System Architecture Specification

(Architekturspezifikation)

(TINF19C, SWE I Praxisprojekt 2020/2021)

Project: *AML NoSQL Database Management*

Customer: *Rentschler & Holder*
Rotebühlplatz 41
70178 Stuttgart

Supplier: by Nils-Christopher Wiesenauer - Team 5
(Nils-Christopher Wiesenauer, Namid Marxen, Johannes Timter, Jonas Bihr, ~~Max Scheub~~)
Rotebühlplatz 41
70178 Stuttgart

Version	Date	Author	Comment
0.1	24.10.2020	Nils-Christopher Wiesenauer	created
0.2	26.10.2020	Nils-Christopher Wiesenauer	added headings and table of contents
0.3	30.10.2020	Nils-Christopher Wiesenauer	finished system overview, architectural concept and systemdesign
0.4	31.10.2020	Nils-Christopher Wiesenauer	added subsystemspecification
0.5	01.11.2020	Nils-Christopher Wiesenauer	finished technical concepts
0.6	08.11.2020	Nils-Christopher Wiesenauer	added figures
0.7	12.11.2020	Nils-Christopher Wiesenauer	updated figures

Contents

1. Introduction

The goal of this project is to develop a software that supports the conversion from an AML file to JSON stored in a database. The main part of this software should be the handle of such a storage in a database.
There also should be a tool with a graphical user interface, which uses a REST API to upload AML files and to show, edit, delete the converted JSON files.

1.1. Glossar

API	Application Programming Interface
AML	Automation Markup Language is an open standard data format for storing and exchanging plant planning data.
Angular	Angular is a TypeScript based front-end framework which is published as open source software.
ExpressJS	ExpressJS is the most popular Node web framework and is the underlying library for several other popular Node web frameworks. It provides many mechanisms.
GUI	Graphical User Interface
JSend	JSend is a specification for a simple, no-frills, JSON based format for application-level communication.
MongoDB	MongoDB is a document-oriented NoSQL database used for high volume data storage.
ngx-translate	Internationalization (i18n) library for Angular
NodeJS	NodeJS is a JavaScript free and open source cross-platform for server-side programming that allows users to build network applications quickly.
npm	Node Package Manager
REST	Representational State Transfer
XML	Extensible Markup Language is a markup language to save data in an organized way, to make it human- and machine-readable.

2. System Overview

The system will work as follows: The user specifies an AML file on the web-app and uploads it via a HTTP POST request. The frontend checks the file type via form. The REST API checks the size of that file. If the type and size are valid, the system performs the conversion to the JSON format after uploading it. The result will be saved in the MongoDB database as a document. This saved document will be returned to the frontend as a JSend response. In the Angular GUI it will be listed for the user to perform actions like edit, download, and delete.

2.1. System Environment

There will be a way to access the API via web browser and HTTP requests.
Firstly, the REST API can be implemented into other projects. Other developers can use the backend for their own projects and define how the result should be used.
Secondly, the GUI will give the user an interface to access the database and the option to either upload an AML file or just show, edit, and delete the converted result as a JSON document.

2.2. Software Environment

The system requires a NodeJS version 12.x to download the needed npm packages and run the system. The REST API can be implemented into any kind of frontend framework that knows how to handle HTTP requests and JSend.

2.3. Quality Goals

The following quality goals listed below should be achieved by the following architecture.

2.3.1. Usability

By offering the tool for any web browser, a high degree of user-friendliness is achieved for everybody. This means, users can work with the graphical user interface on any web browser.

2.3.2. Maintainability

Dividing the project into smaller modules should help to make the software easier to analyse, maintain, update, and modify. The result of dividing the software into smaller modules is shown in chapter 3. Architectural Concept.

2.3.3. Portability

The front- and the backend will be portable. This means that the functionality to convert an AML file to JSON should be easily integrable by other software products.

3. Architectural Concept

The system will be based on the MEAN (MongoDB, Express.js, Angular, Node.js)-Stack with the CRUD (Create, Read, Update, Delete) functionality in the backend.

3.1. Architectural Model

The system can be divided into three main parts.
Firstly, the frontend GUI, where the user can upload, edit, show, and delete an AML file. Secondly, the REST API handles the conversion of the AML file to JSON, CRUD functionalities and the connection to the database. Finally, the database is used to store the files.

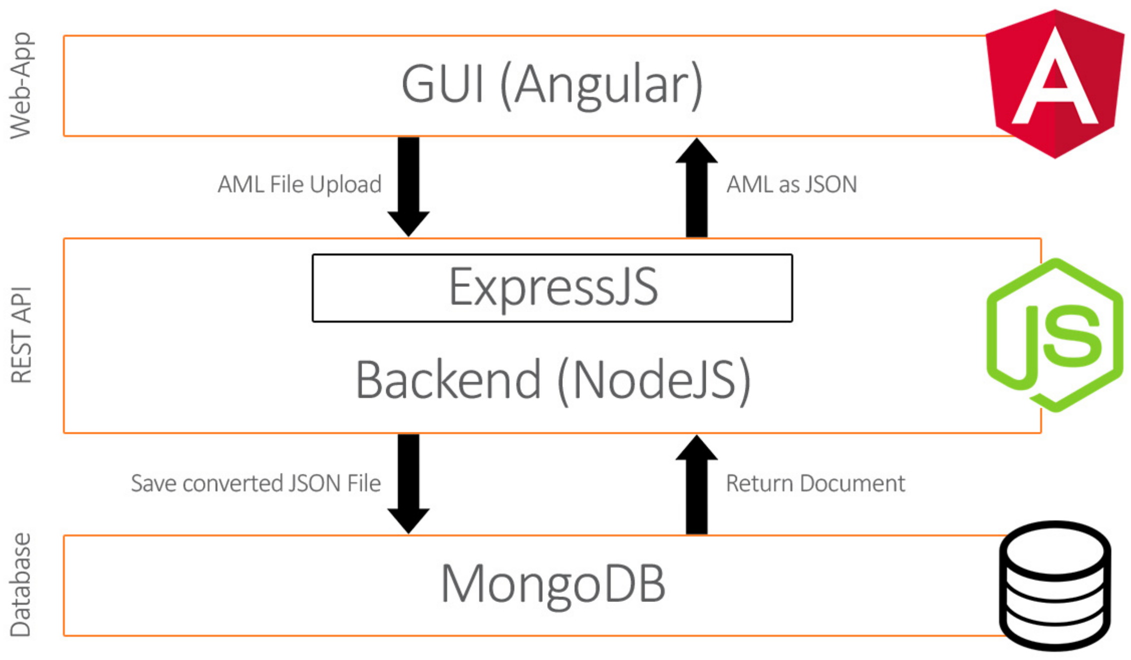
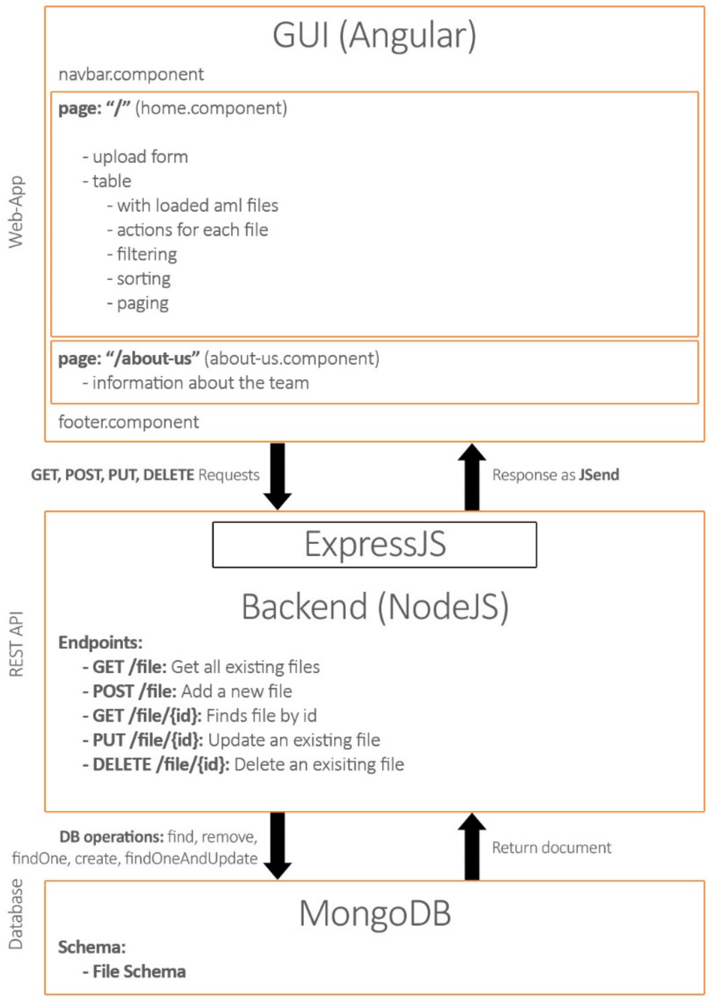


Figure 1 - Architectural Model

3.2. Component Diagram



GUI (Angular)
navbar.component
page: "/" (home.component)
- upload form
- table
- with loaded aml files
- actions for each file
- filtering
- sorting
- paging
page: "/about-us" (about-us.component)
- information about the team
footer.component
Web-App
GET, POST, PUT, DELETE Requests
Response as JSend
ExpressJS
Backend (NodeJS)
REST API
Endpoints:
- GET /file: Get all existing files
- POST /file: Add a new file
- GET /file/{id}: Finds file by id
- PUT /file/{id}: Update an existing file
- DELETE /file/{id}: Delete an exisiting file
DB operations: find, remove, findOne, create, findOneAndUpdate
Return document
Database
MongoDB
Schema:
- File Schema

Figure 2 - Component Diagram

The main part of the whole project is the connection between all the three developed services. It is divided in frontend, backend, and database. Front- and backend can work as a stand-alone project. They communicate via HTTP requests to upload, edit, and delete an AML file.

The backend contains CRUD functionalities for the stored AML files. Only the backend is authorized to store, get, and update data in the defined table in the database.

MongoDB is used as database. It is a document-oriented NoSQL database used for high volume data storage. Every uploaded AML file is stored in it with a well-defined schema.

The GUI allows the user to access the REST API with HTTP requests to interact with the database and to perform actions like show, edit and delete an AML file. Ngx-translate will be used for the internationalization here.

Additionally, the response from back- to frontend will be in the JSend format to get status, status code, description as a message and the data in a better way to handle in the frontend. With an own developed api.service.ts in the frontend, these responses will be shown in a dialog with all needed information.

4. Systemdesign

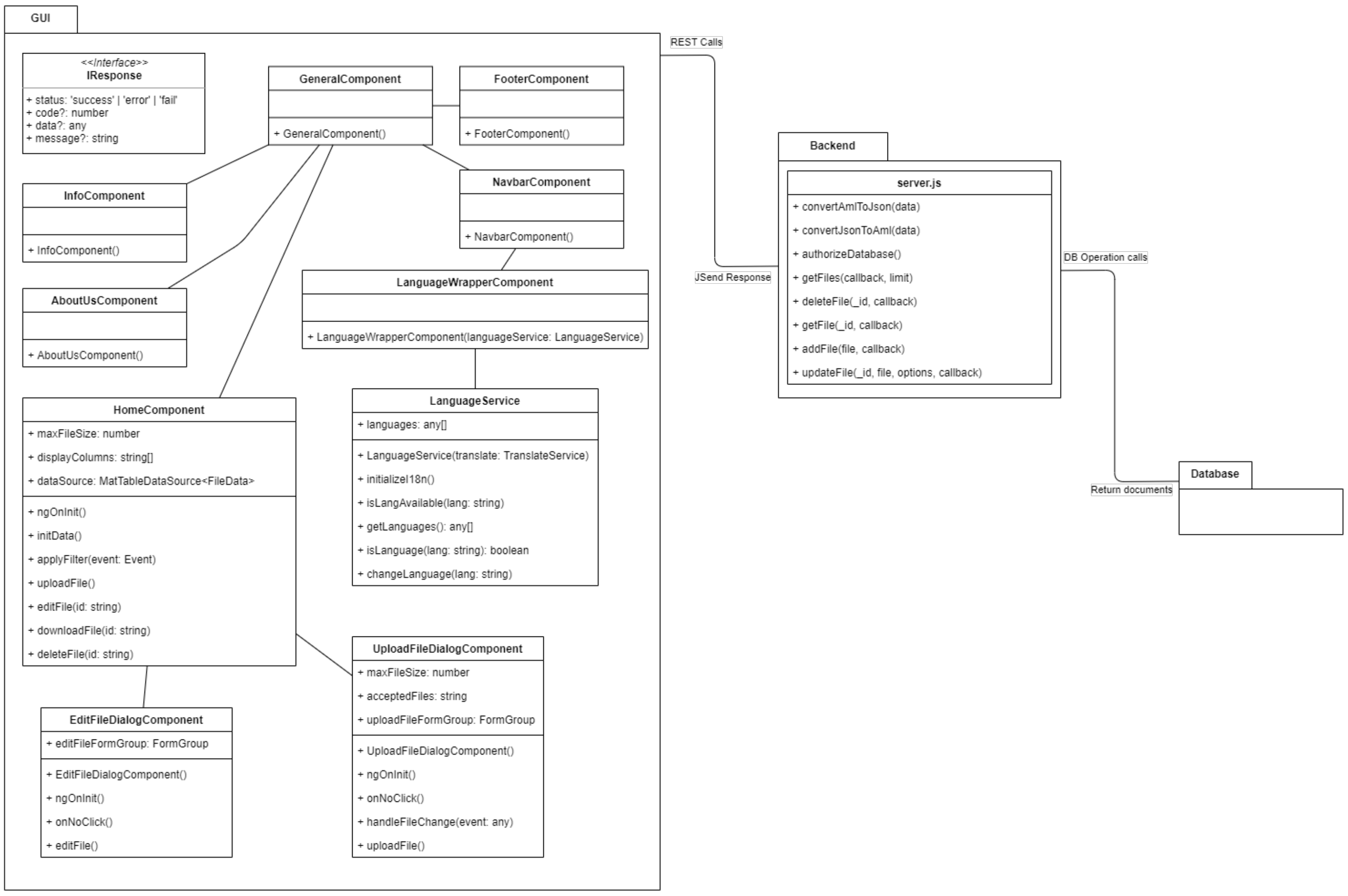


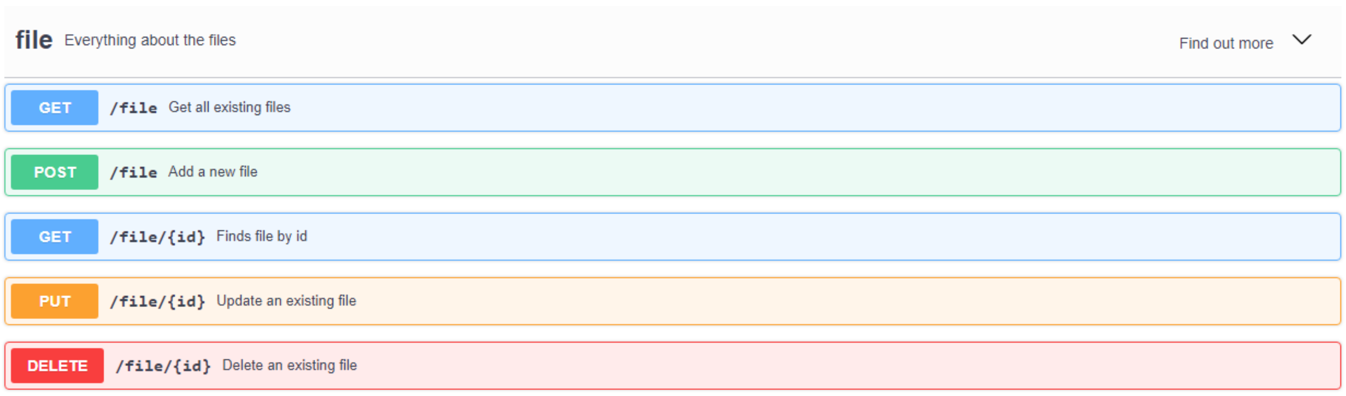
Figure 3 - Systemdesign

5. Subsystemspecification

5.1. <MOD.001>: REST API (Backend)

This REST API is the most important module because it contains the logic for converting the AML file to JSON. It also includes the authorization to the database.

<MOD.001>	REST API (Backend)
System requirements covered:	/LF110/
Service:	• Providing a REST API with CRUD • Connection with authorization to the MongoDB database
Interfaces:	• POST request to upload the AML file with the conversion to JSON and to store it in the database • GET requests to get AML files • PUT request to update/edit a stored AML file • DELETE request to delete a stored AML file
External Data:	• Input data from the GUI • Output JSend (response)
Endpoints:	• **GET /file** – Get all existing files • **POST /file** – Add a new file • **GET /file/{id}** – Finds file by id • **PUT /file/{id}** – Update an existing file • **DELETE /file/{id}** – Delete an existing file
Storage location:	https://github.com/NurNils/TINF19C_Team_5_AML_Database_Management/tree/master/SOURCE/BACKEND


file Everything about the files

Find out more

GET /file Get all existing files

POST /file Add a new file

GET /file/{id} Finds file by id

PUT /file/{id} Update an existing file

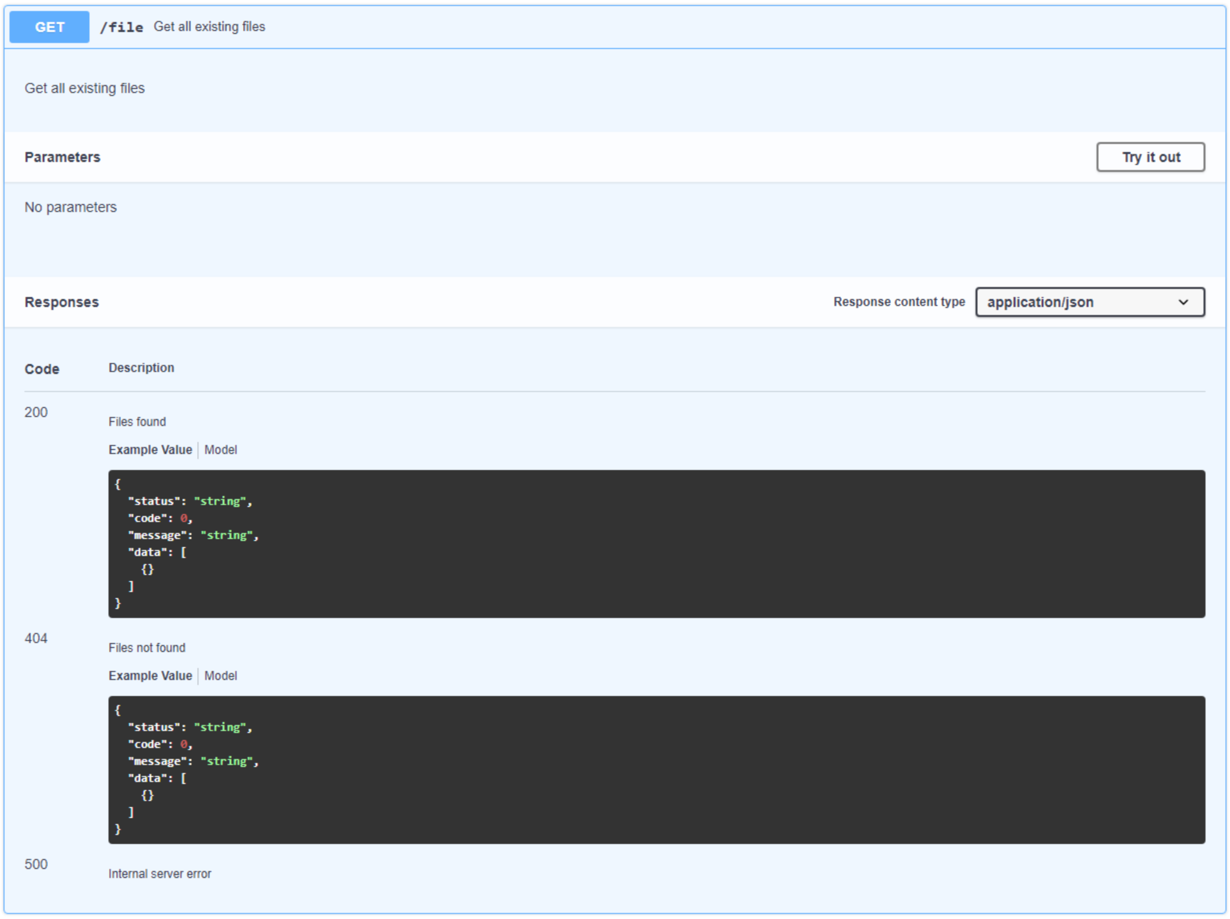
DELETE /file/{id} Delete an existing file

Figure 4 - Swagger UI: REST API Endpoints

5.2. <SUBMOD.001.001>: Get AML Files

This submodule contains the functionality to get all AML files or one AML file base on the ID from the database.

<SUBMOD.001.001>:	Get all stored AML Files
System requirements covered:	/LF80/
Service:	• GET request to get all stored AML files
Interfaces:	• Response as JSend
Endpoint:	• **GET** /file
External Data:	• Output JSend
Storage location:	https://github.com/NurNils/TINF19C_Team_5_AML_Data-base_Management/blob/master/SOURCE/BACKEND/aml-data-base-management-api/server.js


GET /file Get all existing files

Get all existing files

Parameters

Try it out

No parameters

Responses

Response content type application/json

Code Description

200

Files found

Example Value | Model

```
{
  "status": "string",
  "code": 0,
  "message": "string",
  "data": [
    {}
  ]
}
```

404

Files not found

Example Value | Model

```
{
  "status": "string",
  "code": 0,
  "message": "string",
  "data": [
    {}
  ]
}
```

500

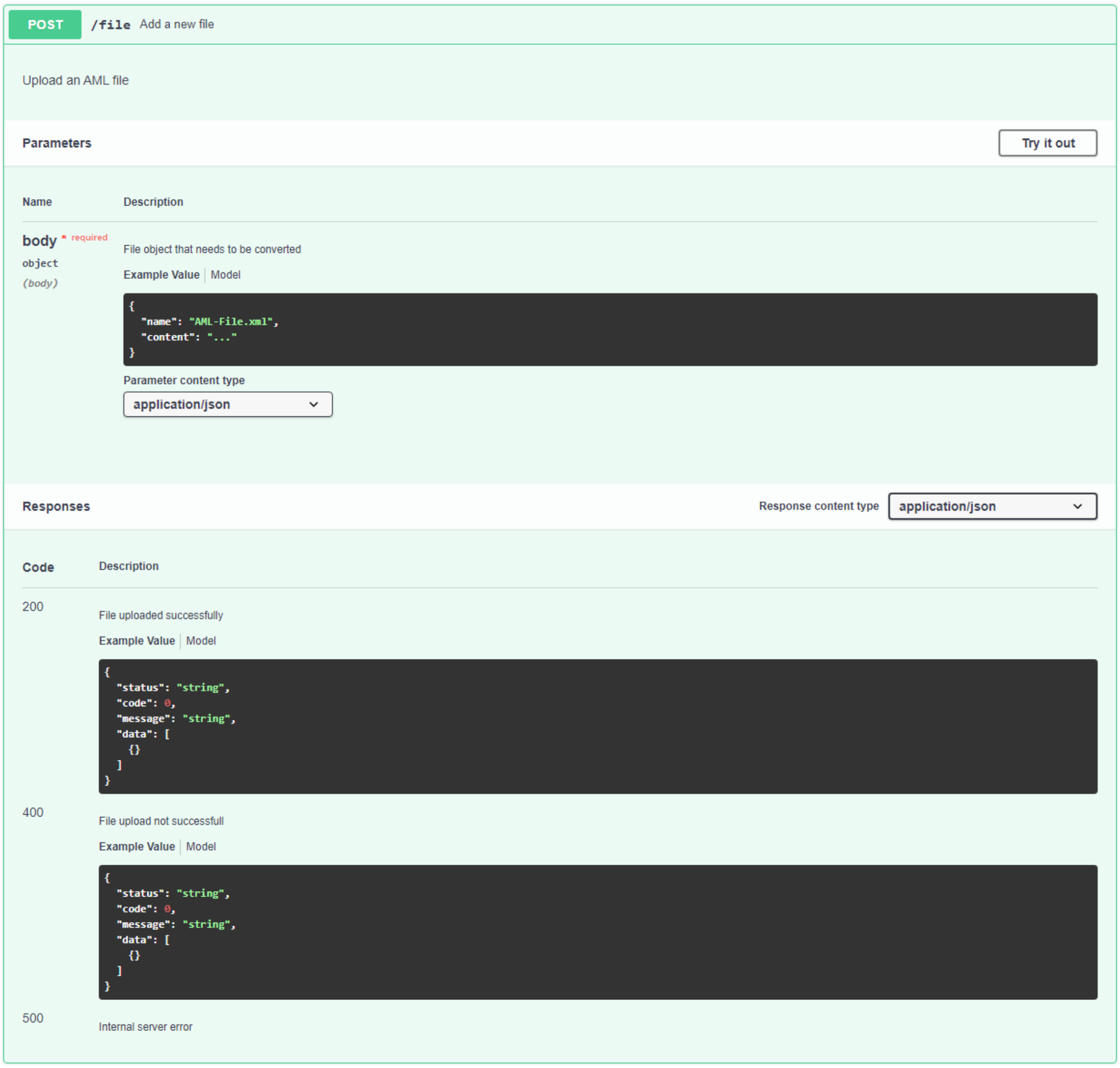
Internal server error

Figure 5 - Swagger UI: GET /file

5.3. <SUBMOD.001.002>: Upload AML File

This submodule performs the conversion and storage of an AML file. It takes the information from the frontend and stores the converted file in the MongoDB database.

<SUBMOD.001.002>	Conversion from AML to JSON logic
System requirements covered:	/LF20/, /LF30/
Service:	• POST request to upload the AML file with the conversion logic to JSON and to store it in the database
Interfaces:	• Response as JSend
Endpoint:	• **POST** /file
External Data:	• Input data from the GUI as request body • Output JSend
Storage location:	https://github.com/NurNils/TINF19C_Team_5_AML_Database_Management/blob/master/SOURCE/BACKEND/aml-database-management-api/server.js



Upload an AML file

Parameters

Name	Description
body * required object (body)	File object that needs to be converted Example Value | Model

```
{
  "name": "AML-File.xml",
  "content": "..."
}
```

Parameter content type: application/json

Responses

Response content type: application/json

Code	Description
200	File uploaded successfully Example Value | Model

```
{
  "status": "string",
  "code": 0,
  "message": "string",
  "data": [
    {}
  ]
}
```

Code	Description
400	File upload not successfull Example Value | Model

```
{
  "status": "string",
  "code": 0,
  "message": "string",
  "data": [
    {}
  ]
}
```

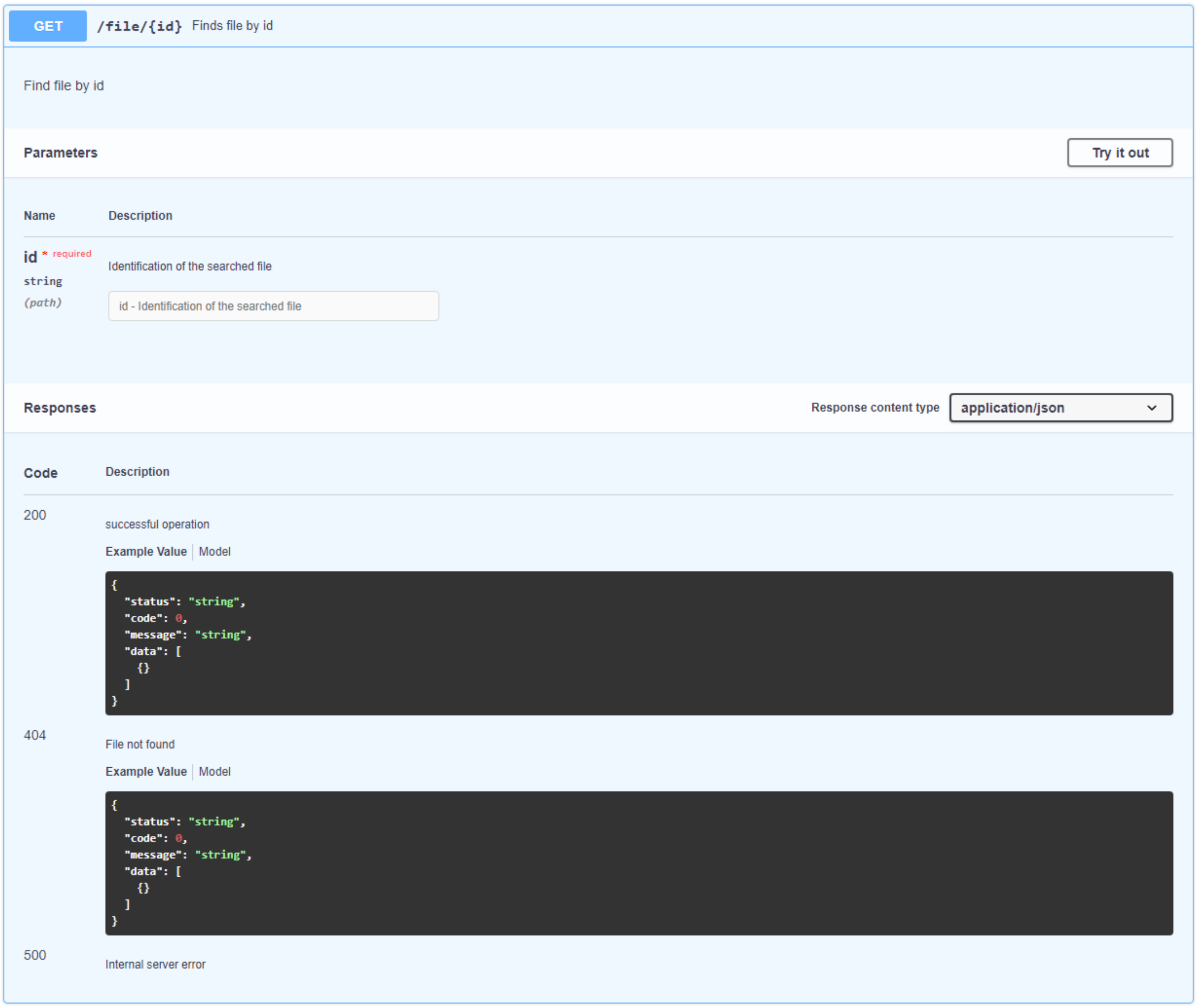
Code	Description
500	Internal server error

Figure 6 - Swagger UI: POST /file

5.4. <SUBMOD.001.003>: Get AML File by ID

This submodule contains the functionality to get all AML files or one AML file base on the ID from the database.

<SUBMOD.001.003>:	Get AML File based on ID or get all stored AML files
System requirements covered:	/LF40/, /LF50/
Service:	• GET request to get a stored AML file by ID
Interfaces:	• Response as JSend
Endpoint:	• **GET** /file/{id}
External Data:	• Input ID as parameter • Output JSend
Storage location:	https://github.com/NurNils/TINF19C_Team_5_AML_Database_Management/blob/master/SOURCE/BACKEND/aml-database-management-api/server.js



```
{
  "status": "string",
  "code": 0,
  "message": "string",
  "data": [
    {}
  ]
}
```

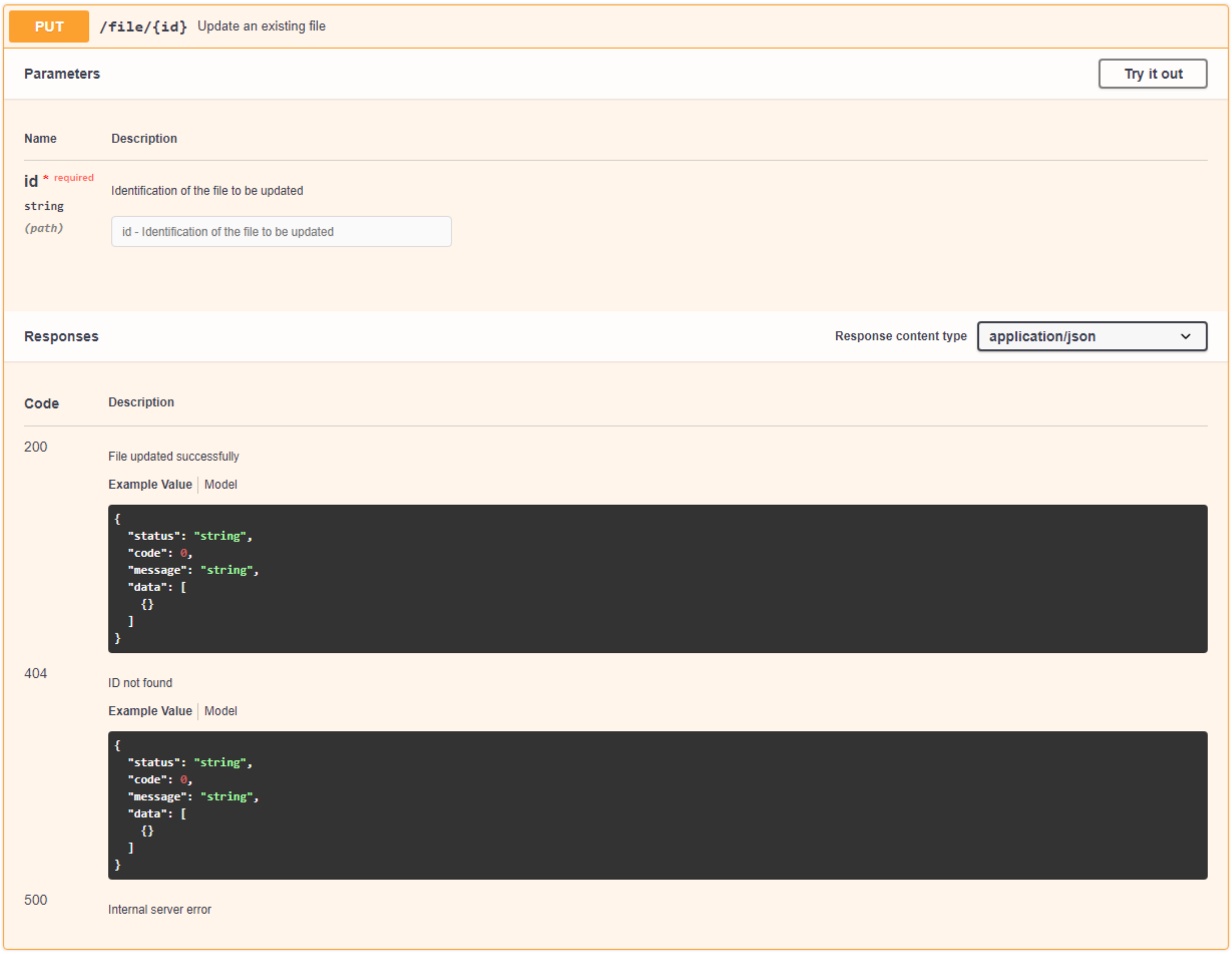
```
{
  "status": "string",
  "code": 0,
  "message": "string",
  "data": [
    {}
  ]
}
```

Figure 7 - Swagger UI: GET /file/{id}

5.5. <SUBMOD.001.004>: Edit AML File

This submodule allows to edit an existing AML file.

<SUBMOD.001.004>:	Edit AML File
System requirements covered:	/LF60/
Service:	• PUT request to update/edit a stored AML file
Interfaces:	• Response as JSend
Endpoint:	• **PUT** /file/{id}
External Data:	• Input request body with updated data and ID as parameter • Output JSend
Storage location:	https://github.com/NurNils/TINF19C_Team_5_AML_Data-base_Management/blob/master/SOURCE/BACKEND/aml-data-base-management-api/server.js


PUT /file/{id} Update an existing file

Parameters

Name	Description
id * required string (path)	Identification of the file to be updated id - Identification of the file to be updated

Responses

Response content type: application/json

Code	Description
200	File updated successfully
404	ID not found
500	Internal server error

200 — Example Value | Model

```
{
  "status": "string",
  "code": 0,
  "message": "string",
  "data": [
    {}
  ]
}
```

404 — Example Value | Model

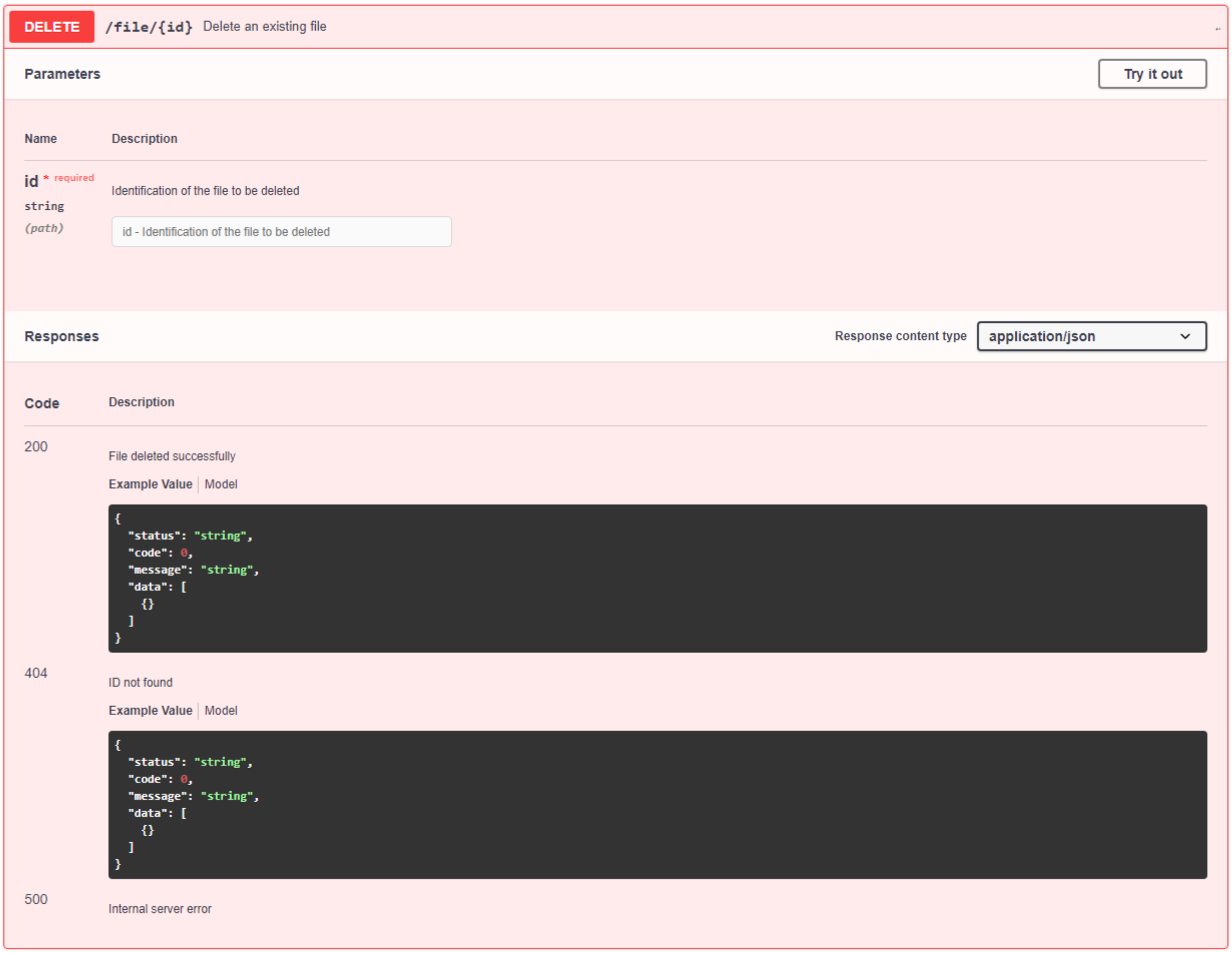
```
{
  "status": "string",
  "code": 0,
  "message": "string",
  "data": [
    {}
  ]
}
```

Figure 8 - Swagger UI: PUT /file/{id}

5.6. <SUBMOD.001.005>: Delete AML File

The functionality to delete an existing AML file in the database is contained in this submodule.

<SUBMOD.001.005>:	Delete AML File
System requirements covered:	/LF70/
Service:	• DELETE request to delete a stored AML file
Interfaces:	• Response as JSend
Endpoint:	• **DELETE** /file/{id}
External Data:	• Input ID as parameter • Output JSend
Storage location:	https://github.com/NurNils/TINF19C_Team_5_AML_Data-base_Management/blob/master/SOURCE/BACKEND/aml-data-base-management-api/server.js



```
{
  "status": "string",
  "code": 0,
  "message": "string",
  "data": [
    {}
  ]
}
```

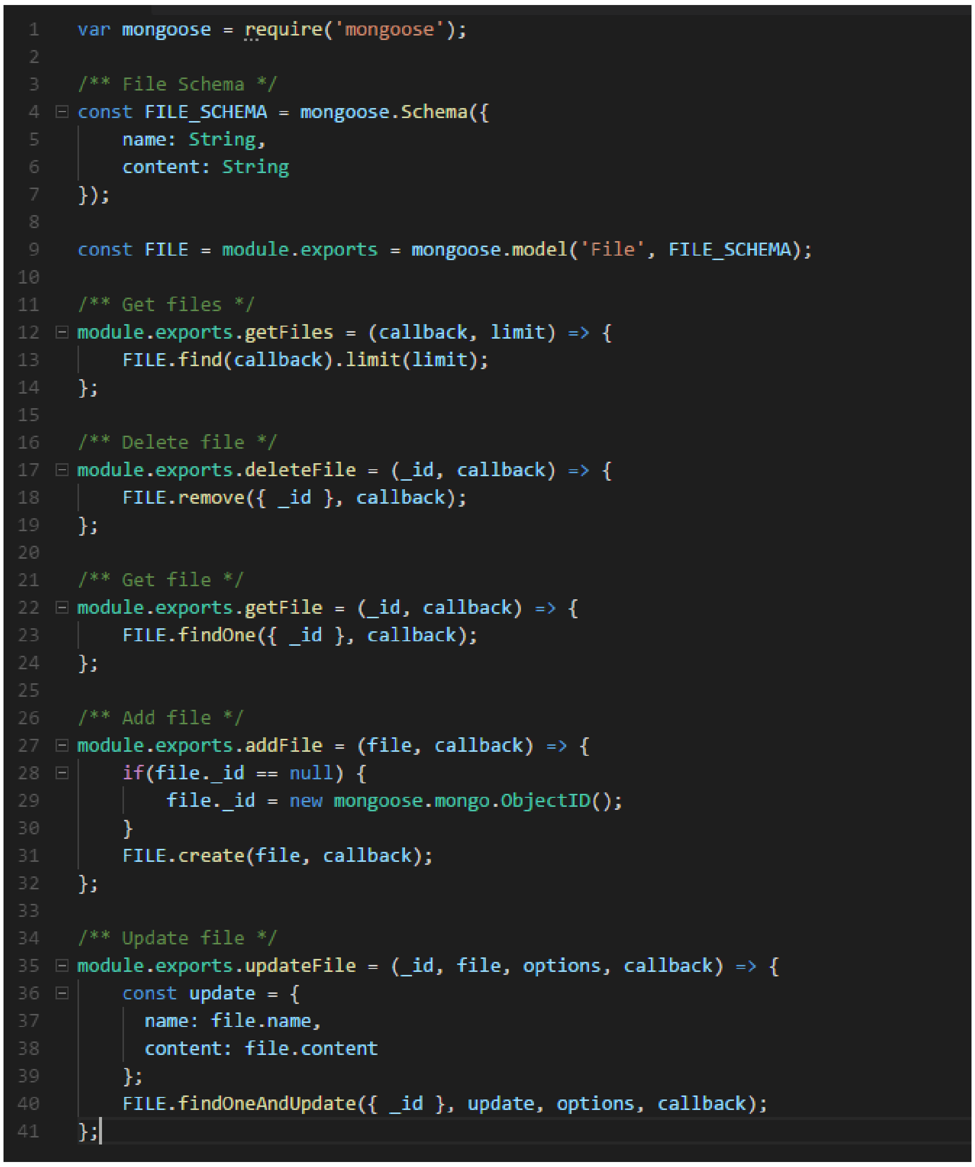
```
{
  "status": "string",
  "code": 0,
  "message": "string",
  "data": [
    {}
  ]
}
```

Figure 9 - Swagger UI: DELETE /file/{id}

5.7. <MOD.002>: Database

This module is about the storage. The uploaded AML file needs to be saved in a MongoDB database.

<MOD.002>	Database
System requirements covered:	/LF100/
Service:	• Handle storage • Sort and filter documents • Save files in a specified schema
Interfaces:	-
External Data:	• Output document from MongoDB which needs to be handled in the backend
Storage location:	https://github.com/NurNils/TINF19C_Team_5_AML_Database_Management/tree/master/SOURCE/BACKEND



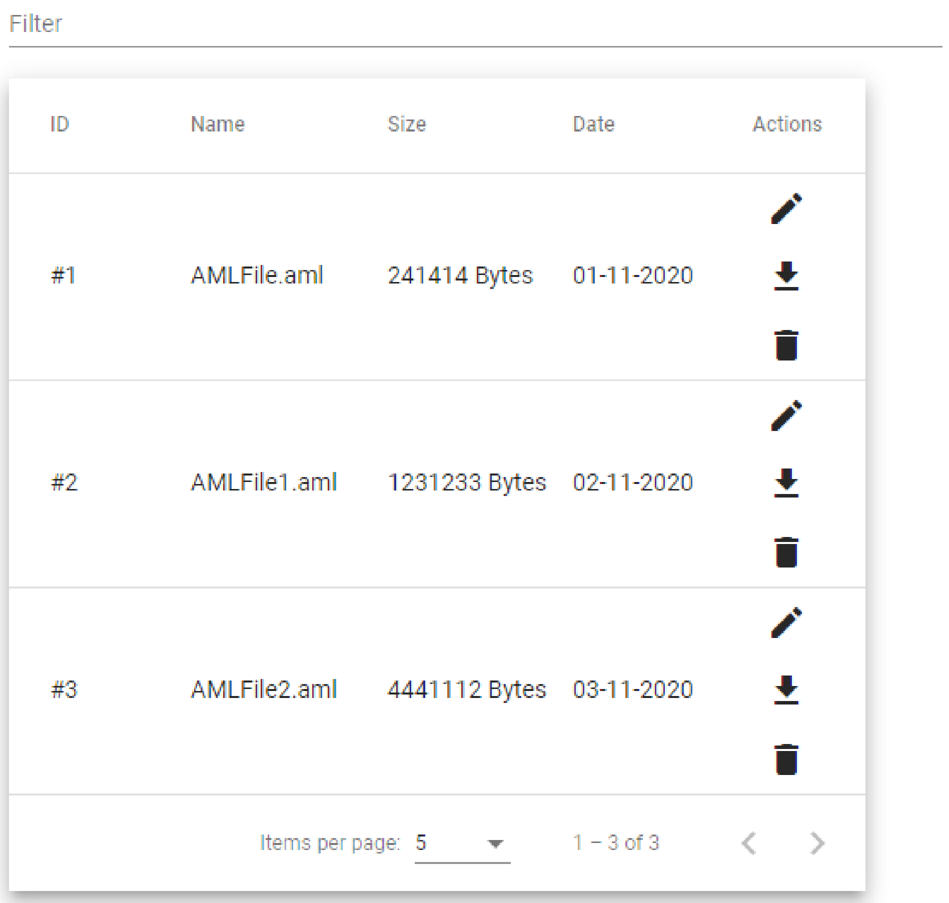
```
var mongoose = require('mongoose');

/** File Schema */
const FILE_SCHEMA = mongoose.Schema({
    name: String,
    content: String
});

const FILE = module.exports = mongoose.model('File', FILE_SCHEMA);

/** Get files */
module.exports.getFiles = (callback, limit) => {
    FILE.find(callback).limit(limit);
};

/** Delete file */
module.exports.deleteFile = (_id, callback) => {
    FILE.remove({ _id }, callback);
};

/** Get file */
module.exports.getFile = (_id, callback) => {
    FILE.findOne({ _id }, callback);
};

/** Add file */
module.exports.addFile = (file, callback) => {
    if(file._id == null) {
        file._id = new mongoose.mongo.ObjectID();
    }
    FILE.create(file, callback);
};

/** Update file */
module.exports.updateFile = (_id, file, options, callback) => {
    const update = {
      name: file.name,
      content: file.content
    };
    FILE.findOneAndUpdate({ _id }, update, options, callback);
};
```

Figure 10 - Database File Schema

5.8. <MOD.003>: Graphical User Interface (Frontend)

This module specifies and implements the graphical user interface and manages all possible in- and outputs.

<MOD.004>	Graphical User Interface
System requirements covered:	/LF80/
Service:	• Display a graphical user interface to the user • Handle user input • Handle all possible outputs, including any kind of occurring exceptions and the handle of JSend and HTTP requests
Interfaces:	• User input • Well-defined graphical interface • Paging for stored AML files • Filtering for stored AML files
External Data:	• REST API JSend Output
Storage location:	https://github.com/NurNils/TINF19C_Team_5_AML_Data-base_Management/tree/master/SOURCE/FRONTEND


Filter

ID	Name	Size	Date	Actions
#1	AMLFile.aml	241414 Bytes	01-11-2020	
#2	AMLFile1.aml	1231233 Bytes	02-11-2020	
#3	AMLFile2.aml	4441112 Bytes	03-11-2020	

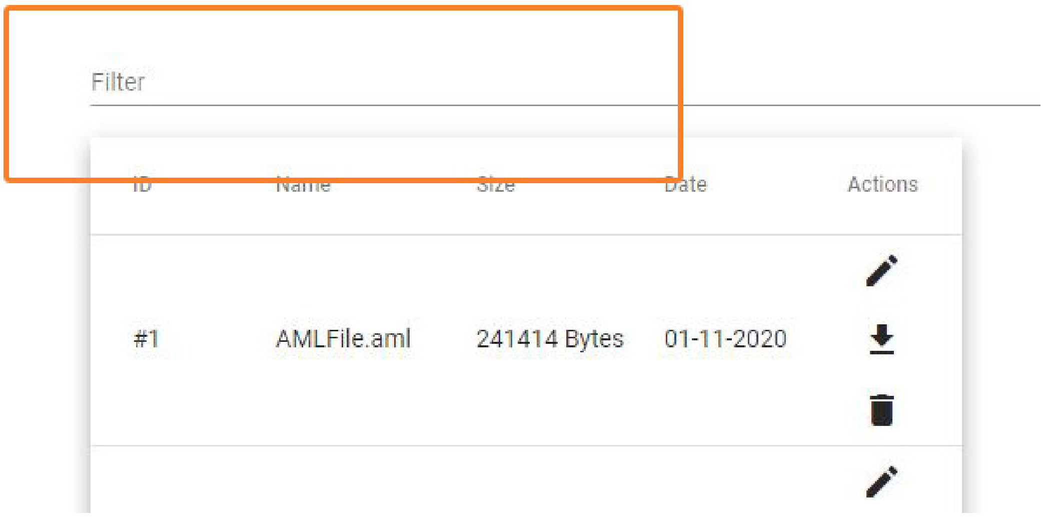
Items per page: 5 1 – 3 of 3

Figure 11 - GUI Overview

5.9. <SUBMOD.003.001>: Filtering

This submodule contains the filtering frontend-side.

<SUBMOD.001.003>:	Filtering of existing AML files
System requirements covered:	/LF60/
Service:	• Filter the list of the table
Interfaces:	• Filter input to search frontend-side for existing AML files base on ID
External Data:	• Loaded AML files
Storage location:	https://github.com/NurNils/TINF19C_Team_5_AML_Data-base_Management/tree/master/SOURCE/FRONTEND/aml-data-base-management/src/app/pages/general/home



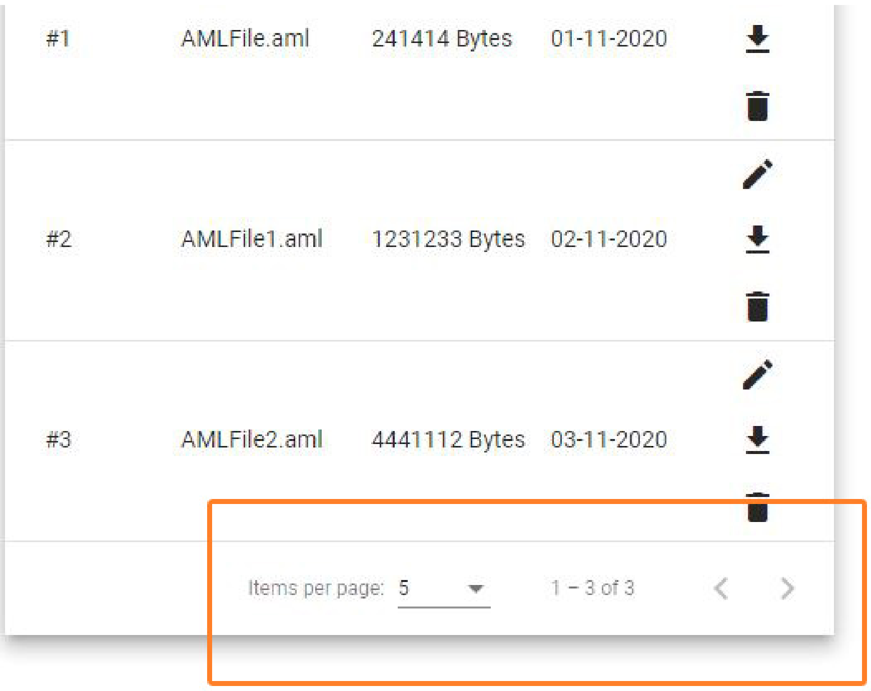
ID	Name	Size	Date	Actions
#1	AMLFile.aml	241414 Bytes	01-11-2020	

Figure 12 - GUI Filtering

5.10. <SUBMOD.003.002>: Paging

This submodule specified the paging for the existing AML files.

<SUBMOD.003.002>:	Paging
System requirements covered:	/LF70/
Service:	• Define how many items should be shown on the current page
Interfaces:	• Drop-down to select how many items should be shown per page
External Data:	• Loaded AML files
Storage location:	https://github.com/NurNils/TINF19C_Team_5_AML_Data-base_Management/tree/master/SOURCE/FRONTEND/aml-data-base-management/src/app/pages/general/home



#1	AMLFile.aml	241414 Bytes	01-11-2020
#2	AMLFile1.aml	1231233 Bytes	02-11-2020
#3	AMLFile2.aml	4441112 Bytes	03-11-2020

Items per page: 5 1 – 3 of 3

Figure 13 - GUI Paging

6. Technical Concepts

6.1. Persistence

One part of this project is to convert an AML file into JSON to be stored in a database. Data persistence is relevant for this kind of project.

6.2. User Interface

Users can access the conversion REST API via HTTP requests. This option is specified in the module specification for the Graphical User Interface (frontend).

6.3. Ergonomics

The graphical user interface will follow the standard ergonomic design patterns with the help of Angular Material.
For example, the font size should be large enough so the user experience is satisfying and the everything should be easy to use.

6.4. Communication with other IT-Systems

A developer can implement the REST API in their own software to interact with other IT-Systems. In addition to that, users can use the REST API to convert their files via GUI, so no other communication is needed.

6.5. Deployment

The REST API and GUI will be deployed on a server.

6.6. Data Validation

Before a conversion can take place, the input file needs to be correct (an .aml file) to ensure a conversion to JSON is possible. This is archived with a form.

6.7. Exception Handling

When the input file is correct the user needs to click on the button “upload”. If the REST API throws errors, everything will be shown to the user via an own implemented error handling service.

6.8. Internationalisation

The main language of the GUI is English and German. It is managed with npm package ngx-translate. If any other language is needed, it can be implemented easily with an additional .json file with translation keys in it.

6.9. Testability

The software is composed of different modules. These modules are tested separately. To receive an overview about the system tests, the system test plan provides more information, and the system test report contains all the results.

6.10. Availability

The program and code can be cloned via GitHub.

7. Figures