# File Statistic Assignment:

**Owner: Venkata Prasanna verma n**

**Mail: Prasanna224466@gmail.com**

**Hi team,**
**Have added Basic Details of the Assignment, If Stuck anywhere or facing issues can please reach out.**
**This task was carried with focus given objectives due to limited time. It can very well be extended into full grown project overtime. Code written is easily extendible at Frontend and Backend to Other file Types as well.**

**Any logic correction in output should be minor change in code if any .**

The Following Assignment caters to the following Functionalities:

1.Accept a file

2.Perform validations on it(for now .txt extension)

3.Process the Text file in chunks for large file with Multithreading to make it faster, wherein the thread pool size can be configured.

4.The file is processed to generate result on:

1. Total count of words:
    a. **Criteria to qualify as word**:
        - Should contain atleast one letter( a,abc,abc@, #ansd, asd4jj)
        - Not a word( 333, @# , @ - just symbols or numbers and both – not considered in count)
2. Count Total Letters in file
3. Get 3 Top used words and 3 Top used Letter

5. Validations Handled:

1. File Size can be configured in application.properties.

2. Empty File

3. Invalid File Extension

**Tech Used:**

**Front End** : Html, CSS, javascript

**Backend** : Java 17 ,SpringBoot,Maven (Build Tool) ,In memory Caching ,Junits ,Mockito – Unit tests

**IDE** : Intellij(Backend), Visual Studio(Front end)

**Documentation Tool** – **OpenApi (Swagger 3.0)** – Details api documentation and api execution on backend server url itself.

**Concepts Used :**

Multithreading,

Chunking (breaking to chunk sizes)

Code Extendable to support files of different formats – currently implementation has Text File Only.
Design is made Generic to easily extend the code than changing existing code logic.
Design patterns used: Factory Pattern, Builder Pattern and few more intrinsically.

**Steps to run the Project:**

1. **Can import Spring Project in any Java IDE preferably IntelliJ.**
   No need to setup Db or external configurations just have right JDK installed and build the code and run it.
   **Tomcat** server by default runs in port 8080. Url http: localhost:8080/
   **Swagger url** : http://localhost:8080/swagger-ui/index.html
   Also properties to limit file Size and Process Chunk size and few others are configurable in application.properties file. We can use redis , db and much more to enhance it further for real life use cases.
   CORS is enabled by default for Front end to access without security checks.
   We can logins ,authentications and authorizations and build entire system.
   **I have strictly limited to the assignment objectives only with time constraint at hand, But Framework is designed to be easily extendible for various fileTypes(eg : csv etc).**

2. **To run Front End we can use visual Studio ,**
   If nodejs or npm not installed no issue.

   Else

   - Can simply Download Live Server extension in Visual Studio.
     Open the index.html file and click on run server.This server by default runs on 5500 port and opens browser automatically.
   - If not also we run following commands post installing nodeJs like any other ui project to locally run ui builds:
     npm i to install liver server dependency and then run npm start command

   **We can execute API's from Both FrontEnd and Swagger url.**

   **Api's (Keeping to bare minimum of functionality – in request params)**
   **Deatiled Documentation in Swagger Url, once backend is Run for easy access.**

   ### 1 .Simple File Upload API(POST) :

   Working Postman Curl :

   ```
   curl --location 'http://localhost:8080/api/v1/file/upload' \
   --form 'file=@"/C:/Users/Priya N/Documents/TestingProject'\''.txt"'
   ```

   **Sample Respones:**
   200 OK – file Processed SuccessFully

```
{
  "wordCount": 35,
  "letterCount": 100,
  "symbolCount": 27,
  "topWords": [
    "hi",
    "prasanna",
    "ab@"
  ],
  "topLetters": [
    "a",
    "b",
    "s"
  ]
}
```


**For Error Case Example 1 : Incorrect File Extension:**


```
{
    "errorCode": "1009",
    "errorMsg": "Invalid File Extension / Not Supported.Supported Types(txt)",
    "wordCount": 0,
    "letterCount": 0,
    "symbolCount": 0
}
```


## API 2:

**2. File Upload History API(GET)**

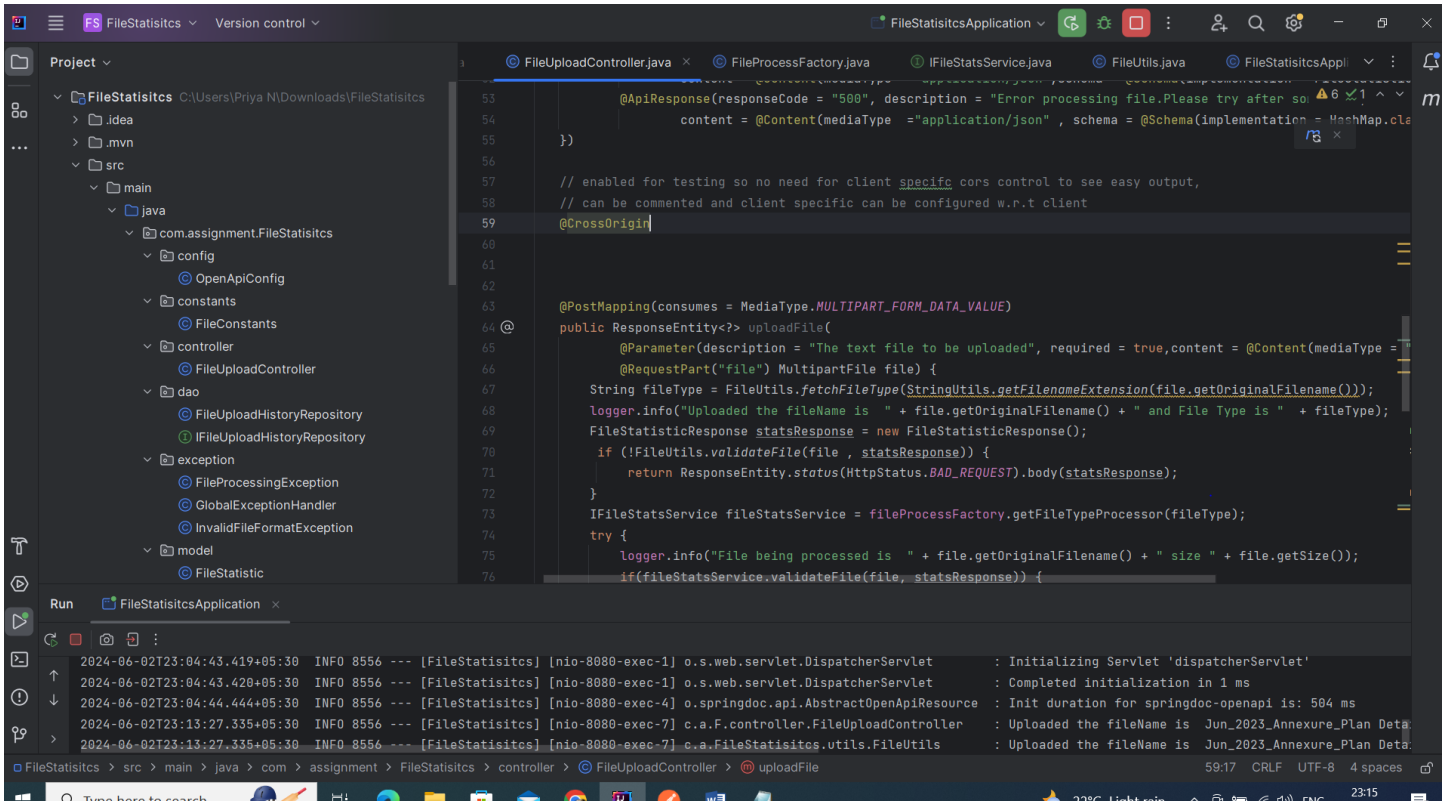
curl --location 'http://localhost:8080/api/v1/file/upload/history'

Provides History of files uploaded:

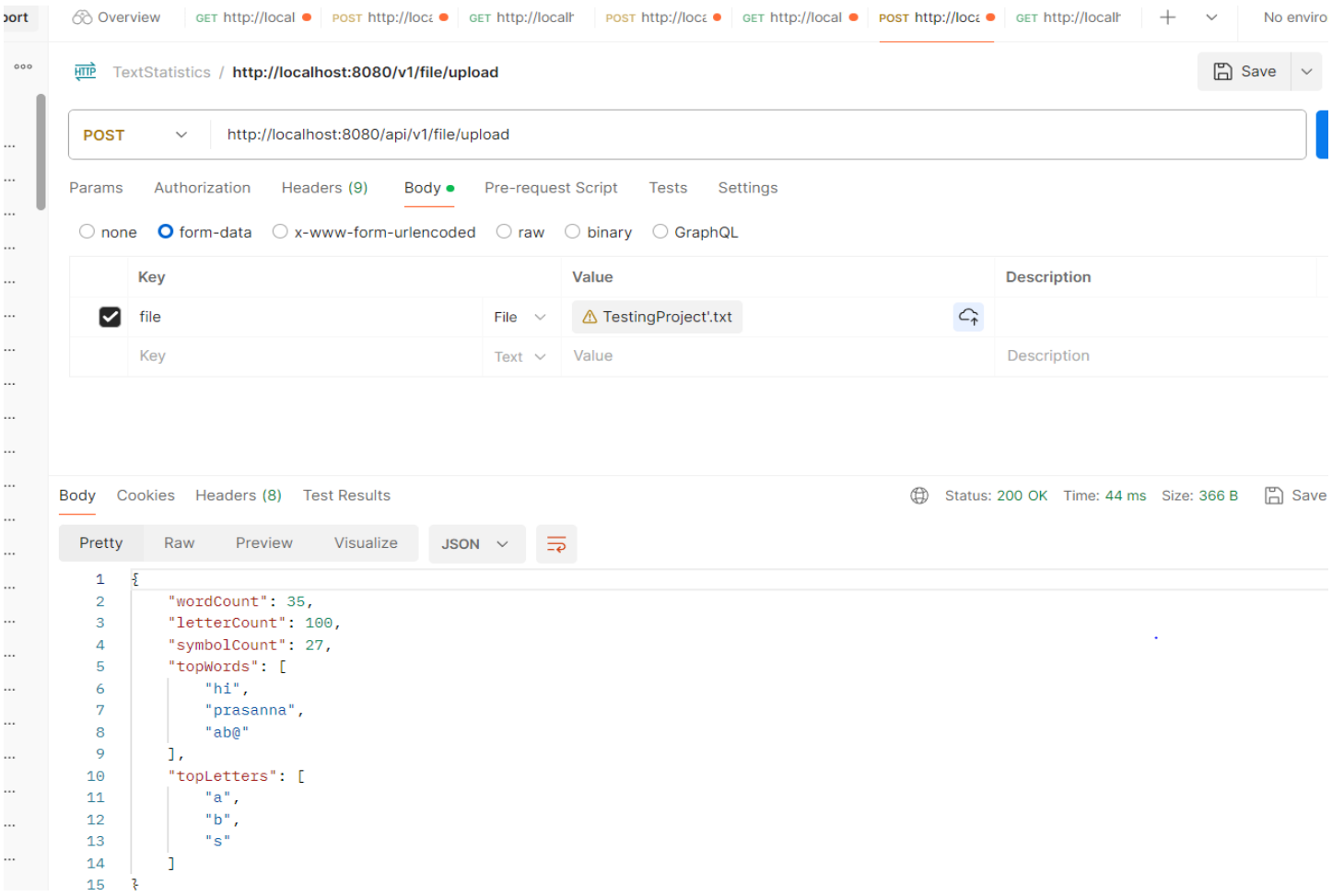

```
[
    {
        "fileName": "TestingProject'.txt",
        "uploadTimestamp": "Jun 02,2024 22:47:58"
    },
    {
        "fileName": "TestingProject'.txt",
        "uploadTimestamp": "Jun 02,2024 22:48:09"
    }
]
```


**Attaching Screenshots of Project Samples:**

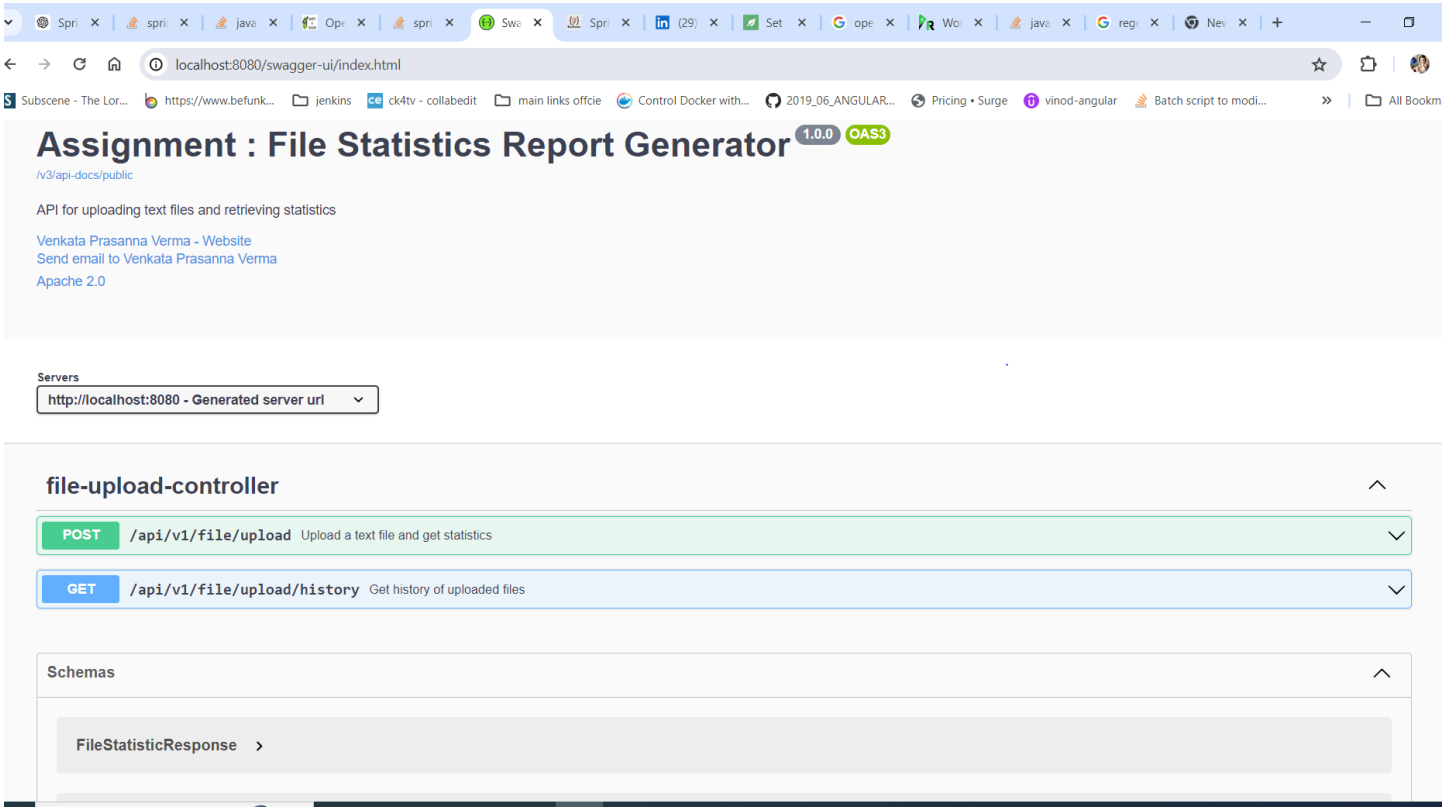
1. **Intellij**

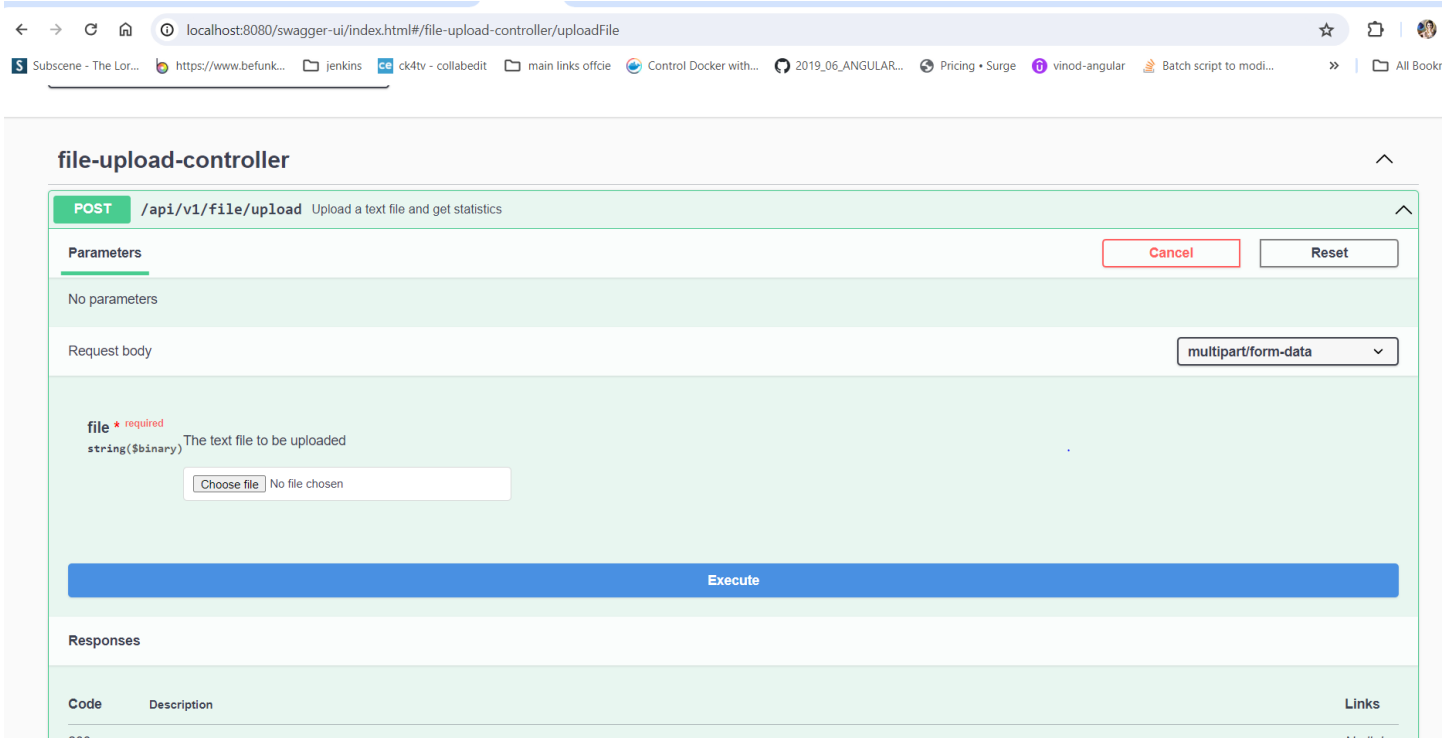

FileStatisitcs
Version control
FileStatisitcsApplication
Project
FileStatisitcs C:\Users\Priya N\Downloads\FileStatisitcs
.idea
.mvn
src
main
java
com.assignment.FileStatisitcs
config
OpenApiConfig
constants
FileConstants
controller
FileUploadController
dao
FileUploadHistoryRepository
IFileUploadHistoryRepository
exception
FileProcessingException
GlobalExceptionHandler
InvalidFileFormatException
model
FileStatistic
FileUploadController.java
FileProcessFactory.java
IFileStatsService.java
FileUtils.java
@ApiResponse(responseCode = "500", description = "Error processing file.Please try after so
content = @Content(mediaType ="application/json" , schema = @Schema(implementation
})
// enabled for testing so no need for client specifc cors control to see easy output,
// can be commented and client specific can be configured w.r.t client
@CrossOrigin
@PostMapping(consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
public ResponseEntity<?> uploadFile(
@Parameter(description = "The text file to be uploaded", required = true,content = @Content(mediaType =
@RequestPart("file") MultipartFile file) {
String fileType = FileUtils.fetchFileType(StringUtils.getFilenameExtension(file.getOriginalFilename()));
logger.info("Uploaded the fileName is " + file.getOriginalFilename() + " and File Type is " + fileType);
FileStatisticResponse statsResponse = new FileStatisticResponse();
if (!FileUtils.validateFile(file , statsResponse)) {
return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(statsResponse);
}
IFileStatsService fileStatsService = fileProcessFactory.getFileTypeProcessor(fileType);
try {
logger.info("File being processed is " + file.getOriginalFilename() + " size " + file.getSize());
if(fileStatsService.validateFile(file, statsResponse)) {
Run
FileStatisitcsApplication
2024-06-02T23:04:43.419+05:30 INFO 8556 --- [FileStatisitcs] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-06-02T23:04:43.420+05:30 INFO 8556 --- [FileStatisitcs] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2024-06-02T23:04:44.444+05:30 INFO 8556 --- [FileStatisitcs] [nio-8080-exec-4] o.springdoc.api.AbstractOpenApiResource : Init duration for springdoc-openapi is: 504 ms
2024-06-02T23:13:27.335+05:30 INFO 8556 --- [FileStatisitcs] [nio-8080-exec-7] c.a.F.controller.FileUploadController : Uploaded the fileName is Jun_2023_Annexure_Plan Deta
2024-06-02T23:13:27.335+05:30 INFO 8556 --- [FileStatisitcs] [nio-8080-exec-7] c.a.FileStatisitcs.utils.FileUtils : Uploaded the fileName is Jun_2023_Annexure_Plan Deta
FileStatisitcs > src > main > java > com > assignment > FileStatisitcs > controller > FileUploadController > uploadFile
59:17 CRLF UTF-8 4 spaces
23:15

## PostMan

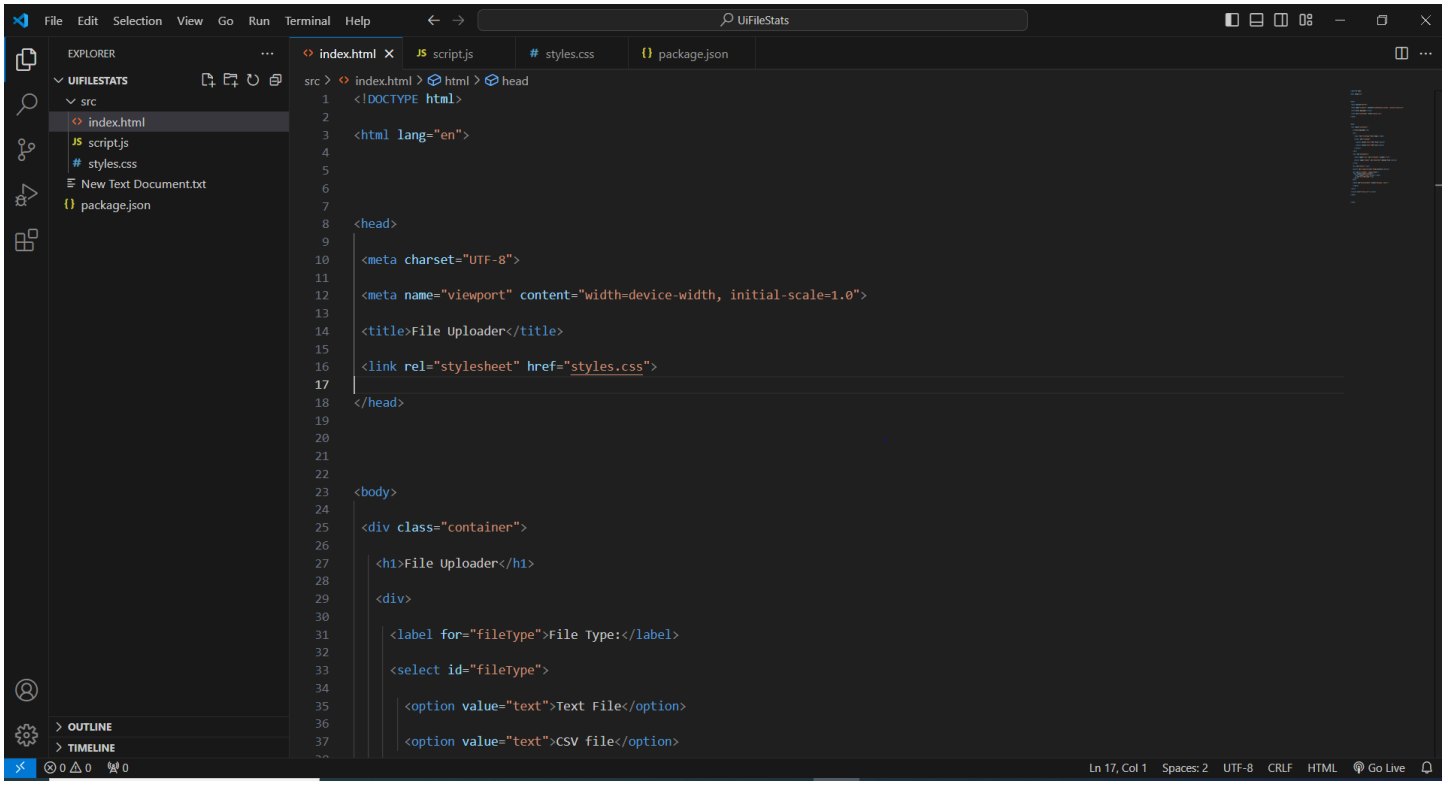

Overview
GET http://local
POST http://loca
GET http://localh
POST http://loca
GET http://local
POST http://loca
GET http://localh
No enviro
TextStatistics / http://localhost:8080/v1/file/upload
Save
POST
http://localhost:8080/api/v1/file/upload
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
Key
Value
Description
file
File
TestingProject'.txt
Key
Text
Value
Description
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 44 ms
Size: 366 B
Save
Pretty
Raw
Preview
Visualize
JSON
{
"wordCount": 35,
"letterCount": 100,
"symbolCount": 27,
"topWords": [
"hi",
"prasanna",
"ab@"
],
"topLetters": [
"a",
"b",
"s"
]
}

# Swagger Tool (Documentation and execution)

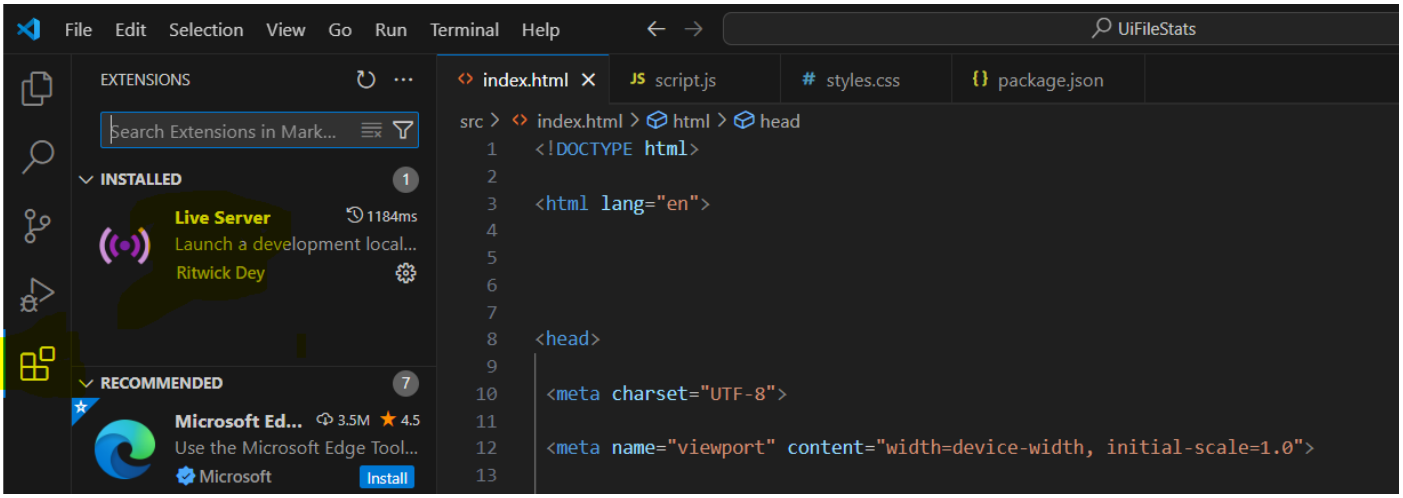


# Visual Studio :

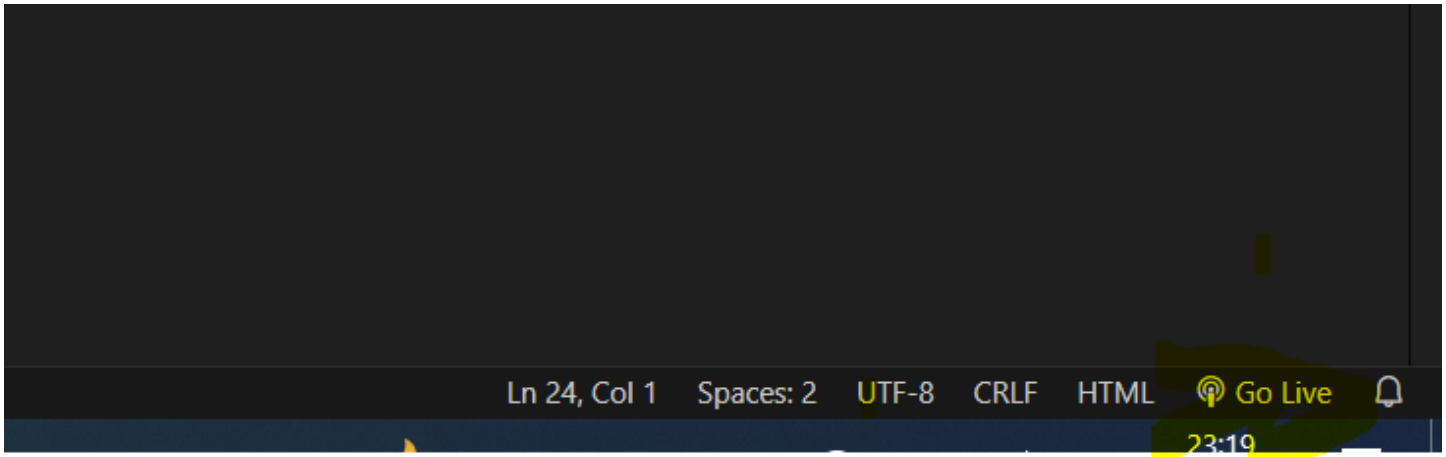


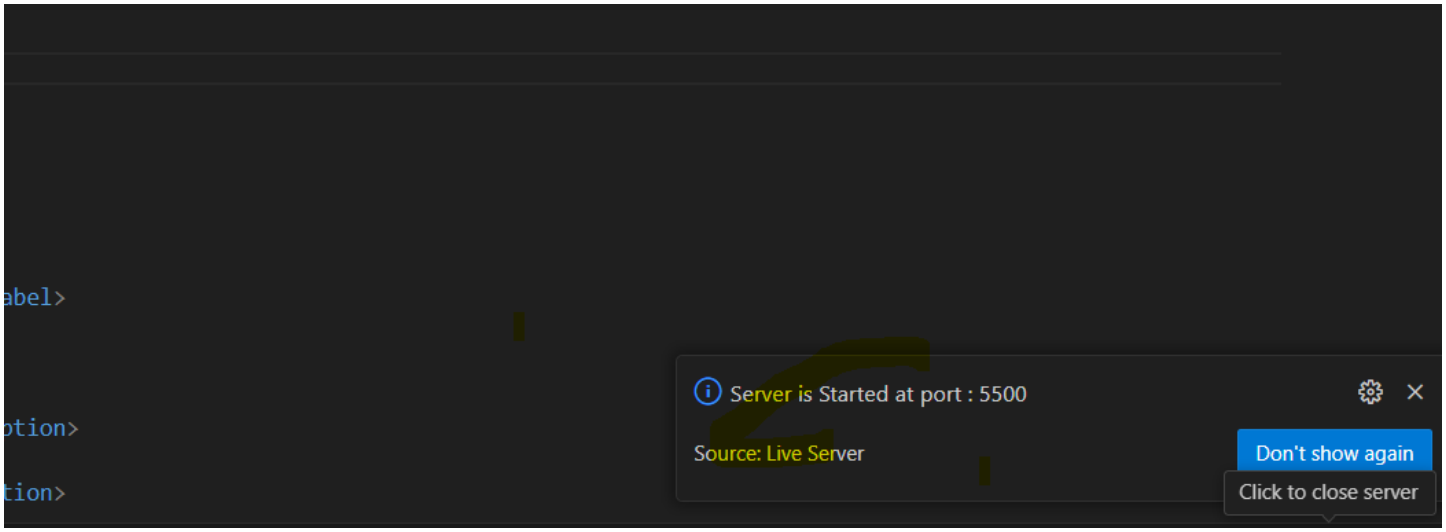

**Live Server Extension (No need to have any nodejs installed simply import 4 ui based files in visual studio and can run)**



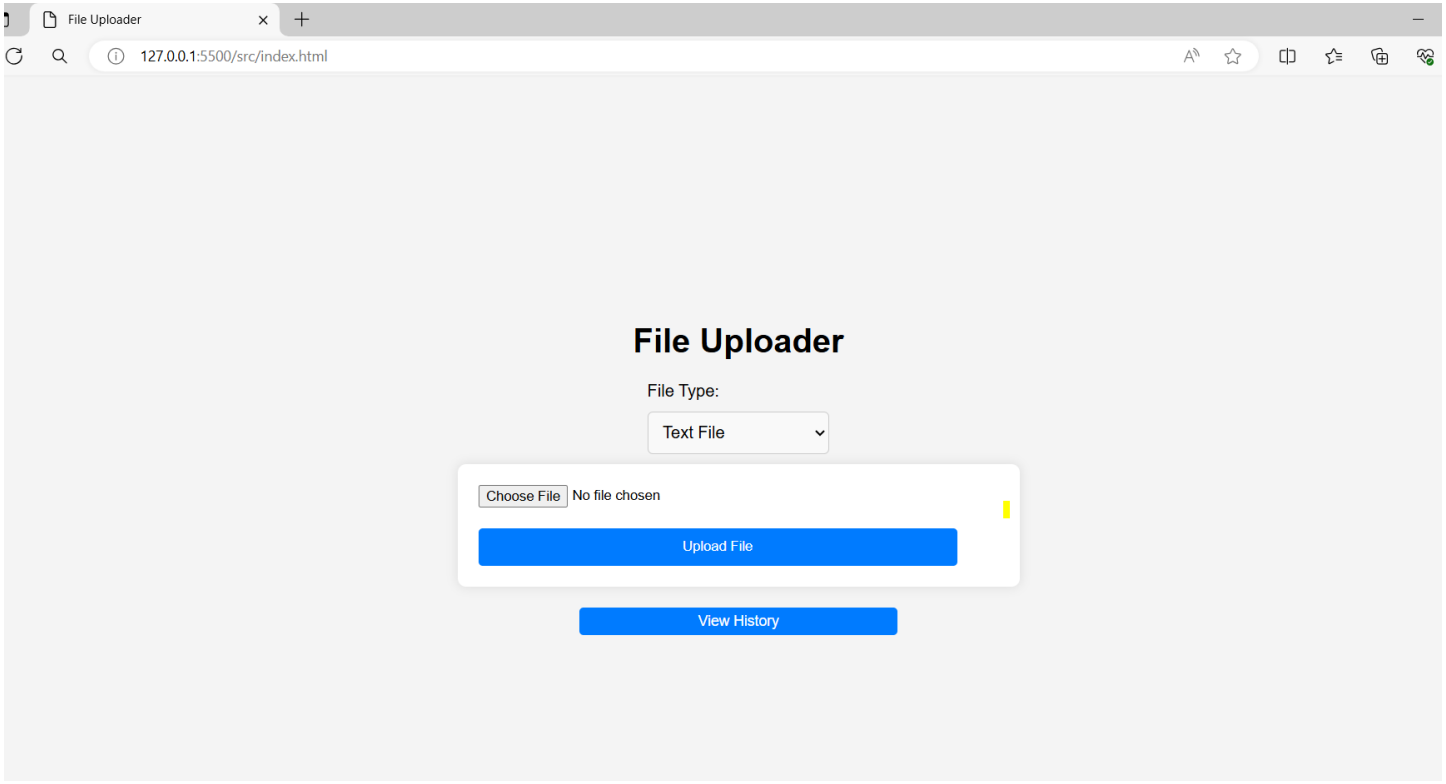
**Once Installed can click on Go Live it automatically opens browser with URL:**

**http://127.0.0.1:5500/src/index.html**

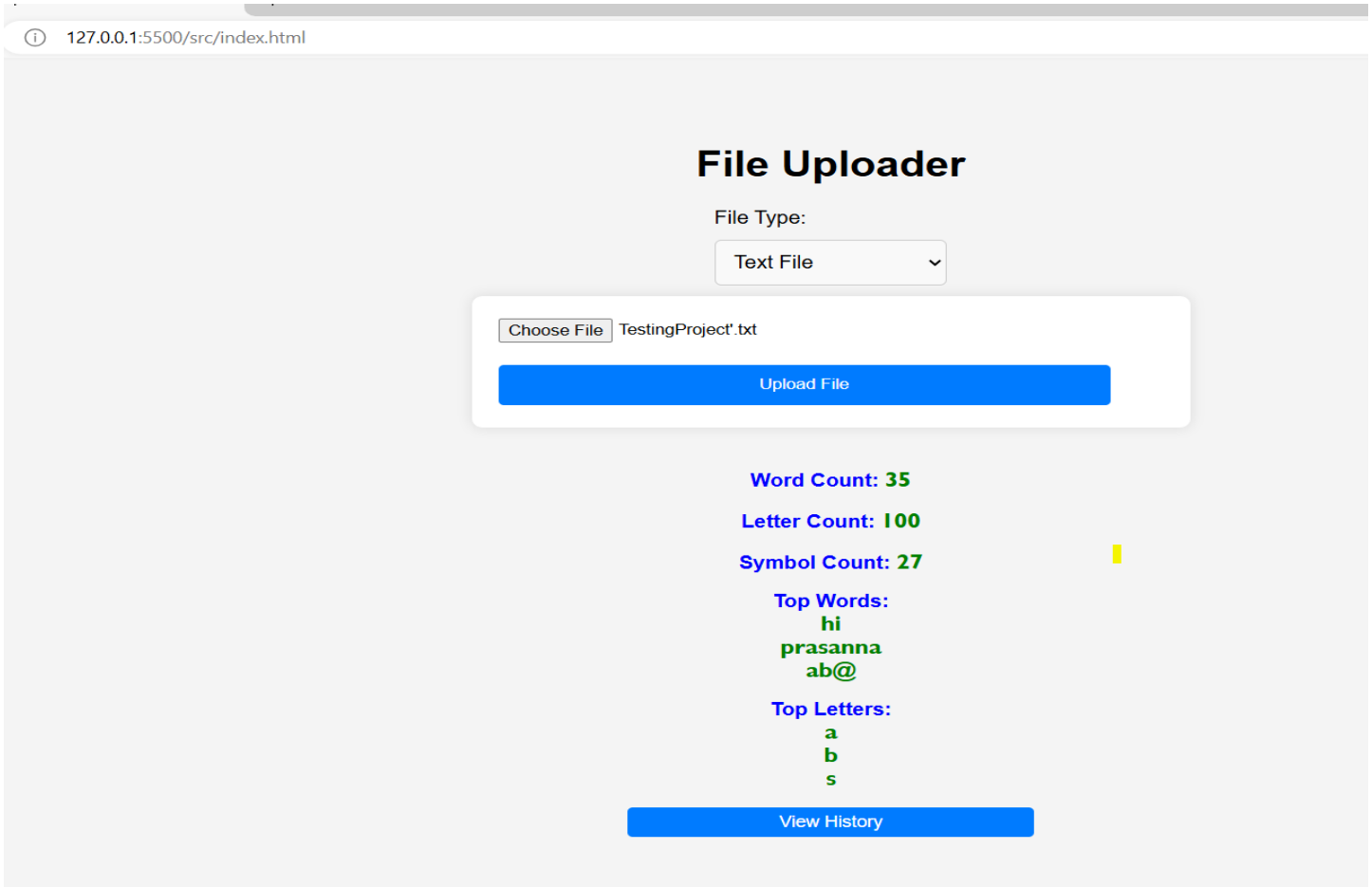


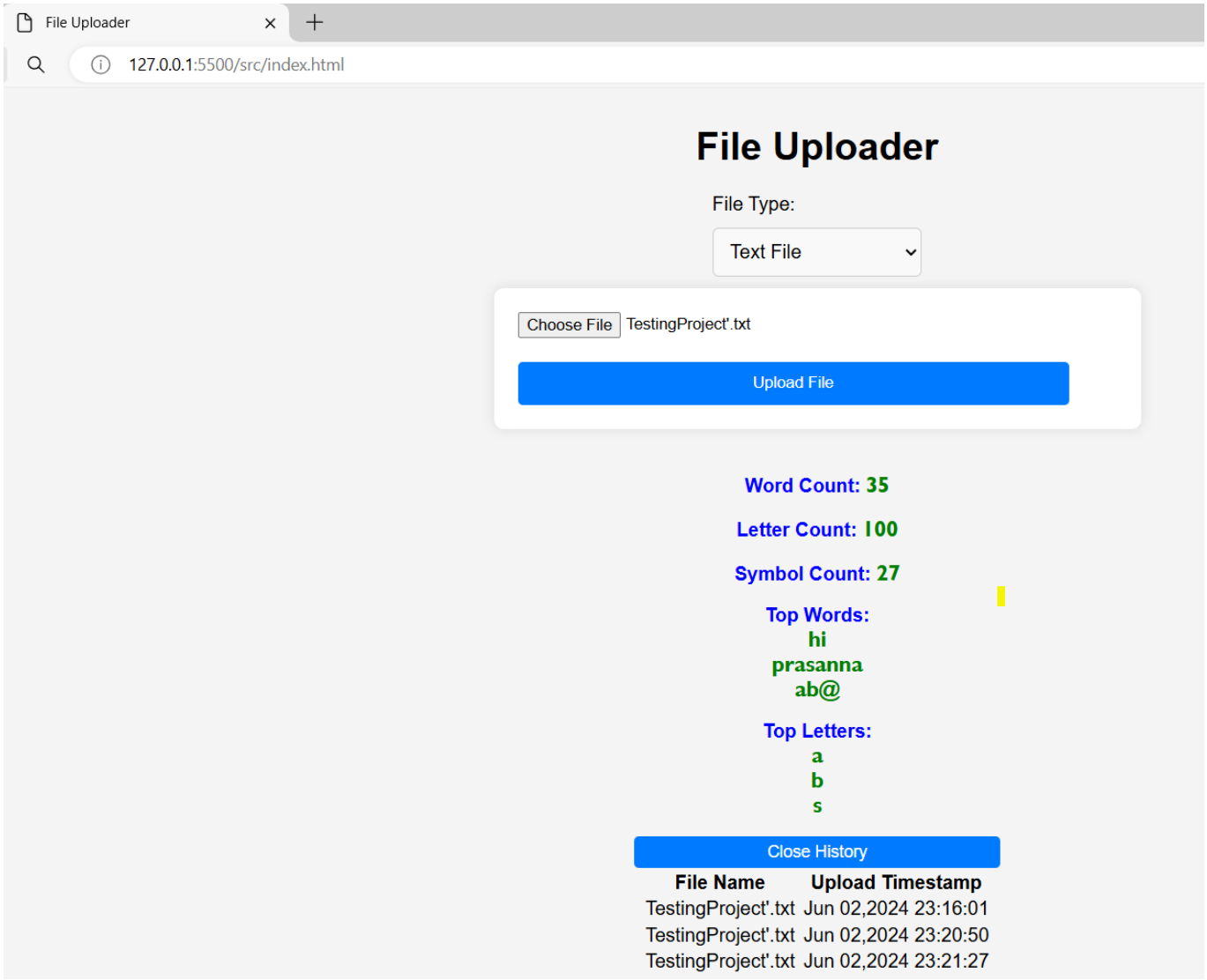



# Sample Ui Screenshot :




**After File Upload on Success , Error cases we get Popups.**






Thanks,
Prasanna Verma