



CALIFORNIA STATE UNIVERSITY
FULLERTONTM

Department of Computer Science CPSC 597 / 598 PROJECT / THESIS DEFINITION

To the graduate student:

1. Complete a project proposal, following the department guidelines.
2. Have this form signed by your advisor and reviewer / committee.
3. Submit it with the proposal attached, to the Department of Computer Science.

☒ Project

☐ Thesis

Please print or type.

Student Name: Sai Supreeth Kolaparthi Student ID: 885582296
Address: 1208 Deerpark Dr Fullerton 92831
Street City Zip Code
Home Phone: 6577998466 Work Phone: _____
E-Mail: saisupreeth@csu.fullerton.edu Units: 30 Semester: 4

Are you a Classified graduate student?

☒ Yes

☐ No

Is this a group project?

☐ Yes

☒ No

Proposal Date: _____

Tentative Date for Demonstration

/Presentation/Oral Defense: May 5, 2024

Completion Deadline: May 5, 2024

Tentative Title: Web and Cloud Development for DigiClips

We recommend that this proposal be approved:

Faculty Advisor	Dr. Abdul Motin Howlader		
	Printed name	Signature	Date
Faculty Reviewer	Dr. Mikhail Gofman		
	Printed name	Signature	Date
Faculty Reviewer	Dr. Mikhail Gofman		
	Printed name	Signature	Date



Department of Computer Science

This project has been satisfactorily demonstrated and is of suitable form.

This project report is acceptable in partial completion of the requirements for the Master of Science degree in Computer Science.

Web and Cloud Development for DigiClips

Project Title (type)

Sai Supreeth Kolaparthi

Student Name (type)

Dr. Abdul Motin Howlader

Advisor's Name (type)

Advisor's signature

Date

Dr. Mikhail Gofman

Reviewer's name

Reviewer's signature

Date

Table of Contents

<i>Table of Figures</i>	<i>3</i>
<i>1. Abstract</i>	<i>4</i>
<i>2. Introduction</i>	<i>4</i>
2.1 Objectives.....	4
<i>3. Preliminaries</i>	<i>5</i>
<i>4. Implementation</i>	<i>8</i>
<i>5. Architecture Diagram</i>	<i>27</i>
<i>6. Project Scheduling.....</i>	<i>28</i>
<i>7. Achievements.....</i>	<i>29</i>
<i>7. References</i>	<i>30</i>

Table of Figures

<i>Figure 1: Architecture Diagram of the Application.....</i>	<i>27</i>
<i>Figure 2 Architecture Diagram of SFTP Server</i>	<i>28</i>

1. Abstract

This report describes my work in DigiClips.Inc. for my capstone project. The following sections describe the detailed description of the tasks, designing the application/services, architectural diagram, issues faced, and research for the tasks done during my tenure in the DigiClips journey. This document also captures my achievements and personal learning and how they will be helpful to my professional career.

2. Introduction

DigiClips is a startup company specializing in recording online streaming media, live television, and radio. The recordings are then sold to journalists and media companies that use them for their work. The company operates a complete ecosystem of web servers, media processing machine-learning-based applications, and enterprise security solutions such as firewalls, intrusion detection systems, and bastion hosts to perform the recordings.

To maintain optimal performance and provide a faultless user experience, it is critical to stay up to speed with the latest cloud computing solutions and frameworks in the fast-expanding world of digital technology. The project involved upgrading the “Search Engine” Angular application of DigiClips to the latest angular version and deploying it on the AWS cloud. The search engine is comprised of both front-end and back-end, where the front-end is responsible for displaying the results for the users for various inputs (I.e., providing an option to select the station like radio, TV, web, and magazines) and showing the analysis of the media results like the number of hits, sentiment analysis, etc. The backend part of the search engine is responsible for the routes and making calls to the other backend applications.

All development will be guided by Mr. Bob Shapiro, Mr. Henry Bremers from DigiClips, and Prof. Dr. Mikhail Gofman from CSUF. The specific tasks will be assigned by the company, and their exact nature will be protected by a non-disclosure agreement unless DigiClips decides otherwise.

2.1 Objectives

The main objective is to upgrade the application to the latest version of angular, remove vulnerabilities, and make the application ready for production deployment in AWS. The following are the objectives of the project.

1. Fixing vulnerabilities for the dependencies:

This will help the application prevent hacking attacks and exploits for malware and make the application productionable.

2. Migrating from the application from protractor to cypress:

Previously, the protractor was used for the test cases. In the latest Angular versions, the support for the protractor is removed, so migration of Cypress is necessary [4].

3. Upgrading the application from Angular 14 to 15:

It has new features and enhancements, security updates, changes in the code, performance optimization, deprecation of some of the APIs, compatibility, and browser support for new standards [2].

4. Upgrading the application from Angular 15 to 16:

It has new features and enhancements, security updates, changes in the code, performance optimization, deprecation of some of the APIs, compatibility, and browser support for new standards [3].

5. Setup the SFTP server:

One of the major features of a search engine is that users can access the recorded videos along with the transcript. Since DigiClips records media from various sources, huge amounts of data are generated, and using online storage will cost a lot. So, we decided to go with the SFTP server to mount the data over SSH for all the recordings we store at the local network and access this data from AWS [9].

6. Deploying the application on the AWS LightSail:

It will enable application scalability, availability, and ease of integration with other AWS resources. It also provides security features like firewall and networking to set up and manage SSL certificates. It also provides monitoring and metric tools. Overall, it makes the application available to various users across the globe [10].

3. Preliminaries

Before going into the implementation details, let's understand the technologies and frameworks we used in this project. This section will provide a brief overview to the client so they can understand the project we are working on. The following sections provide the top overview of each technology we have used:

Angular Framework:

Angular is a popular open-source front-end web application framework maintained by Google and a community of developers. It is used for building dynamic and interactive web applications. Angular provides a comprehensive toolkit for building single-page applications (SPAs) with features like data binding, dependency injection, reusable components, routing, and more. Here, a single-page application means that each time we call different endpoints, we don't need to call the backend server every time. Rather, everything will be sent to the browser during the compile time, and we can use the routes to switch between components. Again, the component is only a small chunk of code in the application. For example, rather than writing an entire web page in the same file, we divide the entire web page into different components like the navbar, footer, body, sidebar, etc., and these components can interact with each other and are also aligned in a way that a web page looks. So, DigiClips uses Angular for the front-end application for the search engine to communicate with the backend server to fetch the results and display them to the user.

Libraries/Dependencies in Angular:

In Angular, dependencies or libraries refer to external packages or modules that extend the framework's functionality or provide additional features for building web applications. These dependencies are typically managed using package managers like npm (Node Package Manager) or yarn. We used many libraries to support various UI components and fetch results from the backend. (Ex- Angular Forms, HTTP Client, RxJs, Angular Material for UI, etc.)

Express.JS:

Express.js, often referred to simply as Express, is a popular open-source web application framework for Node.js, a server-side JavaScript runtime environment. Express.js provides a minimalistic and flexible set of features for building web applications and APIs. Here, we use Express.JS as a backend server to fetch results from the DB and to make third-party API calls, such as Twitter, etc., to scrape the data.

Protractor and Cypress:

Protractor and Cypress are both popular end-to-end testing frameworks for web applications, but they have some differences in their approach and features.

Protractor:

- Protractor is an end-to-end testing framework specifically designed for Angular and AngularJS applications.
- It is built on top of WebDriverJS, which is a JavaScript implementation of the WebDriver API.

- Protractor uses Selenium WebDriver under the hood to interact with web browsers and control them programmatically.
- Protractor supports writing tests in JavaScript or TypeScript and utilizes Jasmine or Mocha as the testing framework.
- It provides built-in support for Angular-specific features such as automatic waiting for Angular promises and Angular-specific locators like `by.model`, `by.binding`, etc.
- Protractor offers features like automatic synchronization with Angular, which helps handle asynchronous behavior in Angular applications more effectively.
- It is commonly used for testing Angular applications, but it can also be used for non-Angular web applications.

Cypress:

- Cypress is a modern end-to-end testing framework that is not tied to any specific JavaScript framework.
- It runs directly in the browser and executes tests within the same runtime as the application being tested, allowing for fast and reliable testing.
- Cypress provides its own automation engine and does not rely on Selenium WebDriver.
- It is built on top of Mocha and Chai for writing tests, and it provides its own assertion library for making assertions in tests.
- Cypress offers a comprehensive set of features, including automatic waiting for elements, real-time reloads during test execution, network stubbing and spying, time-traveling debugger, and more.
- Cypress has a user-friendly and intuitive interface that allows developers to see the state of their application and tests in real-time as they write and execute tests.
- While Cypress can be used for testing any web application, it is particularly well-suited for modern JavaScript frameworks like React, Vue.js, and Angular.

In summary, Protractor and Cypress are powerful tools for end-to-end testing of web applications, but they have different strengths and approaches. Protractor is tailored specifically for Angular applications and integrates well with Angular-specific features, while Cypress offers a more modern and user-friendly testing experience with its own automation engine and real-time testing capabilities. The choice between the two depends on factors such as the technology stack of your application and your preferences for testing tools and frameworks. One of the main reasons we are

currently not using a protractor is from Angular 16; the angular team decided to remove the support and make it obsolete, so we migrated to Cypress for the test framework.

SFTP Server:

SFTP stands for "SSH File Transfer Protocol." An SFTP server is a secure file transfer server that enables users to securely transfer files between a client and a server over a network. It operates over the SSH (Secure Shell) protocol, which provides strong encryption and authentication mechanisms to protect the confidentiality and integrity of data during transmission.

Some key characteristics of developing an SFTP server are secure communication, authentication, permissions access control for the specified user, and compatibility. Another important reason for choosing the SFTP server over the Samba server is that we can access the data over SSH without using the VPN.

AWS LightSail:

AWS LightSail is a simplified and user-friendly service offered by Amazon Web Services (AWS) that provides a straightforward way to launch and manage virtual private servers (VPS) and other cloud infrastructure resources. It is designed to make it easier for developers, small businesses, and individuals to deploy and manage web applications, websites, and other workloads in the cloud without the complexity and overhead of traditional AWS services. One of the best reasons to choose AWS LightSail is that it is a private server with entire hardware dedicated to the instance, unlike the public server, which shares the resources. It is also easy to be flexible, set the network settings and storage, and is less expensive.

4. Implementation

Since I started working with the company, some of the tasks have been completed, and it's an agile model. Every week (I.e., Every Monday), there will be a status meeting with the company folks to give updates on the previous week's progress, clear any roadblocks for development, and assign new tasks. Following are some of the accomplishments for this semester:

1. Angular environment setup in local and framework learning:

Initially, I set up Node.JS and installed Angular CLI on my local laptop to run the Angular application. Previously, I didn't have any experience in Angular, so they assigned a task to create a tour-of-heroes application from Angular documentation to learn in an effective way. Following is the GitHub link for the project I created by seeing the documentation and watching the YouTube Videos for the tour-of-heroes application [8]:

<https://github.com/saisupreeth97/Angular-tour-of-heros>

The above application will teach some of the core concepts of Angular, like creating a component, creating the services, routing, two-way binding by using the forms module, injecting a service using the component constructor, and making API calls using the HttpClient Module. The above GitHub code also consists of a backend server written in Python using the Flask framework to mock the hero's data. This is one of the best ways to start learning a new framework. Although it won't provide deeper implementation details, it is useful for understanding the code quickly and how the flow works in Angular.

2. Environment setup for DigiClips

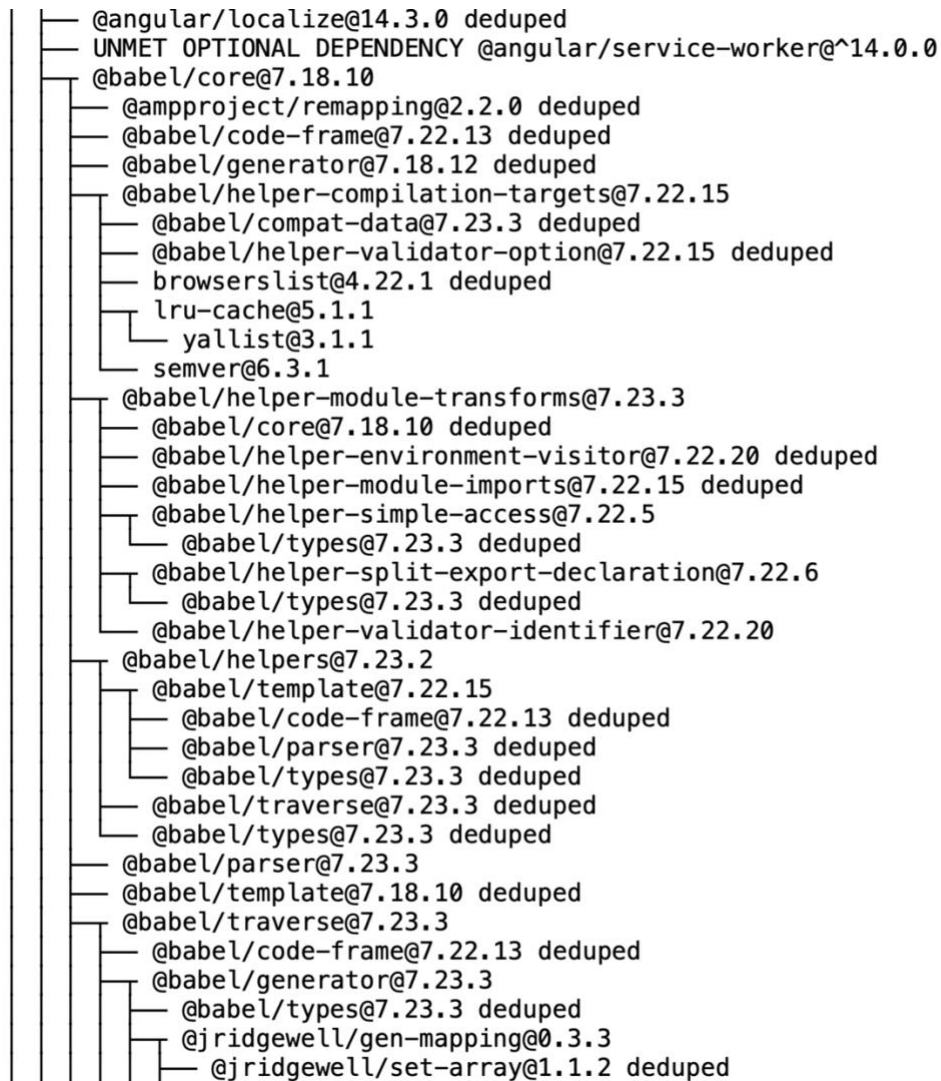
In this phase, I installed the necessary tools required to connect to the company database using SSH connection through VPN. I also got access to the test environment to test and deploy the application.

3. Fixing vulnerabilities for the front-end

Once I got familiar with the working Angular framework, they assigned the task of fixing the vulnerabilities in the front end to make the application more secure and ready to deploy. Here, I made changes to the versions of the node modules in the Package.json file. Also, instead of using the force fix option provided by the Angular framework [7], I tried to override the child modules by generating the project dependency tree and manually picking the modules' versions that don't have vulnerabilities [6]. By using the force method, the dependencies will automatically update to the minor version, which will cause issues in the future, so I overridden the dependencies manually. Following are some of the screenshots for the same:

```
77 + },
78 +   "overrides": {
79 +     "tough-cookie": "4.1.3",
80 +     "semver": "7.5.2"
```

As we can see in the above picture, I used overrides to override the existing dependency version, which will even override the child dependencies. One of the main issues we encounter is finding the child's dependency and checking which parent dependency it imports. Sometimes, this will cause circular dependency as well, so we need to remove dependencies from multiple places as well. To find the parent dependency and also to determine which exact version the dependency is using to override, I generated a dependency tree, and from that, I researched to figure out which version I needed to change. Following are the screenshots for the same:



The above image gives the idea of what the dependency tree looks like, and according to it, we need to make changes in the package.json file. Following is the screenshot for the changes:

50	-	"xlsx": "^0.17.5",
50	+	"xlsx": "https://cdn.sheetjs.com/xlsx-0.19.3/xlsx-0.19.3.tgz",
51	51	"yandex-translate": "^2.1.3",
52	52	"zone.js": "^0.11.8"
53	53	},
@@	-61,7 +61,7 @@	
61	61	"@types/jasminewd2": "^2.0.10",
62	62	"@types/node": "^12.20.55",
63	63	"codelyzer": "^6.0.2",
64	-	"jasmine-core": "~3.6.0",
64	+	"jasmine-core": "~3.8.0",

We have discussed how to fix vulnerabilities, but how to find the vulnerabilities? Let's see that in the following steps. Following is the command to find vulnerabilities in the application:

- `npm audit`

The above command gives the results as follows:

```
Path traversal in webpack-dev-middleware - https://github.com/advisories/GHSA-wr3j-pwj9-hqq6
fix available via `npm audit fix`
node_modules/webpack-dev-middleware
node_modules/webpack-dev-server/node_modules/webpack-dev-middleware

xlsx *
Severity: high
SheetJS Regular Expression Denial of Service (ReDoS) - https://github.com/advisories/GHSA-5pgg-2g8v-p4x9
No fix available
node_modules/xlsx

14 vulnerabilities (11 moderate, 3 high)

To address issues that do not require attention, run:
  npm audit fix

To address all issues possible (including breaking changes), run:
  npm audit fix --force

Some issues need review, and may require choosing
a different dependency.
```

As we can see, it provides a number of vulnerabilities, their severity, and the link for the vulnerability to find the cause. However, it is often better to use the web to find which version has fixed the issue and the cause. The following screenshots demonstrate how to find the issues and the right versions. The screenshots were taken below for xlsx library from web:

VULNERABILITY		VULNERABLE VERSION
	Regular Expression Denial of Service (ReDoS)	*
	Prototype Pollution	*
	Denial of Service (DoS)	<0.17.0
	Denial of Service (DoS)	<0.17.0
	Denial of Service (DoS)	<0.17.0
	Regular Expression Denial of Service (ReDoS)	<0.16.0
	Regular Expression Denial of Service (ReDoS)	<0.12.2

Package versions

1 - 100 of 108 Results

VERSION	PUBLISHED	DIRECT VULNERABILITIES							
0.18.5	24 Mar, 2022	0	C	1	H	1	M	0	L
0.18.4	15 Mar, 2022	0	C	1	H	1	M	0	L
0.18.3	3 Mar, 2022	0	C	1	H	1	M	0	L
0.18.2	14 Feb, 2022	0	C	1	H	1	M	0	L
0.18.1	13 Feb, 2022	0	C	1	H	1	M	0	L
0.18.0	31 Jan, 2022	0	C	1	H	1	M	0	L
0.17.5	10 Jan, 2022	0	C	1	H	1	M	0	L
0.17.4	13 Nov, 2021	0	C	1	H	1	M	0	L
0.17.3	13 Oct, 2021	0	C	1	H	1	M	0	L
0.17.2	15 Sep, 2021	0	C	1	H	1	M	0	L
0.17.1	18 Aug, 2021	0	C	1	H	1	M	0	L
0.17.0	13 Mar, 2021	0	C	1	H	1	M	0	L

By referencing all the above images, we can choose the right version for the dependency and decide whether we should look for an alternative for the dependency:

4. Angular 14 to 15 Upgrade

To upgrade the application to Angular 15, follow the official migration documentation by changing the Angular cli version, NPM, and minor code tweaking. This step will improve the application's performance, fix bugs, leverage new features, enhance security, integrate with other libraries, and extend support. This also helped fix some vulnerabilities, as some dependencies require a newer version of NPM [2].

The following are the steps for upgrading the application to Angular 15:

- Firstly, we need to make sure the Node.JS version is above 14.20.x
- Second, we need to make sure that the typescript version is 4.8 or above
- Once the above requirements are met, run the following Angular CLI command:
 - `ng update @angular/core@15 @angular/cli@15`

Once we ran the CLI command, we faced no issues except one import syntax for the styles.css file. In the latest version, they change the import syntax for CSS files. Following are the screenshots for changes to upgrade Angular 15:

```

@@ -95,7 +95,7 @@ body {
95     }
96
97     /* Importing Bootstrap SCSS file. */
98     - @import '~bootstrap/scss/bootstrap';
98     + @import 'node_modules/bootstrap/scss/bootstrap';
99
100    /* Importing Bootstrap SCSS file. */
101    - @import '~bootstrap/scss/bootstrap';
101    + @import 'node_modules/bootstrap/scss/bootstrap';

```

```

@@ -12,15 +12,15 @@
12     },
13     "private": true,
14     "dependencies": {
15     - "@angular-devkit/core": "^14.2.10",
16     - "@angular/animations": "^14.2.12",
17     - "@angular/common": "^14.2.12",
18     - "@angular/compiler": "^14.2.12",
19     - "@angular/core": "^14.2.12",
20     - "@angular/forms": "^14.2.12",
21     - "@angular/platform-browser": "^14.2.12",
22     - "@angular/platform-browser-dynamic": "^14.2.12",
23     - "@angular/router": "^14.2.12",
15     + "@angular-devkit/core": "^15.2.10",
16     + "@angular/animations": "^15.2.10",
17     + "@angular/common": "^15.2.10",
18     + "@angular/compiler": "^15.2.10",
19     + "@angular/core": "^15.2.10",
20     + "@angular/forms": "^15.2.10",
21     + "@angular/platform-browser": "^15.2.10",
22     + "@angular/platform-browser-dynamic": "^15.2.10",
23     + "@angular/router": "^15.2.10",
24     "angular-file-saver": "^1.1.3",
25     "sass-compiler": "0.0.11",
26     "bootstrap": "^4.6.2",
@@ -52,11 +52,11 @@
52     "zone.js": "^0.11.8"
53   },
54   "devDependencies": {
55     - "@angular-devkit/build-angular": "^14.2.10",
56     - "@angular/cli": "^14.2.10",
57     - "@angular/compiler-cli": "^14.2.12",
58     - "@angular/language-service": "^14.2.12",
59     - "@angular/localize": "^14.2.12",
55     + "@angular-devkit/build-angular": "^15.2.10",
56     + "@angular/cli": "^15.2.10",
57     + "@angular/compiler-cli": "^15.2.10",
58     + "@angular/language-service": "^15.2.10",
59     + "@angular/localize": "^15.2.10",
60     "@types/jasmine": "^3.6.11",
61     "@types/jasminewd2": "^2.0.10",
62     "@types/node": "^12.20.55",

```

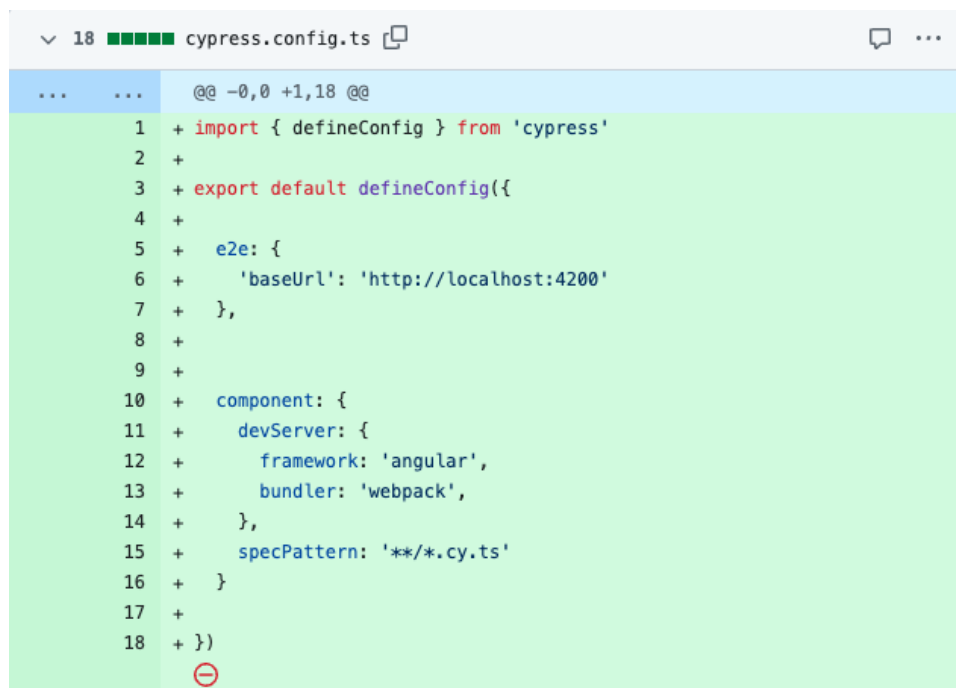
5. Protractor to Cypress migration

Protractor is used to run the test cases in previous versions of Angular, But Angular removed support for protractor from Angular 16 onwards, so upgrading the application is one of the significant steps. I successfully migrated to Cypress by replacing the config files and spec files (I.e., used to write test cases) and successfully executed the automation of the login flow[5].

The following are the steps to install cypress:

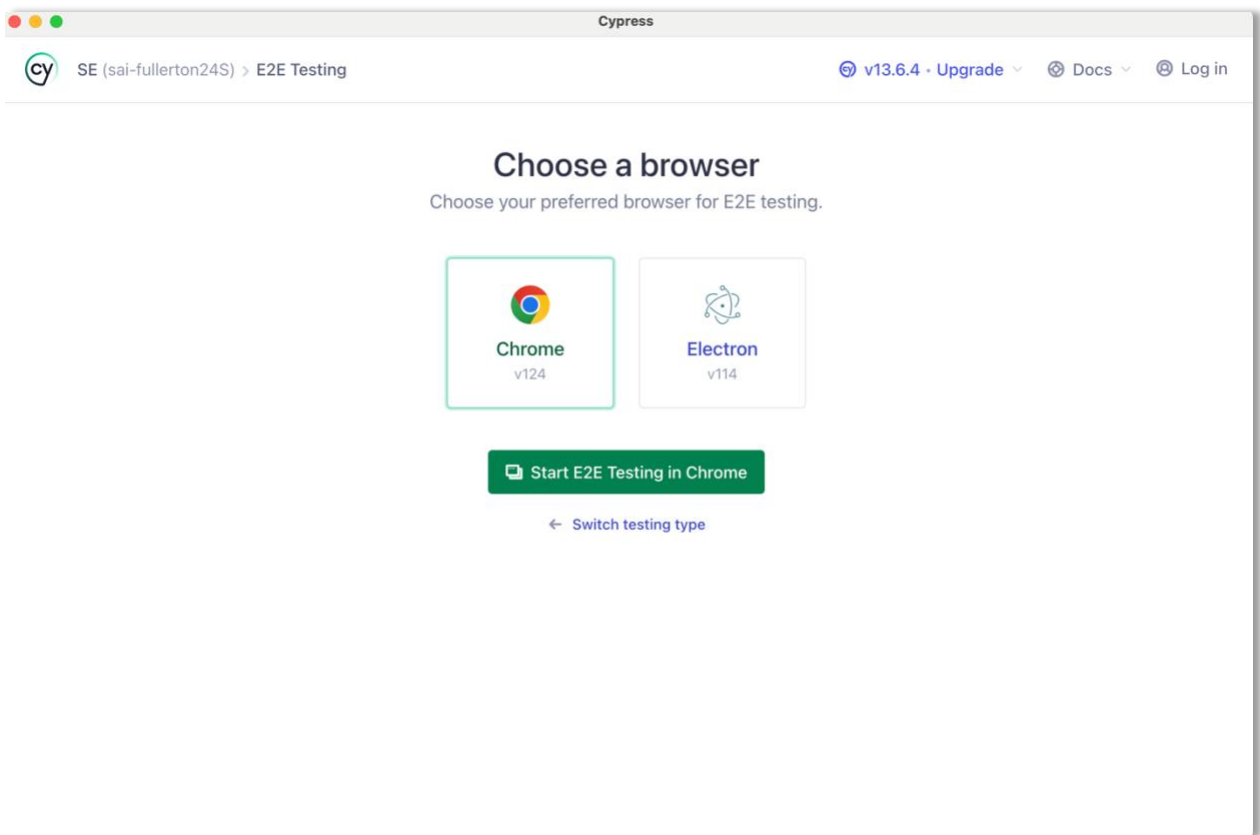
- First, run the following CLI command
 - `ng add @cypress/schematic@latest`
- After running the above command, it will automatically update the e2e configuration in the `angular.json` file
- Once after that, we need to update the `cypress.config.ts` file according to our application needs.
- Once after that, we need to change the spec file (I.e., test cases) code from Protractor to Cypress using the Cypress functions, and we can run using the `ng e2e` command in the browser
- Once the above steps are done, we can completely remove the Protractor from the `package.json` file

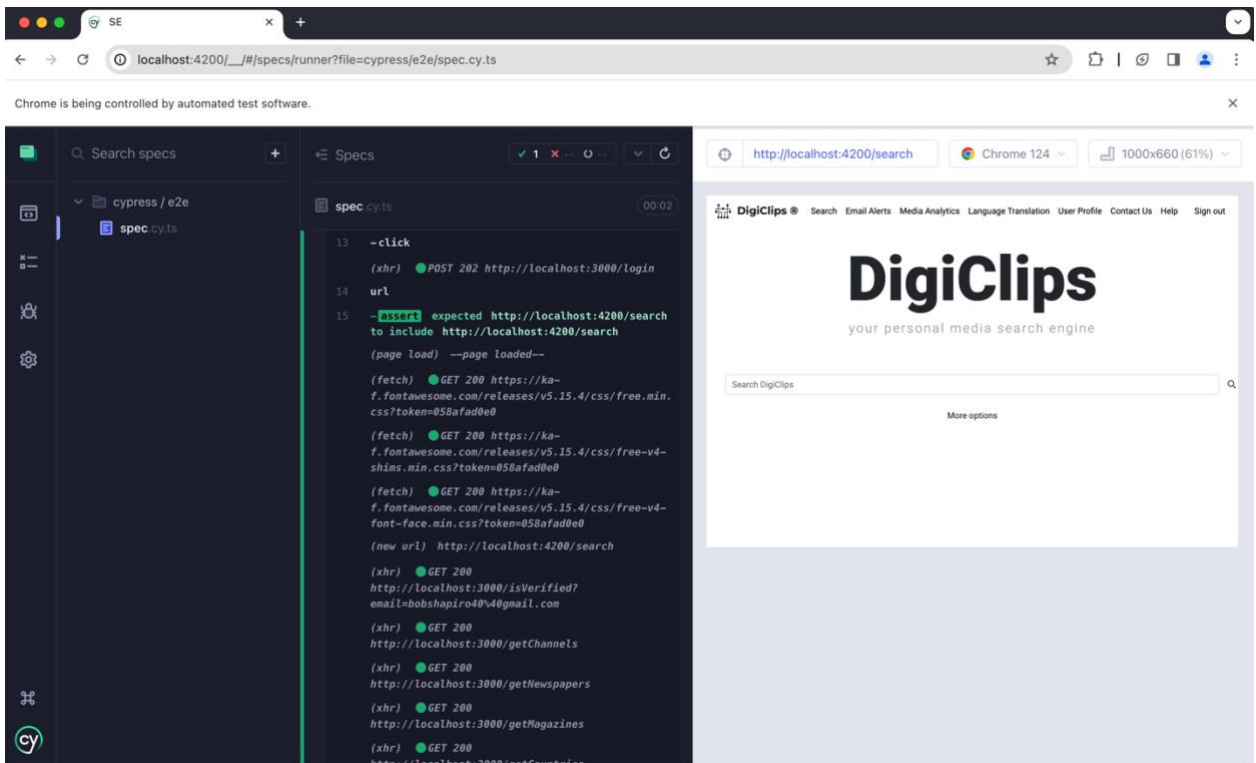
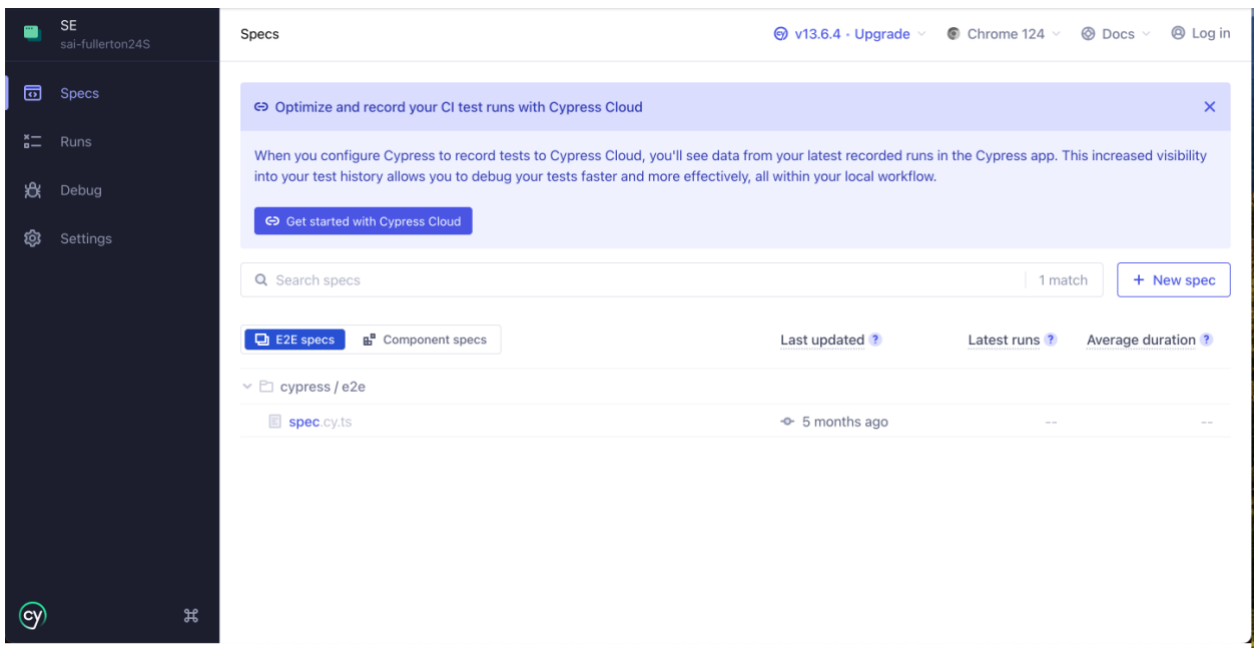
The following are the screenshots for the above steps:



```
18 cypress.config.ts
...
1 + import { defineConfig } from 'cypress'
2 +
3 + export default defineConfig({
4 +
5 +   e2e: {
6 +     'baseUrl': 'http://localhost:4200'
7 +   },
8 +
9 +
10 +   component: {
11 +     devServer: {
12 +       framework: 'angular',
13 +       bundler: 'webpack',
14 +     },
15 +     specPattern: '**/*.cy.ts'
16 +   }
17 + }
18 + })
```

```
12 cypress/e2e/spec.cy.ts
@@ -0,0 +1,12 @@
1 + describe('', () => {
2 +   it('Visits the initial project page', () => {
3 +     cy.visit('/')
4 +     cy.get('#email-input').should('not.be.disabled')
5 +     cy.get('#email-input').type('')
6 +     cy.get('#password-input').should('not.be.disabled');
7 +     cy.get('#password-input').type('')
8 +     cy.get('#check1').check()
9 +     cy.get('#login-btn').click()
10 +    cy.url().should('include', 'http://localhost:4200/search')
11 +  })
12 + })
```





69	72	"karma-coverage-istanbul-reporter": "~3.0.2",
70	73	"karma-jasmine": "^4.0.2",
71	74	"karma-jasmine-html-reporter": "^1.7.0",
72	-	"protractor": "~7.0.0",
73	75	"snapdragon": "^0.8.2",
74	76	"ts-node": "~8.0.2",
75	77	"tslint": "~6.1.0",
76	-	"typescript": "4.8.4"
	78	"typescript": "4.8.4",
	79	"cypress": "latest"

4. Angular 15 to 16 Upgrade

In the next two consecutive weeks, I upgraded the application to Angular 15 following the official migration documentation by changing the angular cli version, NPM, and minor tweaking of the code. This step will improve the application's performance, fix bugs, leverage new features, enhance security, integrate with other libraries, and extend support. This also helped fix some vulnerabilities, as some dependencies require a newer version of NPM [3]. It is similar to the Angular 15 upgrade but faces a few issues; let's discuss it in the following steps.

The following are the steps for upgrading the application to Angular 15:

- Firstly, we need to make sure the Node.JS version is above 16
- Second, we need to make sure that the typescript version is 4.9.3 or above
- Once the above requirements are met, run the following Angular CLI command:
 - `ng update @angular/core@16 @angular/cli@16`
- After this, we also need to upgrade the zone.js library to 0.13.x above version

This will upgrade the packages in the package.json file to the Angular 16 version. However, during this upgrade, other issues were faced. The following are the issues faced:

- Some of the libraries are incompatible with the AOT compiler
- Some of the modules are now moved to the provider's section to provide these packages using dependency injection rather than as an import

Before, Angular was using the JIT compiler, which everything will be compiled on the client side (I.e., browser), but after Angular 12 onwards, they came up with a new compiler called Ivy, which internally uses the AOT compiler (Ahead of Time Compiler). Now, in Angular 16, by default, AOT is enabled, which means all files are compiled on the server side and sent the compiled bundle to the browser so that the browser does not need to compile. However, in our application, some libraries still don't support the AOT compiler. After discussing with Henry, we decided to disable

the AOT compiler and go ahead. The following screenshots show some of the important changes made for the Angular 16 upgrade:

17	-	"@angular-devkit/core": "^15.2.10",
18	-	"@angular/animations": "^15.2.10",
19	-	"@angular/common": "^15.2.10",
20	-	"@angular/compiler": "^15.2.10",
21	-	"@angular/core": "^15.2.10",
22	-	"@angular/forms": "^15.2.10",
23	-	"@angular/platform-browser": "^15.2.10",
24	-	"@angular/platform-browser-dynamic": "^15.2.10",
25	-	"@angular/router": "^15.2.10",
17	+	"@angular-devkit/core": "^16.2.12",
18	+	"@angular/animations": "^16.2.12",
19	+	"@angular/common": "^16.2.12",
20	+	"@angular/compiler": "^16.2.12",
21	+	"@angular/core": "^16.2.12",
22	+	"@angular/forms": "^16.2.12",
23	+	"@angular/platform-browser": "^16.2.12",
24	+	"@angular/platform-browser-dynamic": "^16.2.12",
25	+	"@angular/router": "^16.2.12",
26	26	"angular-file-saver": "^1.1.3",
27	27	"ass-compiler": "0.0.11",
28	28	"bootstrap": "^4.6.2",
@@ -51,14 +51,14 @@		
51	51	"uuid": "^8.3.2",
52	52	"xlsx": "https://cdn.sheetjs.com/xlsx-0.19.3/xlsx-0.19.3.tgz",
53	53	"yandex-translate": "^2.1.3",
54	-	"zone.js": "^0.11.8"
54	+	"zone.js": "^0.13.3"

angular.json		
@@ -12,6 +12,7 @@		
12	12	"builder": "@angular-devkit/build-angular:browser",
13	13	"options": {
14	14	"outputPath": "dist",
15	+	"aot": false,
15	16	"index": "src/index.html",
16	17	"main": "src/main.ts",
17	18	"tsConfig": "src/tsconfig.app.json",
@@ -204,7 +205,6 @@		

Apart from the above changes, Angular has changed some of the library's scope, meaning we need to change from import to providers. In Angular, we have three scopes: declaration, imports, and providers. Declarations are used for the internal components, and pipes are available for the other components, whereas imports are used to make external import components available to the internal components. Providers are used for dependency injection; rather than creating a new object directly, we create and inject the object when required. This will make application loosely coupled and easy to manage. Following are the changes made in the application:

↓	@@ -60,16 +60,9 @@ import { BrowserModule } from '@angular/platform-browser/animations';
60 60],
61 61	imports: [
62 62	BrowserModule,
63 -	AppRoutingModule,
64 63	HttpClientModule,
64 +	AppRoutingModule,
65 65	FormsModule,
66 -	ChartsModule,
67 -	NgMultiSelectDropDownModule,
68 -	CollapseModule.forRoot(),
69 -	AccordionModule.forRoot(),
70 -	BsDatepickerModule.forRoot(),
71 -	TimepickerModule.forRoot(),
72 -	ButtonsModule.forRoot(),
73 66	BrowserAnimationsModule
74 67],
75 68	providers: [
↑	@@ -83,12 +76,20 @@ import { BrowserModule } from '@angular/platform-browser/animations';
83 76	EmailAlertsService,
84 77	AddMediaService,
85 78	CustomizeLogoService,
79 +	ChartsModule,
80 +	NgMultiSelectDropDownModule,
81 +	CollapseModule,
82 +	AccordionModule,
83 +	BsDatepickerModule,
84 +	TimepickerModule,

5. Setting up the SFTP Server

All recorded videos and captions for them are stored in the local network using high-storage SSDs. Previously, we used a CIFS server (i.e., samba server) to access the data from the network and mount the data where the front-end application is hosted. One of the major problems with the CIFS server is that we can access data only in the network, so we were using a VPN to access data. Another issue with the CIFS server is that it's very slow to access the data, which causes a network overhead to access using a VPN. Initially, while deploying the application in AWS, I used OpenVPN CLI for VPN to access the data VPN, but the AWS shell crashed constantly and froze because of network changes inside, and we couldn't access AWS using SSH.

So, after research, I came up with the idea of an SFTP server to access the data through the internet using SSH without a VPN. So, the following are the steps to create an SFTP server [9]:

Here, we are creating the SFTP server, so all the following commands are Linux commands.

- First, run **sudo apt update** and **sudo apt upgrade** commands to update the libraries and also to upgrade the software, although it's not mandatory I would highly recommend following this step
- Secondly, check if the **sshd_config** file exists under **/etc/ssh**, but if the file doesn't exist, run the following command:
 - **sudo apt install ssh**
- Once after that, make sure to run the commands as the root user. For this, we can use the **sudo su** command
- Next, we need to create a user for the SFTP login. First, we need to create a group to keep track of the users. Use the following command to create a new group
 - **groupadd sftpgroup**
- Next, we need to add a user under sftpgroup, but we don't want to give access to the user for login using SSH, so for that, first we need to find where no login is located, so for that use the following command:
 - **which nologin**
- Now, we can add the user using the following command
 - **useradd -g sftpgroup -s /usr/sbin/nologin username**

Here sftpgroup is the group name that we created in the previous step, and /usr/sbin/nologin is the path that we found out using which nologin command
- Now, we need to assign password for the created user; for that, use the below command:
 - **passwd username**
- We must modify the **sshd_config** file under the **/etc/ssh** folder. We need to add a match group to check if the user matches the group and what action needs to be performed. The following are the changes:

Match Group sftpgroup

ChrootDirectory /mnt/

ForceCommand internal-sftp -d /files

AllowAgentForwarding no

AllowTcpForwarding no

X11Forwarding no

PermitTunnel no

PasswordAuthentication yes

Here, we need to mention the root directory to which we need to give access for that **ChrootDirectory** is used. To mention which folder we need to give access to for the particular user, we need to use **ForceCommand** and specify the path for the folder (i.e., /files). Then, as we can **PasswordAuthentication** saying yes, we can access the SFTP server with username and respective password.

- Once after making the changes restart the SFTP server using the following command:
 - **systemctl restart sshd**

To make it more secure, we went ahead with one more layer of security so that we can access the SFTP server only with a private key (.pem file) just like AWS, and a passphrase also needs to be given along with the private key to validate. To add authentication using public and private key, follow the below steps:

- First, we need to generate private and public key using the following command
 - **sudo ssh-keygen**
- Once we execute the command, it will generate public and private keys under the .ssh folder, just like shown in the screenshot below:

```
henry@Digi62:~$ ls .ssh/  
id_rsa  id_rsa.pub  known_hosts  known_hosts.old  
henry@Digi62:~$
```

- Now, create a new folder like this **mkdir /etc/ssh/authorized_keys/**
- Once after creating the folder, change the owner of the folder using chown command for that use the following command:
 - **chown root:root /etc/ssh/authorized_keys/**
- After this change the permission of the folder using chmod command:
 - **chmod 755 /etc/ssh/authorized_keys/**
- Once after that, add the public key to the created folder by using the following command:

- **echo 'public-key value' >> /etc/ssh/authorized_keys/username**
- Now again change the permission for the folder the we created under username for that use the following command:
 - **chmod 644 /etc/ssh/authorized_keys/awsdaemon**
- Once after that again modify the sshd_config file and with the following commands:

Match Group sftpgroup

ChrootDirectory /mnt/

ForceCommand internal-sftp -d /files

AuthorizedKeysFile /etc/ssh/authorized_keys/%u .ssh/authorized_keys

PermitRootLogin no

PermitEmptyPasswords no

AllowAgentForwarding no

AllowTcpForwarding no

X11Forwarding no

PermitTunnel no

PasswordAuthentication no

As we can see, few lines are added for the above code; one important one is the AuthorizedKeysFile where we provide the path for the public and private keys. Also, we need to make password authentication no as we are now using the private key for authentication.

- Once after making the changes again restart the SFTP server using the following command:
 - **systemctl restart sshd**

Once everything is done, we need to port-forward the from the router to the local machine port where the SFTP server is running. After that, we can access the data using the public IP address with username and private key file. Following is the command for the same:

- **sshfs -p 29172 -o IdentityFile=/etc/ssh/sftp_auth_keys/id_rsa username@ip_address:/files /home/henry/codentv1c/**

Here, IdentityFile is the path for the private key, and at the end /home/henry/codentv1c is the folder on the client side where we need to mount the data.

Overall, this provides secure access to the files stored in the local network from the AWS. Once data is mounted, the front-end application can take video, audio, and transcript files from the mounted folder, and any data updates on the server side will also be reflected here.

6. Deploying application into AWS LightSail

We deployed both the front-end and back-end applications in the AWS LightSail to make them available to the users. To deploy, first, we need to create a new instance by selecting the technology (I.e., in our case, it's Node.JS) and memory, RAM. Once an instance is created, we need to assign the static IP address because every time we restart the instance, it will assign a random IP address if we don't assign a static address. To assign a static IP address, go under the Networking tab and click Assign a static IP address. After an instance is up and running, we can open the SSH terminal and, under the htdocs folder, clone our repository by using **git clone**. Once the code is cloned, we can use the npm commands to run the application. Again, there are two ways to access the application's front end. One way is to bind the application with the IP address 0.0.0.0 to bind all the available IP addresses in the host. However, to access the application using the IP address, we need to add a custom rule under the Networking tab to expose the port we are running. This is not a good idea because users can't access it with a port number, so we need to implement it another way. To access the application with an HTTP or HTTPS port, we need to proxy the request. To do that, we need to perform some of the actions on the Bitnami server we use. The following screenshots show the steps for the same [11]:

- Create and edit the `/opt/bitnami/apache/conf/vhosts/myapp-http-vhost.conf` file and add the following lines:

```
<VirtualHost _default_:80>
  ServerAlias *
  DocumentRoot "/opt/bitnami/projects/myapp/public"
  <Directory "/opt/bitnami/projects/myapp/public">
    Require all granted
  </Directory>
  ProxyPass / http://localhost:3000/
  ProxyPassReverse / http://localhost:3000/
</VirtualHost>
```

- Create and edit the `/opt/bitnami/apache/conf/vhosts/myapp-https-vhost.conf` file and add the following lines:

```
<VirtualHost _default_:443>
    ServerAlias *
    SSLEngine on
    SSLCertificateFile "/opt/bitnami/apache/conf/bitnami/certs/server.crt"
    SSLCertificateKeyFile "/opt/bitnami/apache/conf/bitnami/certs/server.key"
    DocumentRoot "/opt/bitnami/projects/myapp"
    <Directory "/opt/bitnami/projects/myapp">
        Require all granted
    </Directory>
    ProxyPass / http://localhost:3000/
    ProxyPassReverse / http://localhost:3000/
</VirtualHost>
```

- Restart the Apache server:

```
sudo /opt/bitnami/ctlscript.sh restart apache
```

The above screenshots provide a way to create a custom virtual and proxy the HTTP request of port 80 and HTTPS request of 443 to the internal IP address of the custom port. But even after doing that, we encountered another issue: web socket forwarding. Basically, Angular relies on web sockets for two-way binding, which means any changes in the browser will reflect in the application, and the opposite is the same. To proxy the WebSocket, we need to use the following code:

RewriteEngine on

RewriteCond %{HTTP:Upgrade} websocket [NC]

RewriteCond %{HTTP:Connection} upgrade [NC]

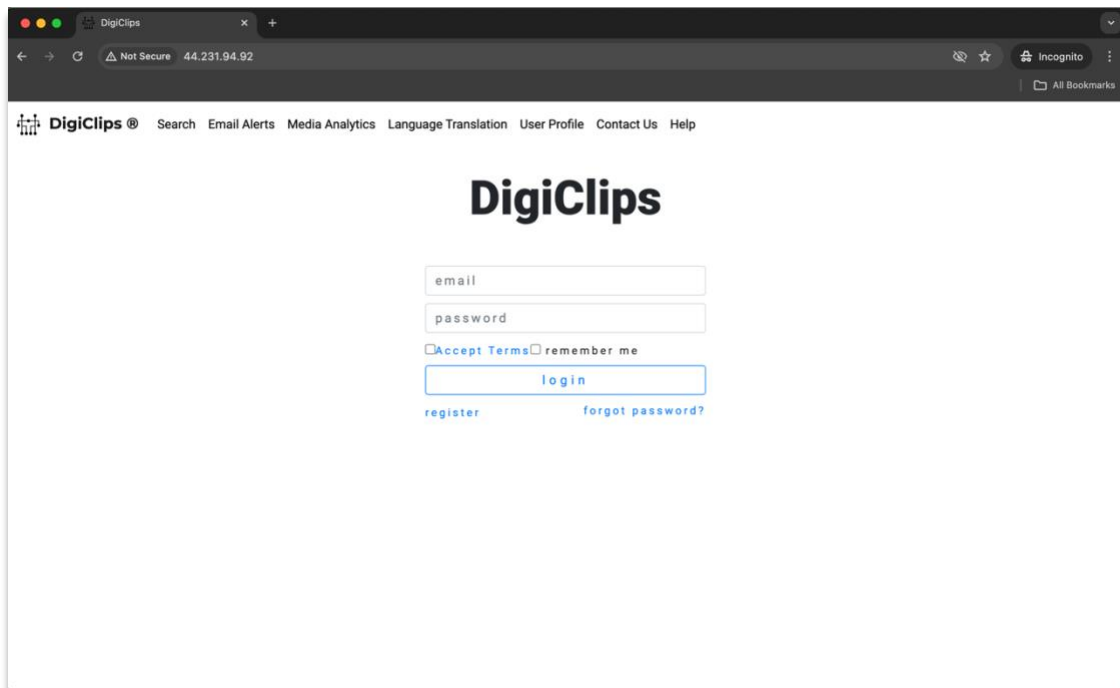
RewriteRule /(.*) ws://localhost:3000/\$1 [P,L]

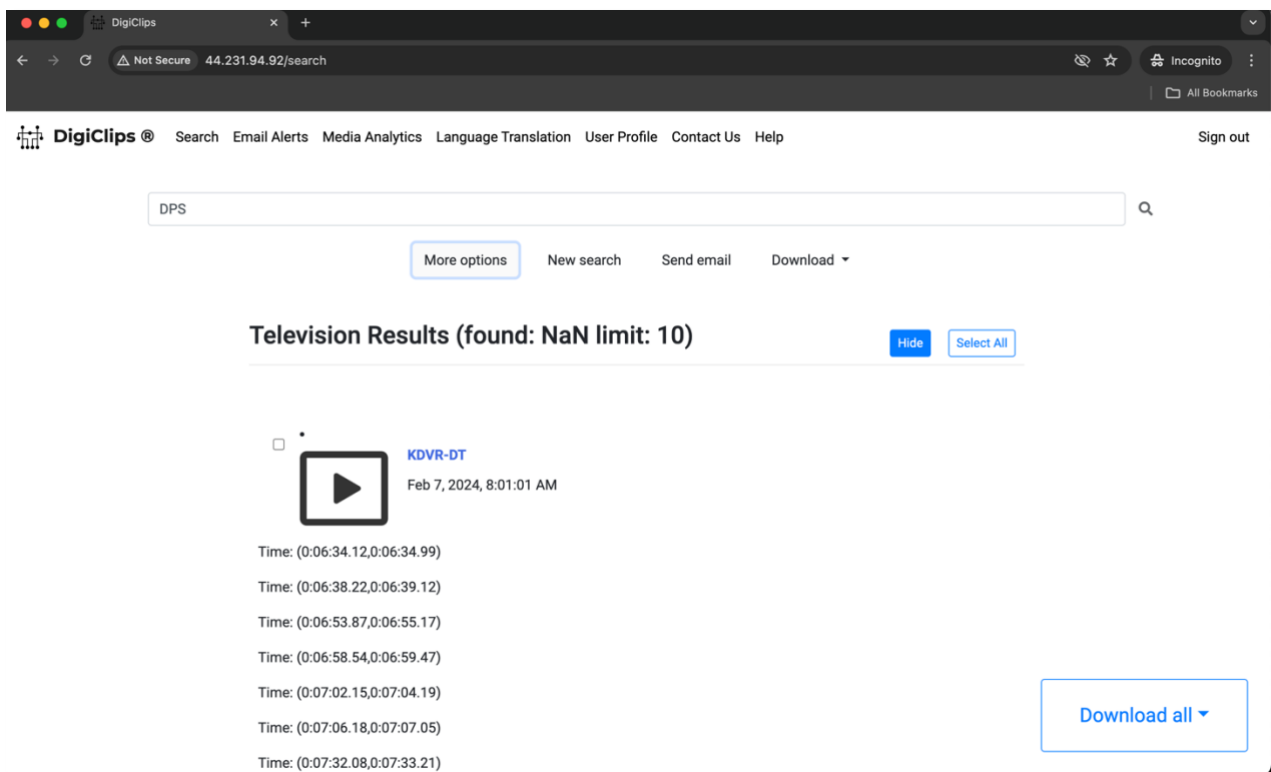
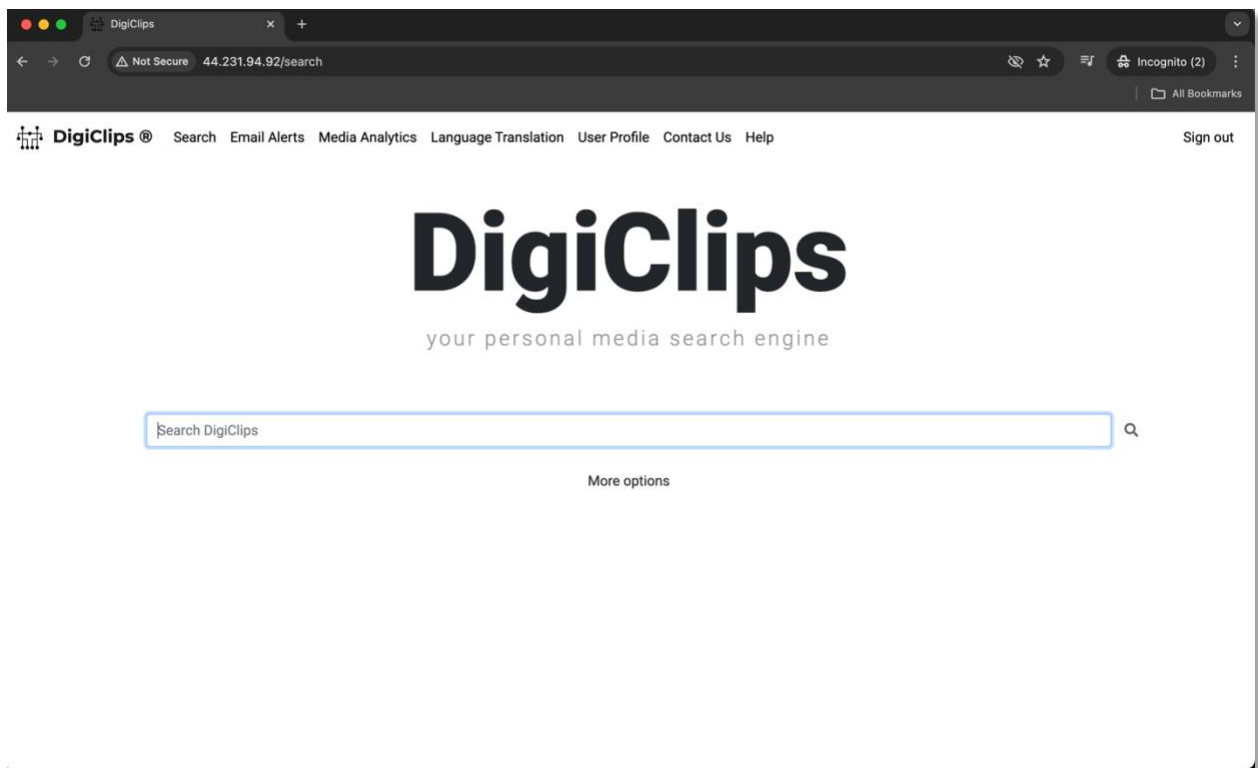
As we can see from the above code, this RewriteRule will forward the request to the WebSocket of port 3000. Once the above step is done, we need to start running the application, but the problem here is if we run the application in the SSH terminal, after some time, the terminal gets closed, and the application will be aborted. To overcome this issue and to use multiple terminals, we came up with installing tmux on AWS. This will allow us to run the terminal in the background and to use

multiple terminals at the AWS. But I figured out there is another way to bring up the application without using Tmux to get the application to keep running in the background forever. The following command use shows how to run the application in the background:

- **forever start node_modules/@angular/cli/bin/ng.js serve --host 0.0.0.0**

As we can see, we are binding the application to 0.0.0.0, in which all the IP addresses are assigned to the applications. As mentioned in the above steps, we can also remove the host parameter, as we are using a custom virtual host as a proxy. The following are the screenshots for the deployed application on AWS:





5. Architecture Diagram

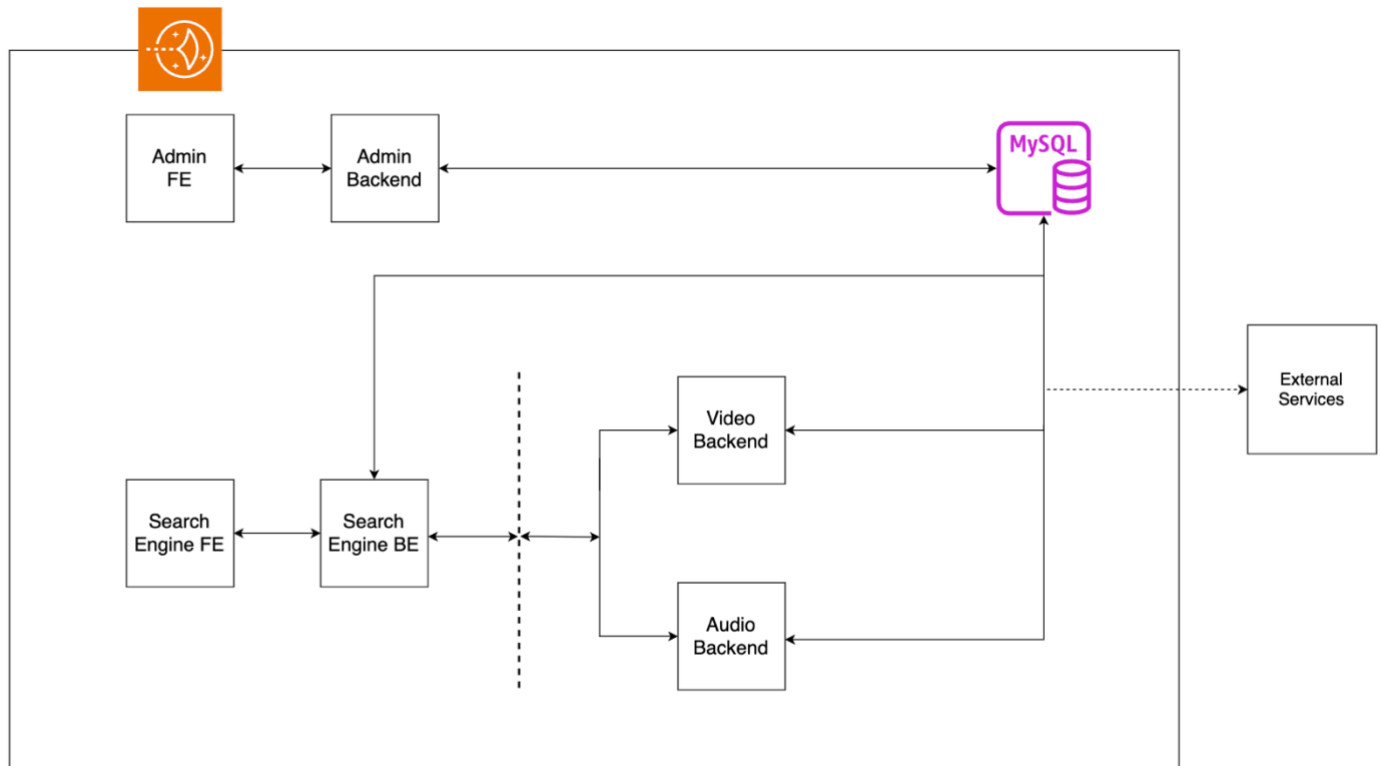


Figure 1: Architecture Diagram of the Application

As we can see in the above diagram, the search engine is responsible for getting the results from the database and displaying them to the user. Admin service is responsible for adding new users and giving authorization roles to the user so that, based on the access level, the other applications will allow the user to enter the application. Also, the two backend applications are the core. They process the results using machine learning algorithms to convert speech-to-text and video-to-text and post the results to the database. Here, these applications are deployed in the AWS LightSail. Here, the search engine will communicate with video and audio backend for media and get the text for the captions from the database.

Next, we also see the architecture diagram of the SFTP server how it communicates with the AWS to mount the data. The following diagram depicts the same:

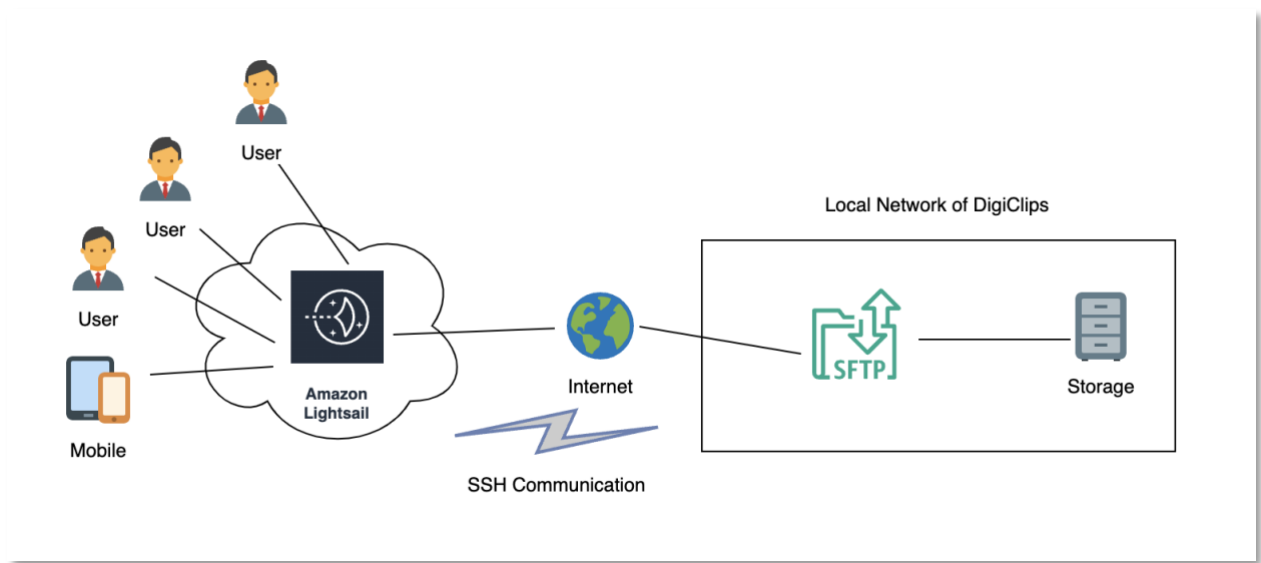


Figure 2 Architecture Diagram of SFTP Server

As we can see in the diagram, we are accessing the data (i.e., recorded videos, audio, and transcripts) from the local network of DigiClips over the internet through SSH. From AWS, the users can access those videos using the Angular front-end application, a search engine.

6. Project Scheduling

2023	October				November				December				Summary	
Tasks:	1	2	3	4	1	2	3	4	1	2	3	4	Hours	Percent
Setting up Angular Environment		8											8	8%
Create Tour-of-Heroes App			12	12									24	24%
Fixing Vulnerabilities for SE Frontend					8	8							16	16%
Angular 14 to 15 Upgrade							12	12					24	24%
Protractor to Cypress Migration									15				15	15%
Seminar Proposal Report										8	5		13	13%
Hours	0	8	12	12	8	8	12	12	15	8	5	0	100	100.0%

2024	January				February				March				April					Summary	
Tasks:	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	5	Hours	Percent
Fixing Vulnerabilities for SE Backend			15															15	8.24%
Angular 15 to 16 Upgrade				12	12													24	13.1%
Discovery and Learning of AWS LightSail						12	12	15										39	21.4%
Creating SFTP Server									15	15	15	15						60	32.9%
AWS Light Sail Instance Creation and Research													8	8				16	8.7%
Deployment and Fixing Issues in the Application															12	12		24	13.1%
Demonstrate and Report																	4	4	2.19%
Hours	0	0	15	12	12	12	12	15	15	15	15	15	8	8	12	12	4	182	100.0%

7. Achievements

- Able to explore new technologies, tools, and frameworks like Angular, Cypress, AWS Light Sail, etc.
- Learned about a few networking concepts and SFTP, samba servers
- Able to meet and collaborate with new people from various colleges.
- I Gained academic and professional experience.

7. References

1. Personal Communication
2. <https://update.angular.io/?l=3&v=14.0-15.0>
3. <https://update.angular.io/?l=3&v=15.0-16.0>
4. <https://docs.cypress.io/guides/end-to-end-testing/protractor-to-cypress>
5. <https://docs.cypress.io/guides/getting-started/installing-cypress>
6. <https://stackoverflow.com/questions/64605805/npm-force-resolutions-not-working-when-installing-a-new-package>
7. <https://stackoverflow.com/questions/58457748/how-to-fix-these-vulnerabilities-npm-audit-fix-fails-to-fix-these-vulnerabilit>
8. <https://www.youtube.com/watch?v=5hJFaiiDU4&list=PLJJcOjd3n1Zdl4rfWuy9UsKDul5Sj5RHm>
9. https://www.youtube.com/watch?v=pS_mY4Rfsq8
10. <https://www.youtube.com/watch?v=rtshCulV2hk>
11. <https://docs.bitnami.com/aws/infrastructure/nodejs/administration/create-custom-application-nodejs/>