EMOTION CLASSIFICATION

backend application

USER MANUAL

January 2023

Table of contents

1. User manual 3
  1.1. Running the application 3
    1.1.1. External services configuration 3
    1.1.2. Local environment 7
    1.1.3. Cloud deployment 8
  1.2. Authentication and authorization 10
  1.3. Services endpoints 12
    1.3.1. Comment resource 13
    1.3.2. Assignment resource 14
    1.3.3. Emotion resource 16
  1.4. Swagger UI 17

1. User manual

This user manual is to present the Emotion Classification backend application created within the project for this engineering thesis. It is dedicated to a developer, who implements a mobile application that makes use of the REST API released by this project. Therefore, the chapter elaborates on a few points. Firstly, the process of running the application locally and deploying it in the cloud environment together with its recommendations is described. After this, the process of getting authenticated and authorized within the Emotion Classification backend is explained. The next part of this chapter focuses on documenting the endpoints of the API and their usage. Finally, the Swagger UI tool incorporated in the application is described.

1.1. Running the application

A basic requirement to run the application locally or deploy it in the cloud is to do it on a machine with a Java environment installed. The Java SE 17 version is the minimum requirement to run the project properly. Having the project's source code downloaded from the remote repository or extracted from an archive file, the next step is to configure the external services, which are crucial for the application functioning.

1.1.1. External services configuration

The external services required to be configured for the application are:

- *Firebase Authentication* – it enables creating user accounts and authorizing users so that they are able to use backend module's services,
- *YouTube Data API* – it is an API provided by Google, which makes it possible to download Internet comments from the YouTube platform,
- *MongoDB Atlas* – it is a platform that runs the MongoDB database used to store application's data.

Both for accessing Firebase and YouTube services, there is needed a Google account, which can be created for free. With an account a Firebase console may be entered with the use of the following link: *https://console.firebase.google.com/*. On the landing page, which is visible in figure 5.1, there is an 'Add project' button giving a possibility to create a Firebase project for the application. This process only asks the user to name the project. After doing that, it is ready for further configuration. Within the project, an authentication service with *Email/Password Sign-in* method should be turned on.

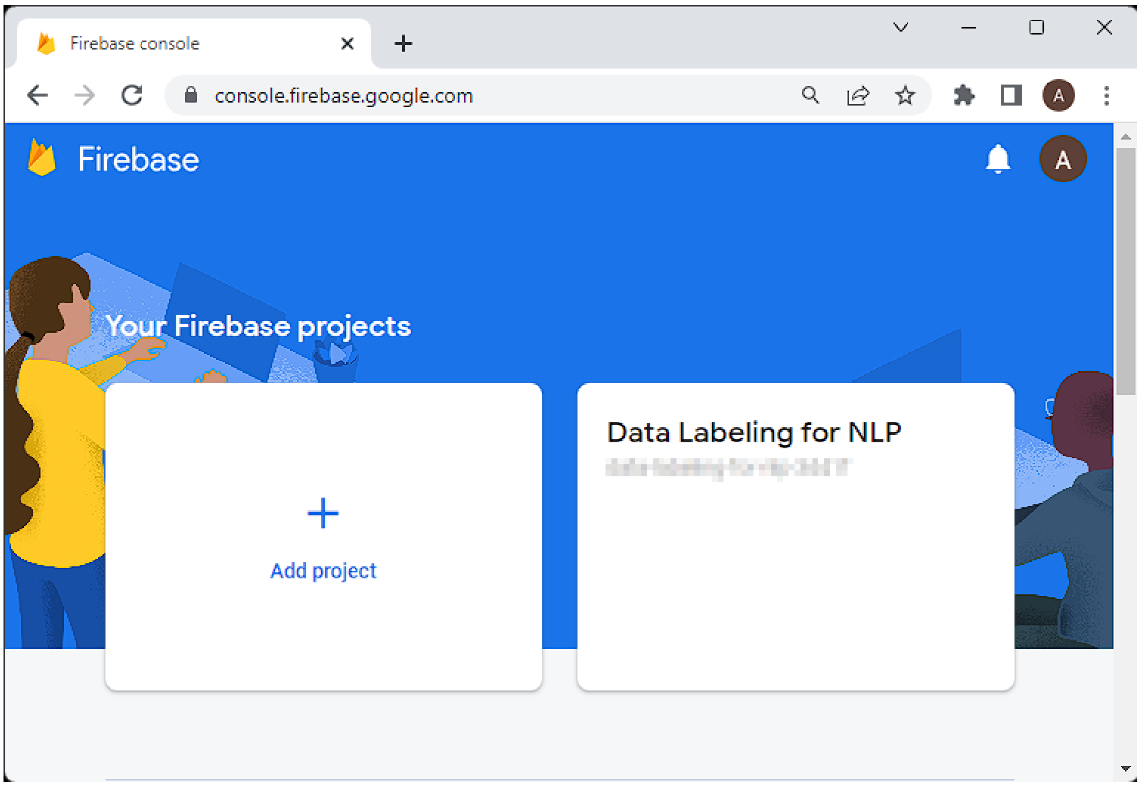
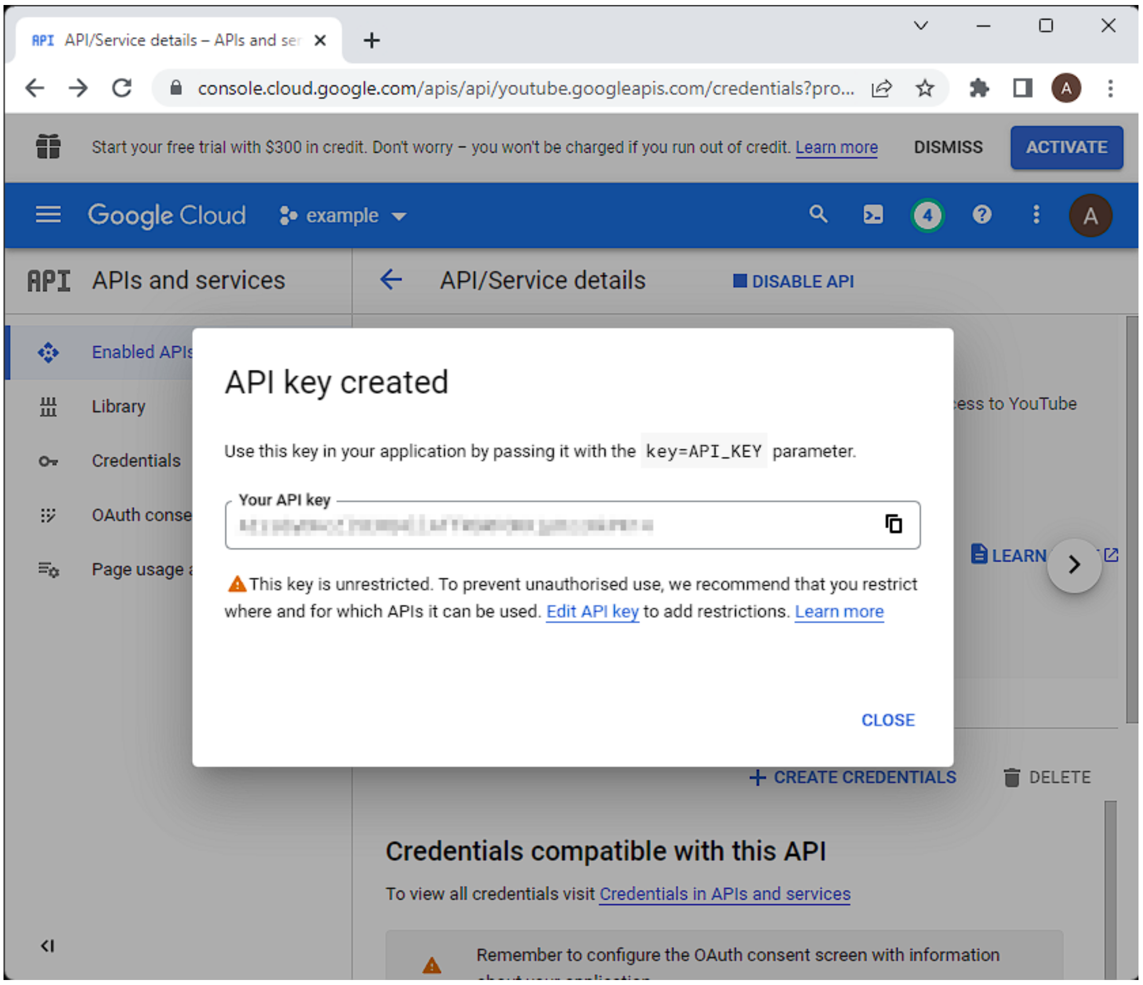


Fig 1.1. Landing page view of the Firebase console

With these steps completed, the project needs to be connected with the Spring Boot Application. To do that, an application.properties file, stored in the ‘resources’ directory needs to be opened. There are a few empty properties, one of which is *firebase.api.key*. It needs to be completed with a Web API key that can be taken from the General tab of the Project settings webpage of the created Firebase project. Also, on this settings page, there is a Service accounts tab, which enables to generate a new private key for Firebase Admin SDK. Having the key generated, its file name should be changed to *service-account.json* and the file should be placed in the resources folder of the application’s source code.

To integrate the application with the YouTube Data API it is required to complete the *youtube.api.key* field of the application.properties file of the project with an API key. To obtain the key, one needs to log in to the Google Developers Console, which is available at *https://console.cloud.google.com/*. As in the case of Firebase platform, here also a project should be created. In the project dashboard, an ‘APIs and services’ menu should be opened, where there is an ‘Enable APIs and services’ button, which needs to be clicked in order to browse the library and find the YouTube Data API v3. There, the YouTube API should be enabled, and a new credential should be created. This operation generates and displays an API key, which needs to be copied and pasted in a proper line in the Spring Boot Application's properties file.

Figure 5.2 shows the API key generation view in the Google Cloud Platform Console. Apart from the key generation, it is recommended to restrict the key with the use of settings panel available at the hyperlink 'Edit API key' visible in the figure. There access to the API key may be restricted so that the Spring Boot Application using the key can only make use of the YouTube Data API of the GCP project.



Start your free trial with $300 in credit. Don't worry – you won't be charged if you run out of credit. Learn more

API key created

Use this key in your application by passing it with the `key=API_KEY` parameter.

Your API key: [redacted]

This key is unrestricted. To prevent unauthorised use, we recommend that you restrict where and for which APIs it can be used. Edit API key to add restrictions. Learn more

Credentials compatible with this API

To view all credentials visit Credentials in APIs and services

Remember to configure the OAuth consent screen with information

Fig 1.2. API key generation in Google Cloud Platform Console

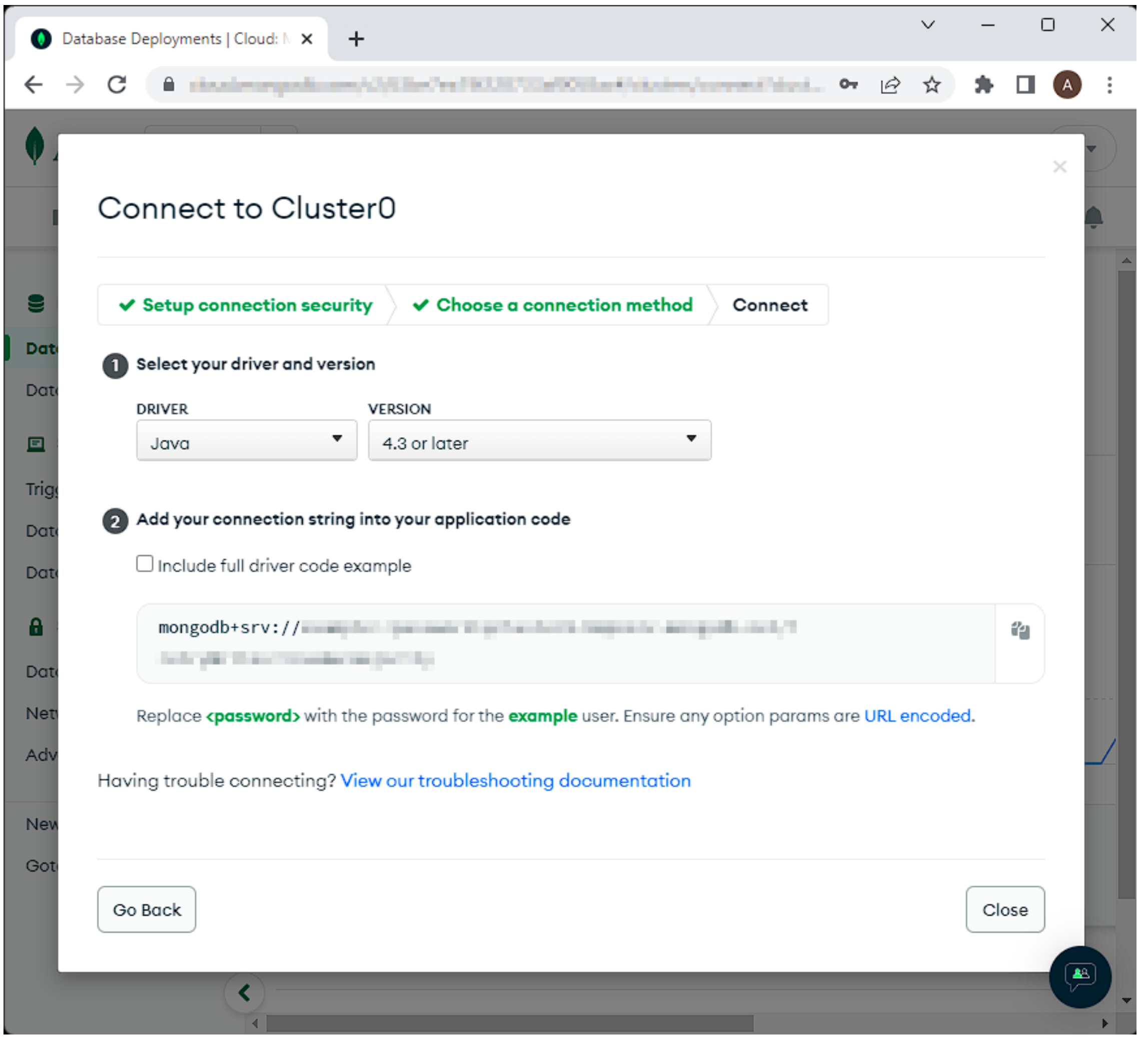
The third external service, which needs to be configured for the application functioning is a database running in the MongoDB Atlas environment. It is independent of the Google platforms, so one needs to sign up and create a new account there to be able to set up new resources. Having an account, a MongoDB project may be configured. During this process, a user needs to enter the project name. Also, an email addresses of co-administrators of the project may be added. The first step to proceed with a created project is to set up the IP Access List in the Network Access tab. This list defines the IP addresses, from which it is possible to connect to MongoDB clusters running in the project. There are two ways of filling that list:

- include the 0.0.0.0/0 IP address, which allows the network access from any device,
- include the IP addresses of the machines dedicated to running the project so that they are the only devices able to connect to the MongoDB cluster.

After defining the access, a new database cluster can be created. Depending on the needs, different MongoDB cluster tiers may be chosen, however, for the initial configuration, it is sufficient to choose the Shared M0 Sandbox Cluster with shared RAM and vCPU. Such cluster enables storage of up to 512 MB of data within a maximum number of 100 databases and 500 collections. This amount of storage is enough because the application makes use of three collections stored in one database. Additionally, a cloud provider and region for the cluster hosting needs to be chosen. In this case, the choice can depend on preferences of the user.

The next step in the process is to create a database user with a username and a password. Such a user gets read and write privileges by default, but they can be modified. This user's credential are required to put them in the connection string generated later by the database cluster. Finally, the cluster can be created, which redirects the user to the Database Deployments page, where a 'Connect' button can be found. One of the options, which is visible in a popup window that appear after clicking the button is 'Connect your application.' When a Java driver and its 4.3 version are chosen in this option's view, a connection string is generated by the MongoDB Atlas (the final stage of this process is shown in figure 5.3). This string needs to be copied, completed with credentials of the user created before, and pasted in the *spring.data.mongodb.uri* field of the *application.properties* file of the Spring Boot Application.

Summarizing, the steps presented in this subchapter described the process of configuring and connecting to the Spring Boot Application the external services, which provide authentication, YouTube comments downloading, and database running. An important remark is that the connection strings and keys, which were generated, should not be shared. Thus, the application.properties file of the project completed with this information should not be published or pushed to any public repositories for security reasons.


Connect to Cluster0

✓ Setup connection security ✓ Choose a connection method Connect

1 Select your driver and version

DRIVER: Java
VERSION: 4.3 or later

2 Add your connection string into your application code

☐ Include full driver code example

```
mongodb+srv://[illegible]
```

Replace <password> with the password for the example user. Ensure any option params are URL encoded.

Having trouble connecting? View our troubleshooting documentation

Fig 1.3. Connection string generation in MongoDB Atlas

1.1.2. Local environment

The Spring Boot Application, being the core of Emotion Classification backend, may be run locally on a private device and in a cloud environment. Both these ways have benefits coming from their usage. The application has an implemented functionality of analysing text comments and inferring emotions (one of: anger, fear, joy, love, sadness, surprise) from them. This service is based on the usage of the NLP model, but it was not implemented to work in the cloud environment. Thus, the application needs to be run locally to use this service. In the application.properties file of the project, there is a *spring.profiles.active* field, which enables setting the environment of running the application so that the NLP service can be managed. In the case of local running, it needs to be set to '*local*' so that the NLP service is active. A second aspect concerns downloading YouTube comments and saving them in a database. This action can be performed both on a local machine and remotely in a cloud, however, this task requires more computation power to be processed than the power sufficient for the basic functioning of the application. The reason for higher requirements of this service is that each downloaded YouTube comment is checked in terms of language. These checks are performed with the use of a computation-wise demanding library. Therefore, it is recommended to perform the comments downloading for database populating with the use of a local machine.

Having proceeded with the steps regarding external services configuration, the application is ready to be run. To do so, the root directory of the project's source code needs to be entered with a Windows PowerShell tool. There, a command presented in listing 5.1 needs to be executed. The output of this command shows the information, such as a confirmation of connecting properly to the MongoDB database. Additionally, the output prints a port number, to which the application requests should be directed. If no project configuration is changed, the application is available at *localhost:8080/*.

```
PS > ./mvnw spring-boot:run
```

Listing 1.1. Command running the Spring Boot Application

1.1.3. Cloud deployment

The Spring Boot Application of this project can be deployed to Azure Spring Apps, which is a service provided by Microsoft corporation within its Azure platform. To do that, one has to have a Microsoft account with a payment method added so that billing for cloud hosting can be performed. Within the Microsoft Azure Portal, which is available at *https://portal.azure.com/*, a resource group needs to be set up. In such resource group, there should be created an Azure Spring Apps instance, on which the application will be run in the next steps.

Another prerequisite is to have an IntelliJ IDEA IDE (*Integrated Development Environment*) installed on a computer. It can be a Community or Ultimate Edition of a minimum version 2020.1. In this IDE one needs to add the Azure Toolkit plugin, which can get connected with the Azure platform and helps in applications deploying.

The first step before the deployment is to set the *spring.profiles.active* field of the application.properties file of the project to '*deployed*.' This disables the emotion analysis service of the project. Then, in order to deploy to Azure, a user has to sign in within the plugin with the credentials of the Microsoft account and choose an active subscription. After that, the source code project should be opened with the IntelliJ tool. Then, the project in this tool should be right-clicked and a 'Deploy to Azure Spring Apps' from the 'Azure' menu should be chosen, as it is in figure 5.4.

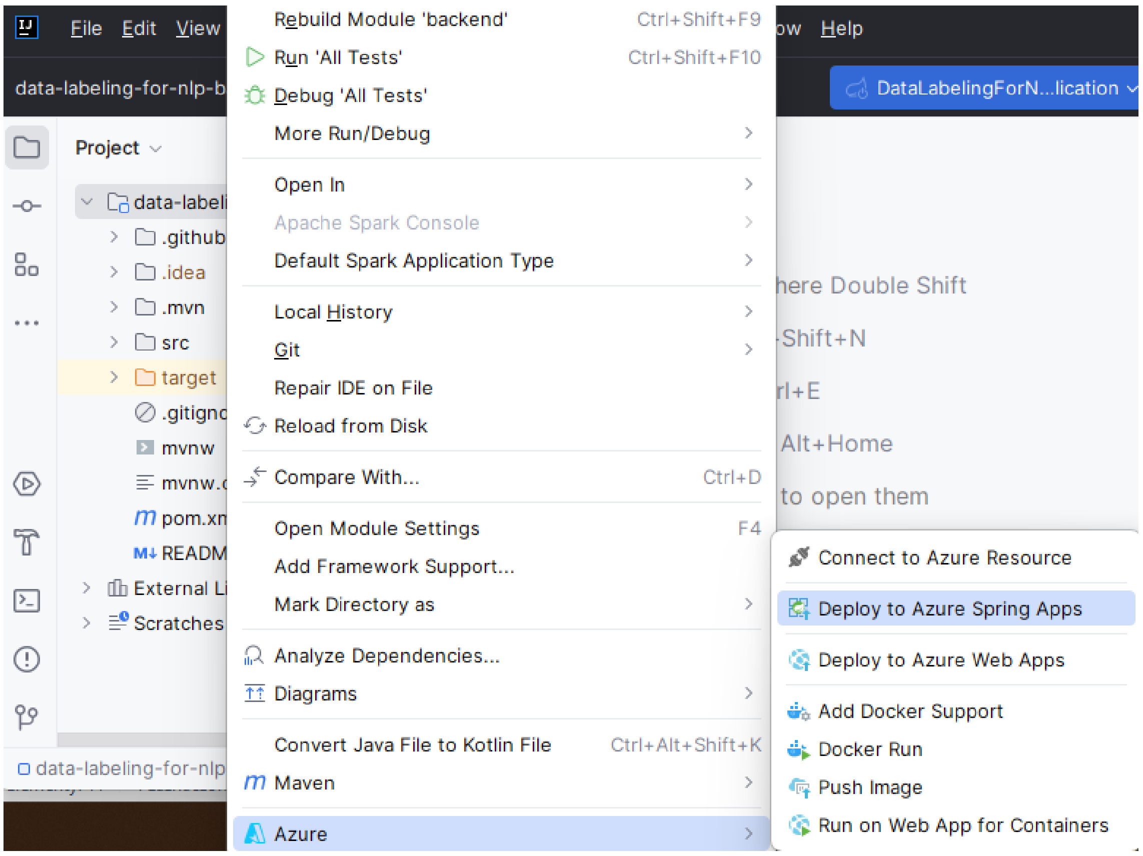
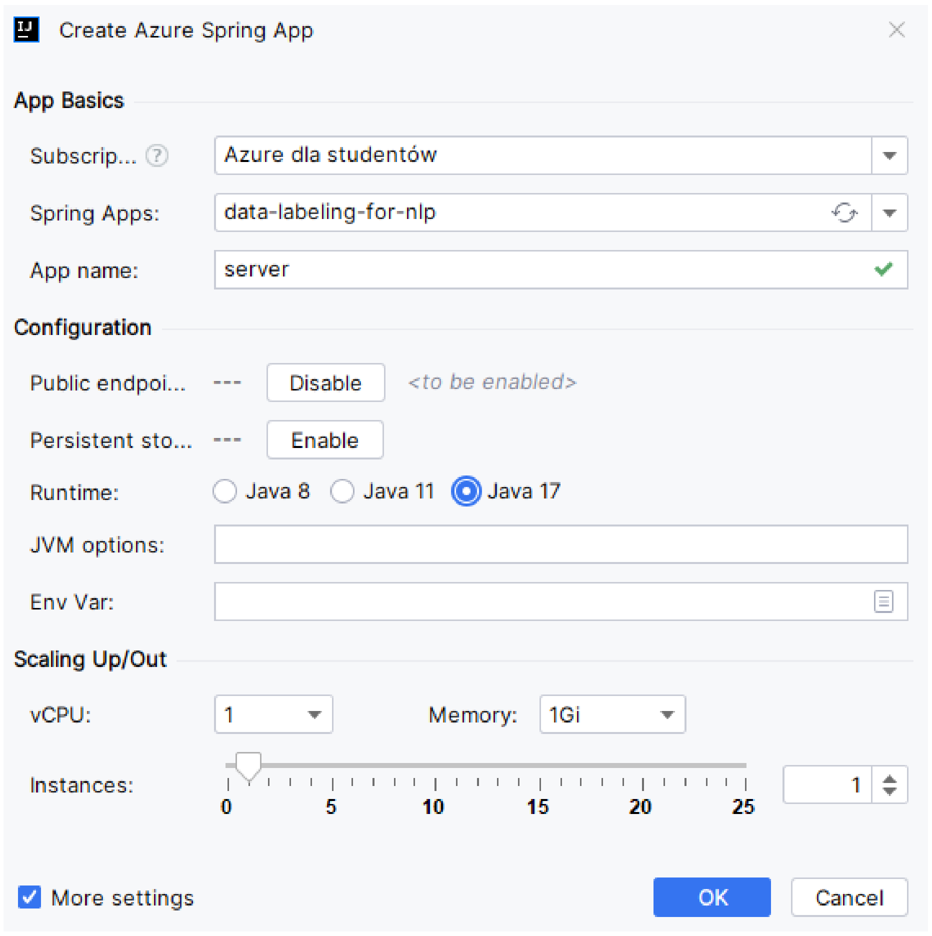


Fig 1.4. Opening Azure deployment tool in IntelliJ

Choosing the indicated option opens a 'Deploy to Azure' menu, which requires completing a few fields. These fields concern the maven artifact to be deployed and the Azure subscription. Additionally, the Azure Spring Apps instance created before needs to be chosen and a new application needs to be configured in the 'App' field. The essential information is that the 'Public endpoints' configuration should be enabled, and the Java 17 runtime should be chosen. Apart from that, the scaling needs to be set to at least one vCPU, 1GB of memory, and one instance running (Fig 1.5). However, as it was said, this configuration is not sufficient to run the YouTube comments downloading task. Therefore, the scaling configuration should be either increased or the scheduling of this task should be turned off (to do that, the *@Scheduled* annotation of the *CommentScheduler* class in the project's source code needs to be commented out) so that the task does not take place in the cloud. The last step is to set the Maven goal as '*package -DskipTests*,' which omits running tests when building the application.


Create Azure Spring App

App Basics

Subscrip...: Azure dla studentów
Spring Apps: data-labeling-for-nlp
App name: server

Configuration

Public endpoi... --- Disable <to be enabled>
Persistent sto... --- Enable
Runtime: Java 8 Java 11 [✓ Java 17]
JVM options:
Env Var:

Scaling Up/Out

vCPU: 1 Memory: 1Gi
Instances: 1

[✓] More settings

Fig 1.5. Azure Spring App configuration

Eventually, the 'Run' button can be clicked, which starts deploying the application. After the process gets finished, the output shows the URL address of the deployed backend module of this project. Application details are also available in the Microsoft Azure Portal.

1.2. Authentication and authorization

To call application's services, the user needs to get authenticated and authorize their requests to the Emotion Classification backend's endpoints with an ID token obtained during the authentication process. The application, run either locally or in a cloud environment, exposes three endpoints, which provide the authentication functionalities. These endpoints are presented in table 5.1. Request and response bodies of all these endpoints are of type *application/json*.

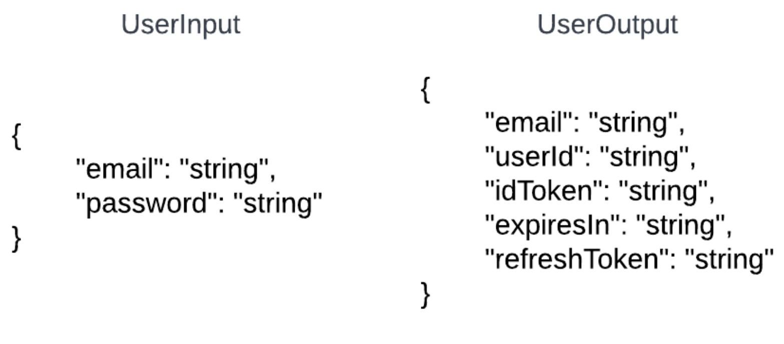
Table 1.1. Authentication API endpoints

No.	Path	Method	Input	Output
1	/api/v1/auth	POST	*UserInput* body object	*UserOutput* body object
2	/api/v1/auth/user	POST	*UserInput* body object	*UserOutput* body object
3	/api/v1/auth/token	POST	*RefreshTokenInput* body object	*RefreshTokenOutput* body object

Endpoints from table 5.1 provide the following functions:

- */api/v1/auth* – sign up service, which creates an account of a user. Requires a valid email address and a password composed of at least six characters.
- */api/v1/auth/user* – sign in service, which logs in a user and returns authorization data for other services. Requires a valid email address and password.
- */api/v1/auth/token* – refresh token service, which returns a new ID token for authorizing application requests. Requires a valid refresh token.

Both the *UserInput* and *UserOutput* objects are presented in figure 5.6. The first one contains two string fields dedicated to storing email address and password of the user. The *UserOutput* object contains user’s email address, user ID, ID token, ID token expiration time, and refresh token. All the data in this object is of type string.



UserInput

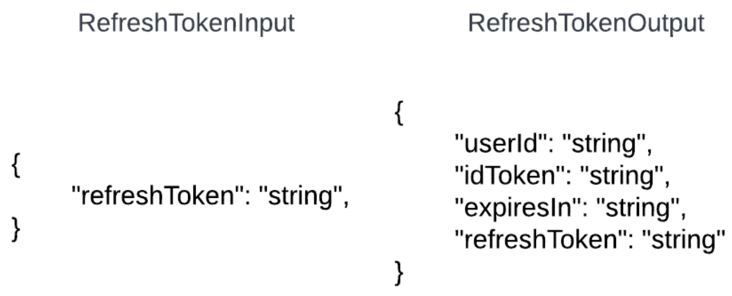
```
{
        "email": "string",
        "password": "string"
}
```

UserOutput

```
{
        "email": "string",
        "userId": "string",
        "idToken": "string",
        "expiresIn": "string",
        "refreshToken": "string"
}
```

Fig 1.6. UserInput and UserOutput objects

The *RefreshTokenInput* and *RefreshTokenOutput* objects (Fig 1.7) are used to refresh ID tokens for authorization. The first one includes one string field for the refresh token, whereas the second one contains four string fields. These fields store user ID, ID token, ID token expiration time, and refresh token. In general, ID tokens returned by the authentication services have their expiration time represented by the ‘*expiresIn*’ field of the *UserOutput* and *RefreshTokenOutput* objects. Due to that, refresh tokens need to be used in order to obtain a new ID token.


RefreshTokenInput

```
{
	"refreshToken": "string",
}
```

RefreshTokenOutput

```
{
	"userId": "string",
	"idToken": "string",
	"expiresIn": "string",
	"refreshToken": "string"
}
```

Fig 1.7. RefreshTokenInput and RefreshTokenOutput objects

In case one of the authentication services throws an exception, an error message with a ‘400 BAD REQUEST’ status code is returned. Table 1.2 shows the error messages that may be returned.

Table 1.2. Authentication error messages

Error message	Error meaning
Email address is invalid	Email address passed to the service does not have a valid form.
Password must be at least 6 characters	Password passed to the service is too short. It needs to contain at least six characters.
Failed to sign up	Firebase platform failed to create an account.
Failed to set user claims	Firebase platform failed to set user permissions.
Failed to sign in	Firebase platform failed to log in a user.
Failed to refresh token	Firebase platform failed to generate a new ID token.

The last important aspect concerns a fact, that all requests directed to application services, except the ones described in this subchapter, need to be authorized with the use of an ID token. This token can be obtained during the authentication process done by one of the three presented endpoints. To authorize the request, it needs to contain an ‘Authorization’ header with a value ‘*Bearer <idToken>*,’ where the *‘<idToken>’* part should be replaced with the obtained ID token.

1.3. Services endpoints

Apart from the authentication endpoints, the application provides other services, which are divided into three resource groups. These are: *comment*, *assignment*, and *emotion*.

1.3.1. Comment resource

Endpoints created under within the *comment* resource enable populating the database with YouTube comments, accessing the stored comments, and fetching them from the database for labelling reasons. They are described in table 5.3. Request and response bodies of these endpoints are of type *application/json*.

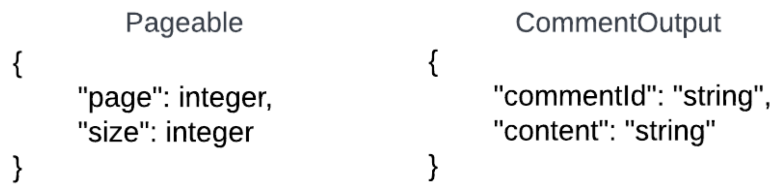
Table 1.3. Comment API endpoints

No.	Path	Method	Input	Output
1	/api/v1/comment/youtube	GET	-	Body with list of *CommentOutput* objects
2	/api/v1/comment	GET	String with user ID, String with comments number	Body with list of *CommentOutput* objects
3	/api/v1/comment/all	GET	*Pageable* body object	Body with paginated *CommentOutput* objects

Endpoints from table 5.3 provide the following functions:

- */api/v1/comment/youtube* – service for downloading most relevant Polish comments of most popular YouTube videos at the time. The comments have between 5 and 250 words.
- */api/v1/comment* – service for fetching a given number of comments to be assigned with an emotion by a user with given ID.
- */api/v1/comment/all* – service for fetching paginated list of comments stored in the database.

The *Pageable* is an object that enables paginating the output. It consists of a page number field and size field, which indicates the page's size. The *CommentOutput* object is composed of two fields: comment ID and content of the comment. The structure of both these objects is presented in figure 5.8.



```
        Pageable
{
    "page": integer,
    "size": integer
}

        CommentOutput
{
    "commentId": "string",
    "content": "string"
}
```

Fig 1.8. Pageable and CommentOutput objects

The execution of the services may cause an error, which produces a response containing the error message and a ‘400 BAD REQUEST’ status code. Table 1.4 shows the meaning of errors that may occur.

Table 1.4. Comment error messages

Error message	Error meaning
There is no comment with entered id	Comment with entered ID does not exist in the database.
There is no user with entered id	User with entered ID does not exist in the database.
Comments number is not an integer value	The entered comments number is not a valid number.
Comments number needs to be a number between 1 and 100	The entered comments number exceeds the accepted range from 1 to 100.
Error while fetching YouTube videos list	YouTube platform failed to return the list of most popular videos.

1.3.2. Assignment resource

The assignments resource, managing *comment-emotion assignments*, provides functionalities of uploading to Emotion Classification backend the labelled comments as well as exporting these data in a form of a CSV file. The endpoints exposing these services are presented in table 5.5. Request and response bodies of these endpoints are of type *application/json*. An additional remark is that in the project’s source code the input and output objects of the services are called *CommentEmotionAssignmentInput* and *CommentEmotionAssignmentInput*. However, to make these names fitting the table better, they are presented in a shorter form: *AssignmentInput* and *AssignmentOutput*.

Table 1.5. Comment-emotion assignment API endpoints

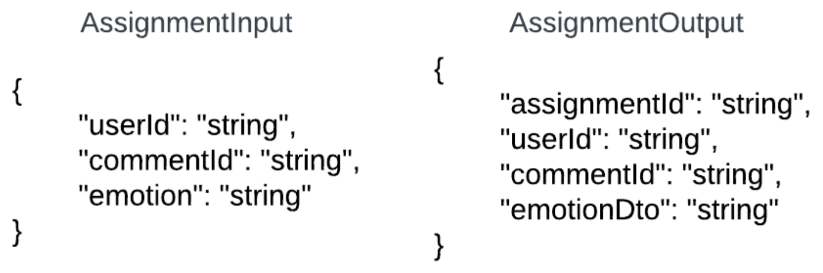
No.	Path	Method	Input	Output
1	/api/v1/assignment	POST	Body with list of *AssignmentInput* objects	Body with list of *AssignmentOutput* objects
2	/api/v1/assignment/dataset	GET	-	CSV file

The endpoints, presented in table 5.5, are responsible for the following services:

- */api/v1/assignment* – service for uploading the comments labelled with an emotion out of six (anger, fer, joy, love, sadness, surprise) or an ‘unspecifiable’ label, which means that a comment could not be assigned with an emotion. It accepts a list of objects containing ID of the assigned comment, assigned emotion, and ID of the user who created that assignment.

- */api/v1/assignment/dataset* – exports the comment-emotion assignments as a CSV file containing two columns: comment and emotion. The first column has a list of comment contents, whereas the second one includes emotions assigned to them by the application's users. The service omits the 'unspecifiable' labels during exporting data. Additionally, if there are numerous assignments for a single comment, the most frequent one is chosen to be exported. The file can be downloaded after requesting this endpoint.

To transfer the data, the assignment service makes use of the *AssignmentInput* and *AssignmentOutput* objects, presented in figure 5.9. The first one contains three string fields: ID of the user, who has created the label, ID of the labelled comment, and the assigned emotion. The emotion must match one of the seven acceptable options, which are ANGER, FEAR, JOY, LOVE, SADNESS, SURPRISE or UNSPECIFIABLE. As mentioned, the last label should be assigned to the comments particularly difficult to analyse and infer one emotion from them. The *AssignmentOutput* object includes four fields. The first of them is the ID of a newly created assignment, whereas the other three (user ID, comment ID, and emotion DTO) represent the same entities as those in *AssignmentInput* object.



```
AssignmentInput
{
    "userId": "string",
    "commentId": "string",
    "emotion": "string"
}
```

```
AssignmentOutput
{
    "assignmentId": "string",
    "userId": "string",
    "commentId": "string",
    "emotionDto": "string"
}
```

Fig 1.9. AssignmentInput and AssignmentOutput objects

The services concerning managing the assignments resource may fail during their execution. In such case, an error message is returned with a '400 BAD REQUEST' status code. Table 1.6 presents the error messages that may be returned by Emotion Classification backend.

Table 1.6. Assignment error messages

Error message	Error meaning
There already exists this user's assignment for this comment	The assignment sent to be saved in the database is already present there.
There is no user with entered id	User with entered ID does not exist in the database.
Entered emotion does not exist	The entered emotion is not valid. It needs to fit the set of acceptable values (ANGER, FEAR, JOY, LOVE, SADNESS, SURPRISE, or UNSPECIFIABLE).
Failed to write to csv file	The application failed to write the assignments to the CSV file.

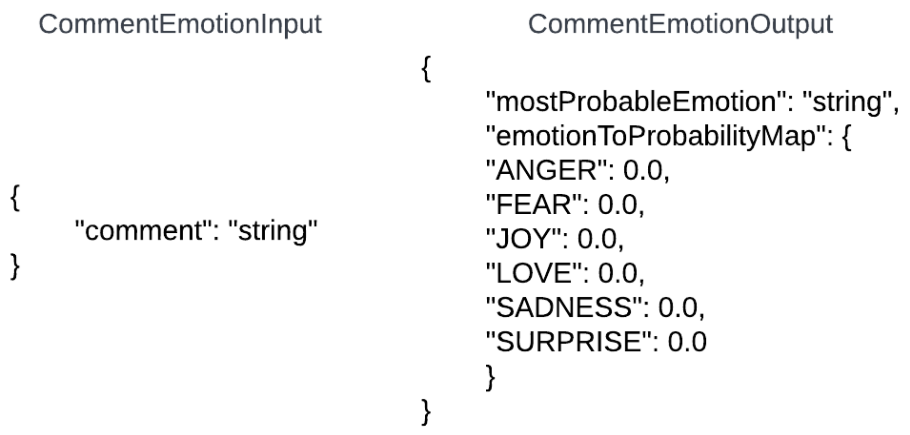
1.3.3. Emotion resource

This function of Emotion Classification backend provides a possibility of inferring an *emotion* from a comment passed to the application. However, this function is available only when running the application in the local environment. In such case the endpoint presented in table 5.7 is active. When the application is deployed in the cloud, this service is disabled. Request and response body of the presented endpoint are of type *application/json*.

Table 1.7. Emotion analysis endpoint

No.	Path	Method	Input	Output
1	/api/v1/emotion	POST	*CommentEmotionInput* body object	*CommentEmotionOutput* body object

The */api/v1/emotion* endpoint takes a *CommentEmotionInput* object (Fig 1.10), which contains a string comment field. This enables passing a comment to the service, which is analysed by an NLP model in order to infer an emotion from it. After the analysis, a *CommentEmotionOutput* object (Fig 1.10) is returned in a response body. It includes two fields, the first of which is the most probably expressed emotion, which was deduced by the service. The next field contains a map with all emotion accompanied by a floating number representing the probability of the emotion being expressed by the passed comment.


CommentEmotionInput

```
{
	"comment": "string"
}
```

CommentEmotionOutput

```
{
	"mostProbableEmotion": "string",
	"emotionToProbabilityMap": {
	"ANGER": 0.0,
	"FEAR": 0.0,
	"JOY": 0.0,
	"LOVE": 0.0,
	"SADNESS": 0.0,
	"SURPRISE": 0.0
	}
}
```

Fig 1.10. CommentEmotionInput and CommentEmotionOutput objects

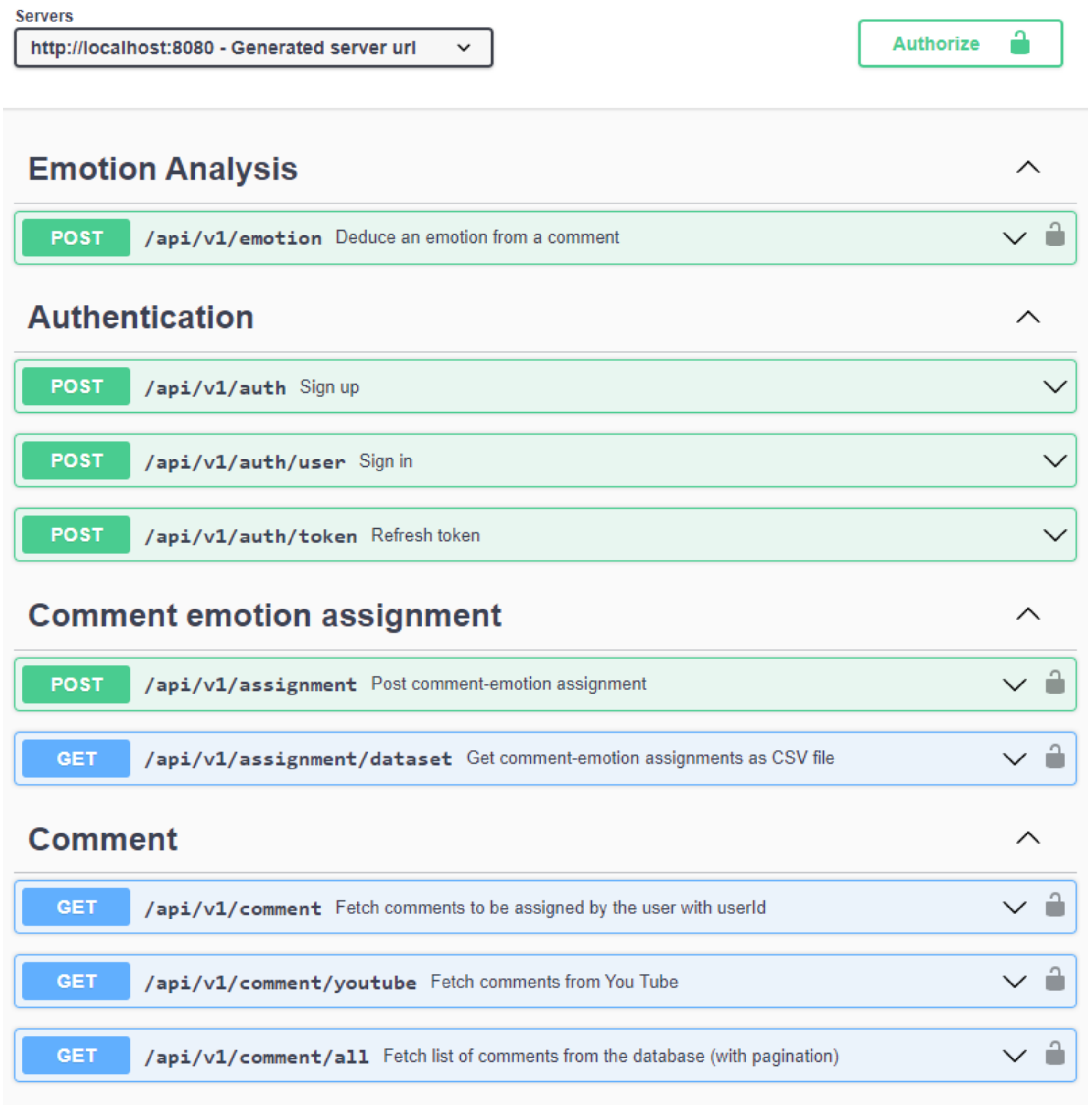
Table 1.8 describes the error messages that may be returned from the described service when its execution fails. In such case the error message is returned with a ‘400 BAD REQUEST’ status code.

Table 1.8. Emotion error messages

Error message	Error meaning
Failed to load model responsible for inferring emotions from comments	The NLP model loading process failed. It may be not present in project’s resource directory.
Failed to infer an emotion from a given comment	A comment input is invalid so that the NLP model failed to infer an emotion from it.

1.4. Swagger UI

The *Swagger UI* function is turned on within Emotion Classification backend application to help in using and understanding the REST API, because the API represented by this tool has a visual and interactive form. When the application is run, Swagger UI is available at the root URL path (‘/’) and at the ‘*/swagger-ui/index.html*’ path. Figure 5.11 depicts the view of this platform when the application is run in local environment (the emotion analysis functionality is available). The tiles with endpoint paths visible there represent application services. Clicking each of them unrolls the view, from which a request to the endpoint can be sent. Each such card enables setting proper request parameters or request body and viewing the response.


Servers

http://localhost:8080 - Generated server url

Authorize

Emotion Analysis

POST /api/v1/emotion Deduce an emotion from a comment

Authentication

POST /api/v1/auth Sign up

POST /api/v1/auth/user Sign in

POST /api/v1/auth/token Refresh token

Comment emotion assignment

POST /api/v1/assignment Post comment-emotion assignment

GET /api/v1/assignment/dataset Get comment-emotion assignments as CSV file

Comment

GET /api/v1/comment Fetch comments to be assigned by the user with userId

GET /api/v1/comment/youtube Fetch comments from You Tube

GET /api/v1/comment/all Fetch list of comments from the database (with pagination)

Fig 1.11. Swagger UI page of the Emotion Classification backend application run locally

Additionally, some of the endpoint cards are marked with a padlock icon, which signifies, that these endpoints require authorizing with a valid ID token. When this icon or an ‘Authorize’ button available at the top of the page is clicked, a popup menu appears. This menu enables passing the ID token so that the requests sent to the application can be authorized.