conference on Mining Software Repositories (MSR '18)*. Association for Computing Machinery, New York, NY, USA, 181–191. DOI:<https://doi.org/10.1145/3196398.3196401>
- [2] Larry L. Smith. 2006. *BASH Shell: Essential Programs for Your Survival at Work: Book 3 in the Rosetta Stone Series for Computer Programmers and Script-Writers* (Rosetta Stone). BookSurge Publishing.
- [3] Lambert M. Surhone, Mariam T. Tennoe, and Susan F. Henssonow. 2010. *XML Template Engine*. Watson-Guptill Publications, Inc., USA.
- [4] Lambert M. Surhone, Mariam T. Tennoe, and Susan F. Henssonow. 2010. *Node.js*. Betascript Publishing, Beau Bassin, MUS.
- [5] Marie Taylor. 2014. *Introduction to JavaScript Object Notation: a to-the-point guide to JSON*. CreateSpace Independent Publishing Platform, North Charleston, SC, USA.
- [6] Mohammed Thakir Mahmood and Osama Ibraheem Ashour Ashour. 2020. *Web Application Based on MVC Laravel Architecture for Online Shops*. In *Proceedings of the 6th International Conference on Engineering & MIS 2020 (ICEMIS'20)*. Association for Computing Machinery, New York, NY, USA, Article 98, 1–7. DOI:<https://doi.org/10.1145/3410352.3410834>
- [7] Paul David Yanzick. 2009. *Web service architecture framework for embedded devices*. Ph.D. Dissertation. Colorado Technical University. Advisor(s) Tim Maifeld. Order Number: AAI3405680.
- [8] Robert E. Noonan and Richard H. Prosl. 2002. *Unit testing frameworks*. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education (SIGCSE '02)*. Association for Computing Machinery, New York, NY, USA, 232–236. DOI:<https://doi.org/10.1145/563340.563429>
- [9] Robert E. Noonan and Richard H. Prosl. 2002. *Unit testing frameworks*. *SIGCSE Bull.* 34, 1 (March 2002), 232–236. DOI:<https://doi.org/10.1145/563517.563429>
- [10] RashidahF Olanrewaju, Thouhedul Islam and Nor'ashikin Bte.Ali. 2015. *An Empirical Study of the Evolution of PHP MVC Framework*. In *Advanced Computer and Communication Engineering Technology - Proceedings of the 1st International Conference on Communication* , pp. 399-410.

[11] Rui Zhong, Yongheng Chen, Hong Hu, Hangfan Zhang, Wenke Lee, and Dinghao Wu. 2020. *SQUIRREL: Testing Database Management Systems with Language Validity and Coverage Feedback*. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*. Association for Computing Machinery, New York, NY, USA, 955–970. DOI:<https://doi.org/10.1145/3372297.3417260>