Department of Computer Science
Technical University of Cluj-Napoca

Polyglot

-master a foreign language by reading-

Developed By: Borbála Fazakas
GitHub Repository: ?
Group: 30432

Laboratory Assistant: Teodora Vezan

Contents

Chapter 1

Project Specification

Polyglot is an application which helps its users learn foreign languages while reading books, the news or any texts that they enjoy.

This language learning application is targeted at users who already know the basics of the grammar and vocabulary of a certain language, but would like to **enrich they vocabulary and hit fluency as fast as possible**. The core idea of the app is based on the observations that:

- seeing a word in a context helps memorizing it and incorporating it in your active vocabulary
- reading promotes word precision, as written word is more nuanced than spoken word
- and last but not least, learning languages by reading allows users to combine a pleasant activity with a useful one

Apart from **language learners (i.e. students)**, the application also provides a platform for **language teachers** to easily create and share courses with several lessons and practice exercises with their students. Furthermore, the **admins** can manage the data of the users and mark the teachers who can provide proof of their qualification for teaching a certain language as "certified".

The application will be implemented in form of a web app but can be further extended with a mobile interface, to allow users to always carry it in their pocket.

1.1 Domain Model Diagram

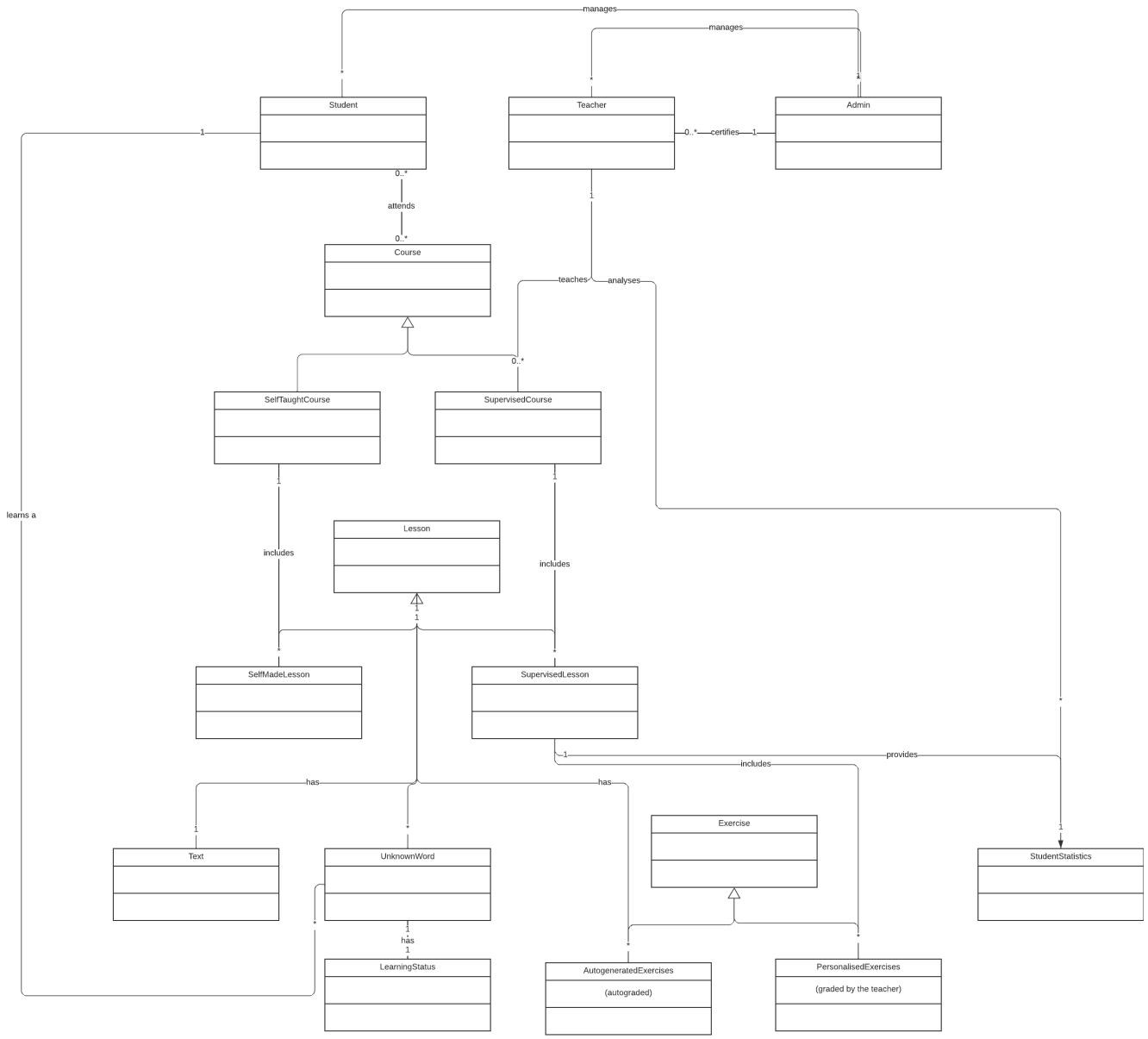

Figure 1.1: The Domain Model Diagram

Chapter 2

Use-Case Model

This section describes who the users of the application are and what they can use the application for.

2.1 Users and stakeholders

The application supports 3 types of users, namely:

1. **language students**: users who attempt to learn a new language. They can attend courses and learn lessons by reading texts, marking and learning the unknown words and solving practice exercises.
2. **teachers**: users who attempt to help students to learn a language by setting up courses with several lessons, providing a text and practice exercises for each lesson, and grading these exercises. They can also analyse the automatically generated statistics about the results and progress of the students attending their courses.
3. **admins**: they have an administrative role, they can manage the data of the other users and upon request they can mark qualified teachers (i.e. teachers who have a university diploma) as "certified".

The stakeholders interested in the outcome of the project are:

1. **the users**, i.e. the customers, anyone learning/teaching a language, possibly even language schools.
2. **the steering committee**, i.e. the laboratory teacher and the professor teaching the course

2.2 Use-Case identification

2.2.1 Creating a Self-Taught Course

- Level: User-Goal
- Main actor: Student
- Main success scenario:
  1. the Student selects the (unique) name of the course
  2. the Student creates the course

2.2.2 Creating a Supervised Course

- Level: User-Goal
- Main actor: Teacher
- Main success scenario:
  1. the Teacher selects the *unique) name of the course
  2. the Teacher creates the course
  3. a unique joining code is created for the course
  4. the teacher privately shared the joining code with anyone they want to join the course

2.2.3 Creating a Self-Taught Lesson

- Level: User-Goal
- Main actor: Student
- Main success scenario:
  1. the Student selects the Self-Taught Course to add a new lesson to
  2. the Student specifies a unique title for this new lesson
  3. the Student uploads a text file for this new lesson, from their computer
- Extension (not in the basic app version, but in a possible future, improved version of it)
  1. 3.b.1 the Student selects a topic to study
  2. 3.b.2 the app provides a list of newspaper articles relevant to the selected topic
  3. 3.b.3 the Student selects one of the newspaper articles as the lesson's text

2.2.4 Creating a Supervised Lesson

- Level: User-Goal
- Main actor: Teacher
- Main success scenario:
  1. the Teacher selects one of their supervised Courses to add the Lesson to
  2. the Teacher specifies a unique title for this new lesson
  3. the Teacher uploads a text file for this new lesson, from their computer
  4. the Teacher adds personalised exercises to the lesson
- Extension (not in the basic app version, but in a possible future, improved version of it)
  1. 3.b.1 the Teacher selects a topic to study
  2. 3.b.2 the app provides a list of newspaper articles relevant to the selected topic
  3. 3.b.3 the Teacher selects one of the newspaper articles as the lesson's text

2.2.5 Adding Personalized, Auto-Graded Exercises to a Lesson

- Level: User-Goal
- Main actor: Teacher
- Main success scenario:
  1. the Teacher selects the lesson to add an exercise to
  2. the Teacher selects one of the available exercise templates
  3. the Teacher fills in the exercise template with the relevant data
  4. the Teacher specifies the correct answers for the Exercise

2.2.6 Studying a lesson

- Level: User-Goal
- Main actor: Student
- Main success scenario:
  1. the Student selects a course
  2. the Student selects one of the lessons of the course
  3. the Student reads the text provided for the lesson
  4. the Student marks the unknown words from the text
  5. the unknown words' translation is provided for the student
  6. practice exercises for the unknown words are automatically generated
  7. the Student learns the previously unknown words by solving the automatically generated practice exercises

2.2.7 Practicing the knowledge gained from a supervised lesson

- Level: User-Goal
- Main actor: Student
- Main success scenario:
  1. the Student selects a course
  2. the Student selects one of the lessons of the course
  3. the Student solves the personalised, auto-graded exercises provided by the teacher
  4. the Students gets feedback of their results and can see the correct answers

2.3 Use-case diagrams

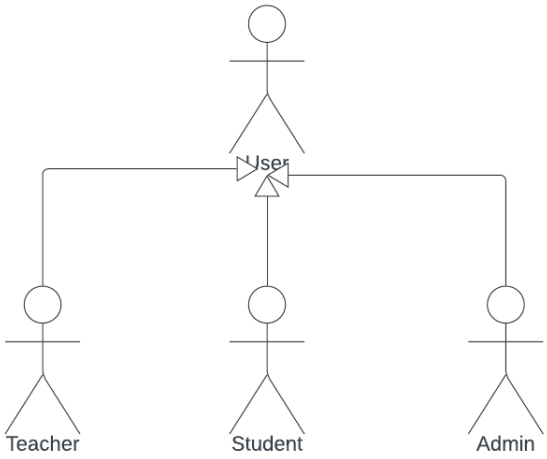

Figure 2.1: The user types

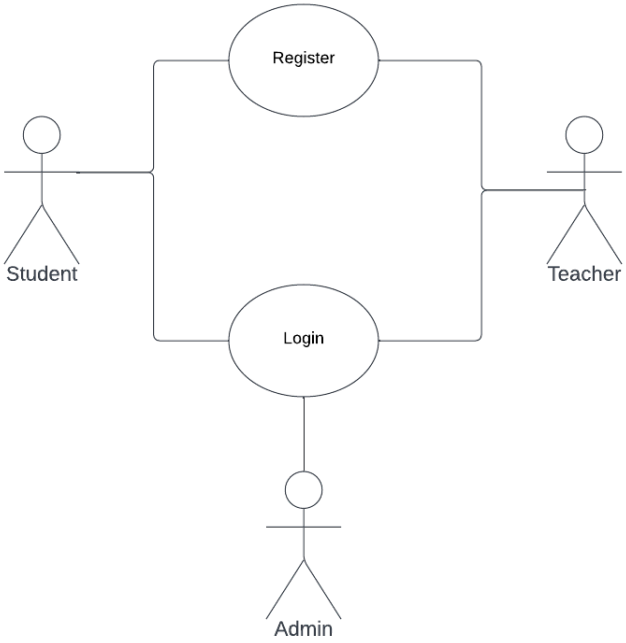

Figure 2.2: General use cases

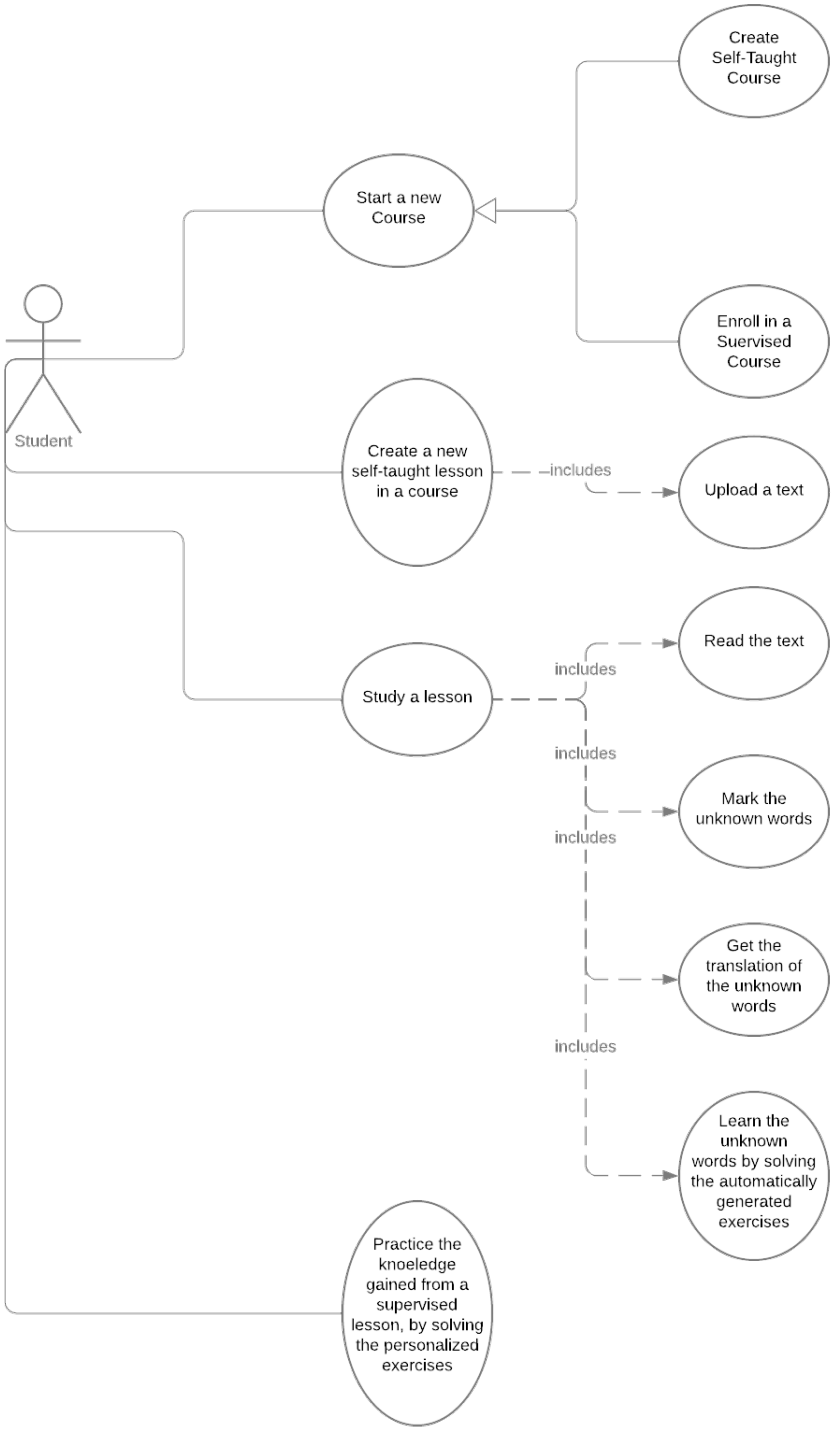

Figure 2.3: The use cases for students

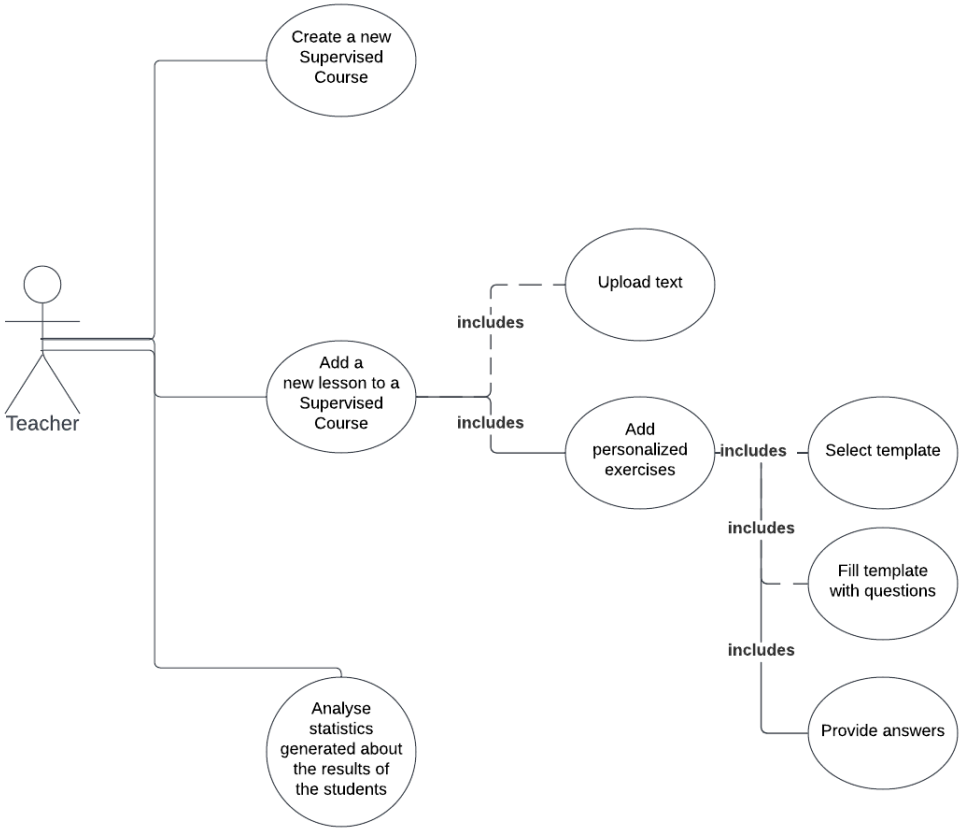

Figure 2.4: The use cases for teachers

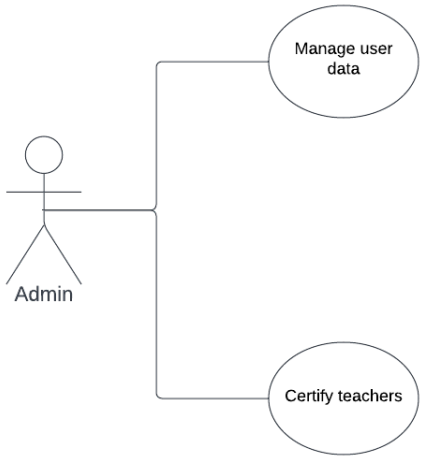

Figure 2.5: The use cases for admins

Chapter 3

Architectural Design

The following chapter describes the high-level architecture of the project.

3.1 Conceptual Architecture

The Minimum Viable Product (MVP) is a **Web Application**, but a further extension for a Mobile (Android/IOS) Platform is planned:

- a Web Application accessible through the browser provides general access for everyone, can be used both from a PC, laptop, smartphone, etc, so this is the prioritized platform,
- a Mobile Application is a preferable improvement, because many users would prefer to learn languages on their phone in those short time intervals, when they're away from their home, probably transitioning between two major activities (while staying in a traffic jam, waiting for an appointment, ...). In such cases, users do not have a PC/laptop at hand, but they always carry their smartphones with themselves.

Furthermore, the architecture of the web-application will be the well-known **Client-Server** one.

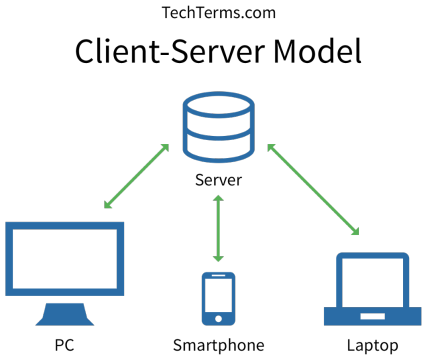

Figure 3.1: Architecture

Responsibilities:

- **Server**
  - connection to the **database** (see the next section for more information)
  - business logic: handling the logical part of the actions accessible to each user, ensuing that all data saved in the database is valid, implementing the business logic for tracking the students' achievements, generating the statistics for the teachers, ...
- **Client**
  - presentation
  - data acquisition from the users

Thin client model: he Client is planned to be as thin as possible, with the goal of reusing all the functionalities implemented on the Server-side for the upcoming mobile application. This way, each functionality needs to be implemented only once.

Furthermore, inside the Server, a **Layered Architecture** is planned, where

- **Controllers** take up requests from clients and prepare the results computed by the Services to be sent back as a response in an appropriate form.
- **Services** implement the Business Logic: all the domain-specific knowledge and logic of the application, for example, data validations, status updates (learning progress), ...
- **Repositories** abstract and implement the connection and communication with the database, ensuring that all the data is correctly persisted.

3.2 Package Diagram

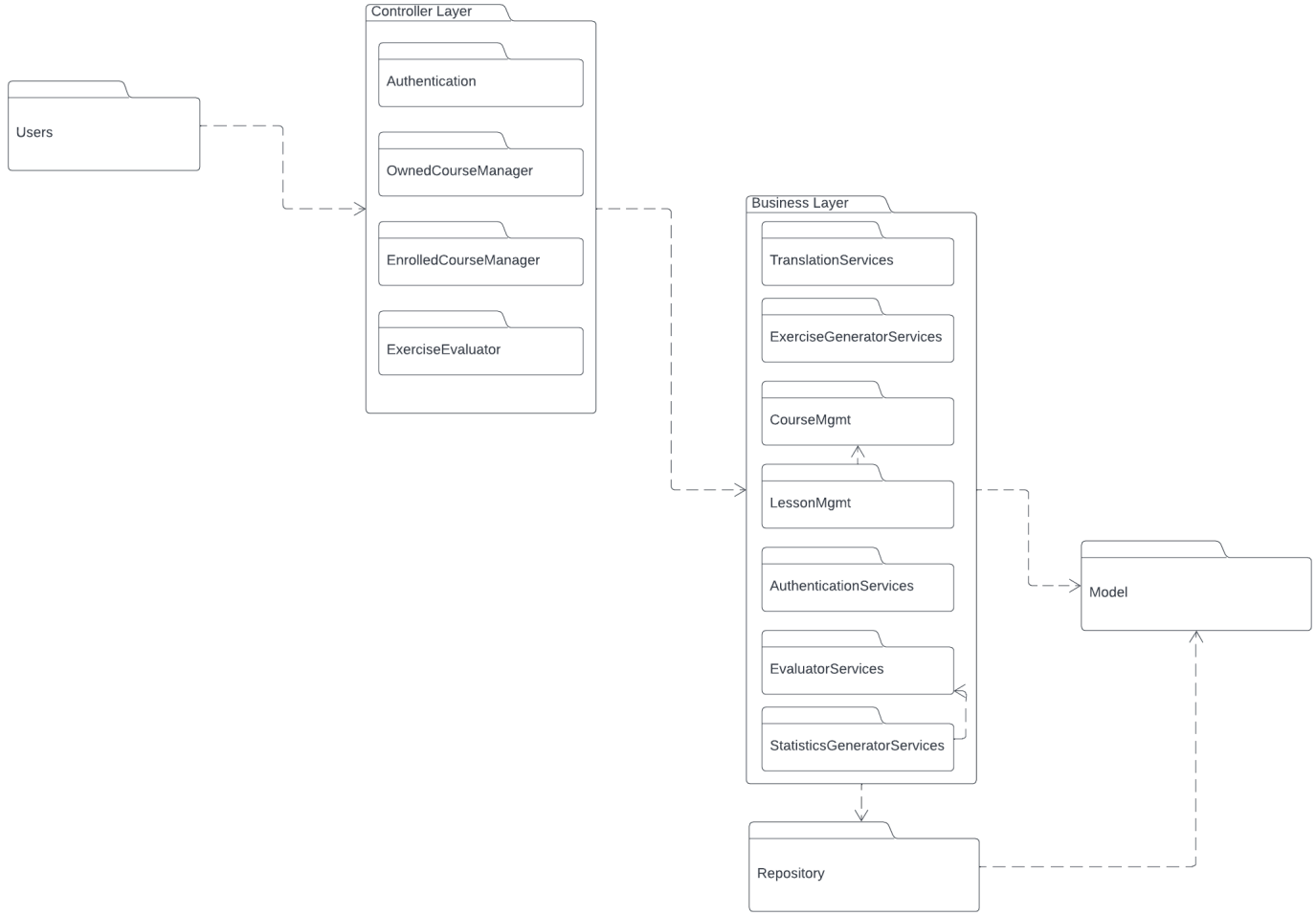

Figure 3.2: Package Diagram

3.3 Class Diagram of the Model

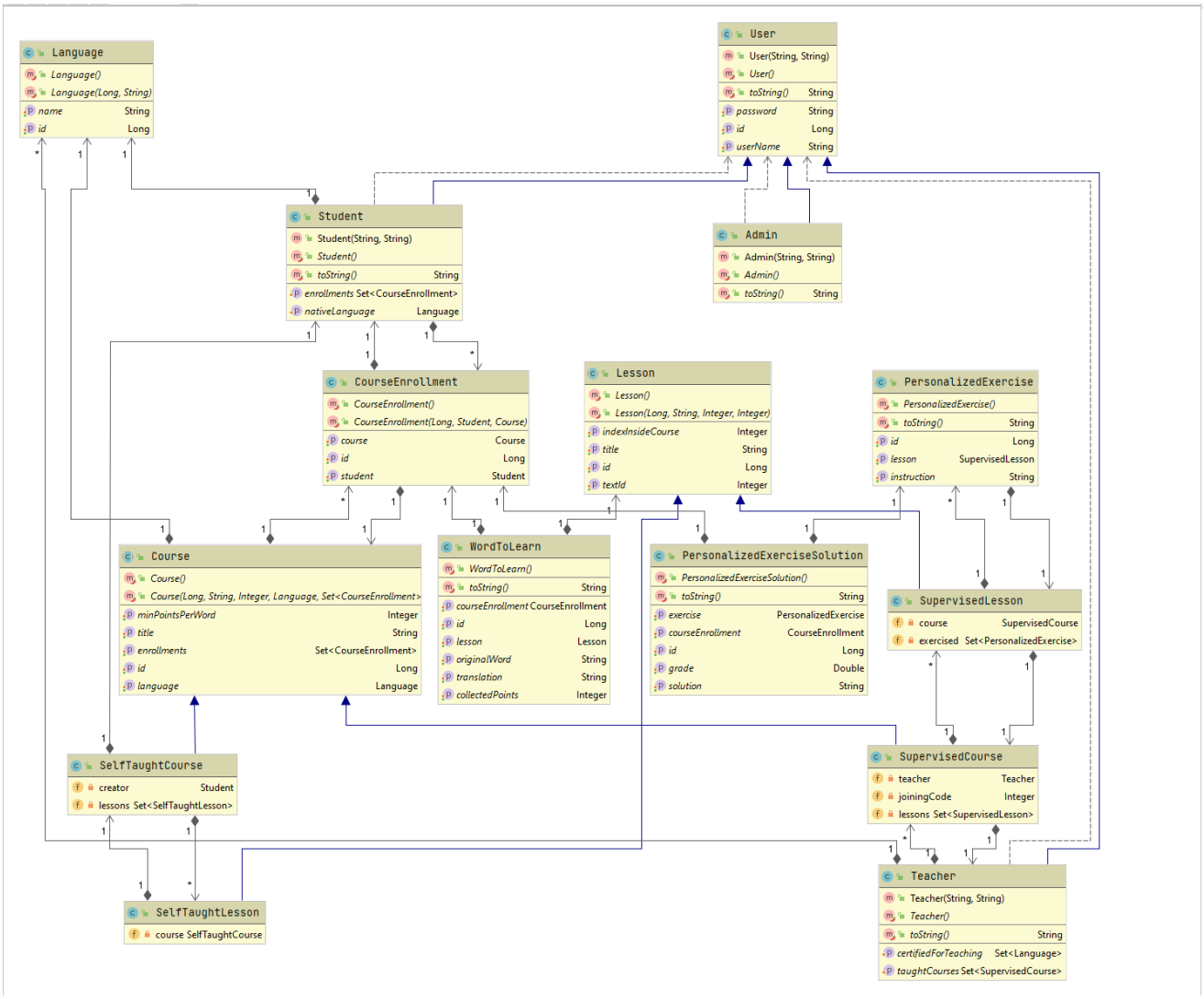

Figure 3.3: Class Diagram

3.4 Database Diagram

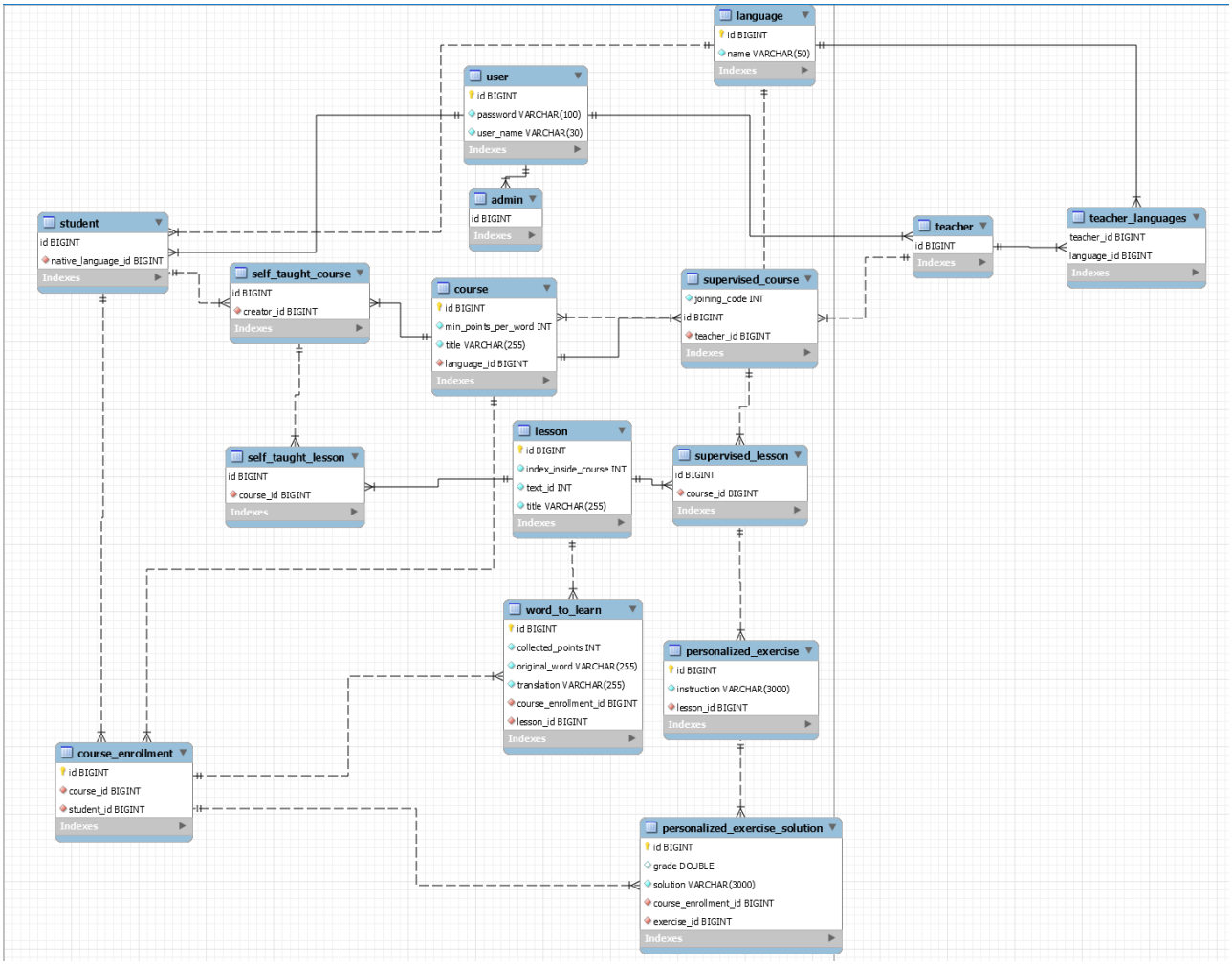

Figure 3.4: Database Diagram

3.5 Sequence Diagram

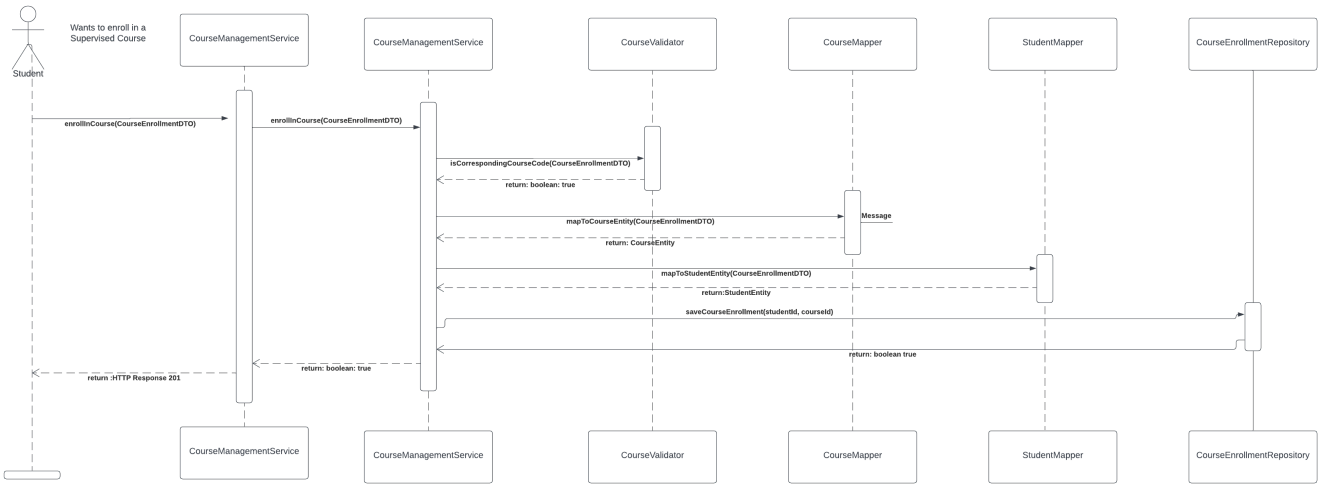

Figure 3.5: Sequence Diagram

3.6 Activity Diagram

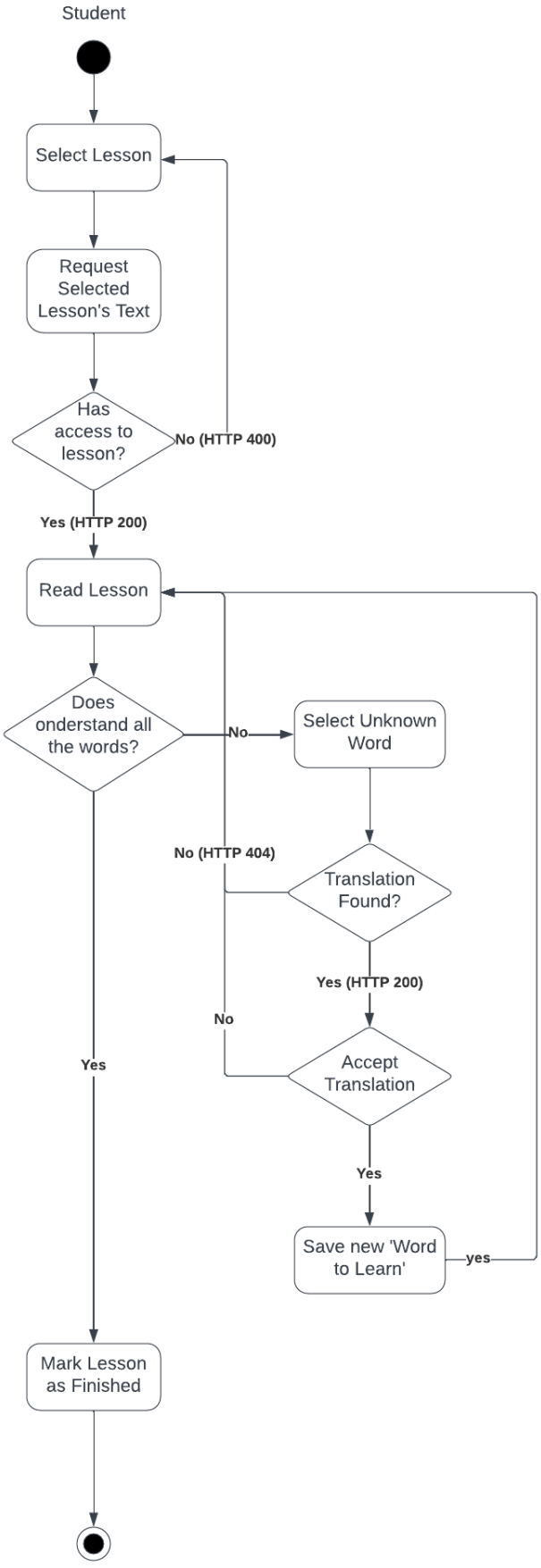

Figure 3.6: Activity Diagram

Chapter 4

Supplementary specifications

4.1 Non-functional requirements

4.1.0.1 Accessibility across devices

The application should run on any device capable of internet connection, independently from the underlying operating system, assuming that it can run a browser, in order to give access to as many users as possible.

4.1.0.2 Portability

As already mentioned, the Server side should be designed such that the Database and the Business Logic layer can serve a Mobile Application as well, in the future, without any major modifications. This ensures that the work of the developers doesn't need to be duplicated: the current business logic and data access layer should be reused for other platforms as well.

4.1.0.2.1 Security

Despite the application not dealing with any highly-sensitive personal data, special care must be provided to the authentication and authorization processes, and to the safe storage of the users' passwords: even if an admin has access to the database, they should not be able to find out and steal the password of any of the users.

4.1.0.2.2 Availability

We aim to build a product with a maximum monthly downtime of 1%. Thus, we can only use external services with a Service Level Agreement (SLA) that gurantees a monthly uptime percentage of at least 99%. This is relevant, beause if users experienced many frictions in the execution of the application, they could simply not enjoy the functionalities provided by it, even if they were otherwise useful for them.

4.2 Design constrains

4.2.1 Expectations from the stakeholders

Based on the expectations expressed by the stakeholders, the application will be developed

- in **Java** language for the Server Side
- relying on the **Spring Boot** framework, for building a production-ready Web App in a time-efficient and flexible manner.

4.2.2 Further technical constraints

4.2.2.1 Database

As previously specified, a database is needed for data storage. Given that we expect the application to have a few thousands of users, the amount of data to be stored does not exceed the amount from an average web application, and database queries are not performance critical, so a **relational database** seems a good fit. In particular, we'll be using a **MySql** database server and will be accessing it through **Hibernate**, which makes the data access flexible and easily reconfigurable.

4.2.2.2 Translation

A critical part of the application is translating the unknown words of a user to their native language. For this functionality, we'll be relying on **Google's Cloud Translation API** (more information here: `https://cloud.google.com/translate`. For this, a Google Cloud project must be created with the required API key. The translations will be performed on the Server side (because we want to have a think server and a thin client, see the 'Conceptual Architecture' for the explanation).

Note that the Service Level Agreement of Cloud Translation API guarantees a monthly uptime percentage of 99.9%, which is well above our expectations specified above.

4.2.2.2.1 The Client Side

The client side code will be written in JavaScript, and optionally may be implemented relying on React. The MVP is expected to have a functional client with pure JavaScript only, but if the time constraints are permissive, switching from JavaScript to TypeScript and using React is preferable.

4.2.2.3 Security

Access to the Application will be granted only through **Spring Security**, which ensures that only the authenticated users are granted access and that the application is resistant to common attacks.

Chapter 5

API Documentation

5.1 POST /polyglot/register

Request sample:

- body(JSON): {
  name: "sample name"
  password: "sample password",
  emailAddress: "sample email address",
  userType: "STUDENT",
  nativeLanguage: "English" }

Responses:

- 200 OK - user successfully registered
- 403 FORBIDDEN - username is already taken

5.2 POST /polyglot/login

Request sample:

- body(JSON): {
  name: "sample name"
  password: "sample password",

Responses:

- 200 OK - user successfully registered

  - sample response content:
    body(JSON): {
    usaerName: "sample name",
    role: "STUDENT",
    accessToken: "sample token",
    tokenType: "Bearer" }

- 401 UNAUTHORIZED - invalid username or password

5.3 GET /polyglot/get_lesson_file

Request sample:

- lessonId=3

Responses:

- 200 OK - file successfully found and returned
  - response content: byte[], representing the lesson's file
- 401 UNAUTHORIZED - active user does not have the right to access the content of the requested lesson (i.e. is not enrolled in and is not the supervisor of the course containing the lesson)

5.4 GET /polyglot/get_lesson_data

Request sample:

- lessonId=3

Responses:

- 200 OK - lesson'd sata successfully found and returned
  - sample response content:
    body(JSON): {
    title: "lesson's title",
    courseId: 10,
    indexInsideCourse: 12,
    courseTitle: "lesson's course's title" }
- 401 UNAUTHORIZED - active user does not have the right to access the content of the requested lesson (i.e. is not enrolled in and is not the supervisor of the course containing the lesson)

5.5 GET /polyglot/get_word_question

Request sample:

- lessonId=3

Responses:

- 200 OK - word question successfully generated and returned
  - sample response content:
    body(JSON): {
    wordToLearnId: 15,
    lessonId: 18,
    word: "apple",
    currentPoints: 3,
    targetPoints: 10,
    sourceLanguage: "English",
    targetLanguage: "Romanian",
    foreignToNative:: true }
- 400 BAD REQUEST - there are no unknown words marked for this lesson, and thus no question could be generated
- 401 UNAUTHORIZED - active user does not have the right to access the content of the requested lesson (i.e. is not enrolled in and is not the supervisor of the course containing the lesson) OR the active user is not a student

5.6 POST /polyglot/answer_word_question

Request sample:

- body(JSON): {
  wordToLearnId: 15
  submittedTranslation: "mar"
  foreignToNative: true }

Responses:

- 200 OK - answer successfully evaluated
  - sample response content:
    body(JSON): {
    wordToLearnId: 15,
    lessonId: 18,
    word: "apple",

currentPoints: 4,
targetPoints: 10,
officialTranslation : "măr",
submittedTranslation: "mar",
accepted: true sourceLanguage: "English",
targetLanguage: "Romanian",
foreignToNative:: true }

- 401 UNAUTHORIZED - active user does not have the right to access the content of the requested lesson (i.e. is not enrolled in and is not the supervisor of the course containing the lesson) OR the active user is not a student

5.7 POST /polyglot/add_unknown_word

Request sample:

- body(JSON): {
word: "apple",
lessonId: 18 }

Responses:

- 200 OK - unknown word successfully saved
- 401 UNAUTHORIZED - active user does not have the right to access the content of the requested lesson (i.e. is not enrolled in and is not the supervisor of the course containing the lesson) OR the active user is not a student
- 403 FORBIDDEN - the given word was already saved as an unknown word, duplicates are not allowed.

5.8 GET /polyglot/get_lesson_vocabulary_in_pdf

Request sample:

- lessonId=15

Responses:

- 200 OK - the pdf containing the vocabulary of the lesson for the active user was successfully generated and returned
- 401 UNAUTHORIZED - active user does not have the right to access the content of the requested lesson (i.e. is not enrolled in and is not the supervisor of the course containing the lesson) OR the active user is not a student

.1 GET /polyglot/get_course_statistics

Request sample:

- courseId=15

Responses:

- 200 OK - the statistics for the course were successfully generated
  - sample response content:
    body(JSON): {
    nrEnrolledStudents: 15,
    lessonTitles: ["Lesson 1", "Lesson 2"],
    avgNrOfUnknownWordsPerLesson: [1, 2.3] }
- 401 UNAUTHORIZED - active user does not have the right to access the content of the requested course (i.e. is not enrolled in and is not the supervisor of the course) OR the active user is not a teacher

.2 GET /polyglot/get_lesson_statistics

Request sample:

- lessonId=15

Responses:

- 200 OK - the statistics for the lesson were successfully generated
  - sample response content:
    body(JSON): {
    title: "Lesson 1: Fruits",
    indexInsideCourse: 3,
    courseTitle: "English Course",
    unknownWordsToFrequency: {"apple": 2, "orange": 3, "strawberry": 12} }
- 401 UNAUTHORIZED - active user does not have the right to access the content of the requested lesson (i.e. is not enrolled in and is not the supervisor of the course containing the lesson) OR the active user is not a teacher