Virtual Teacher

Project Description

Virtual Teacher is an online platform for tutoring. Users will be able to register as either a teacher or a student.

Students must be able to enroll for video courses, watch the available video lectures. After finishing each video within a course, students will be asked to submit an assignment, which will be graded by the teacher. After a successful completion of a course, the students must be able to rate it.

Teachers should be able to create courses and upload video lectures with assignments. Each assignment will be graded individually after submission from the students. Users can become teachers only after approval from an administrator.

Functional Requirements

Entities

- Each **user** (teacher or student) must have a profile picture, first and last name, email, and a password.
    - The email will serve as their username so it must be a valid email and unique in the application.
    - The password must be at least 8 symbols and should contain capital letter, digit, and special symbol.
    - The first and last names must be between 2 and 20 characters long.
- **Courses** must have a title, a topic, a description, and a list of lectures. They should also have a starting date.
    - The title is a string between 5 and 50 characters long.
    - The topic is one of a predefined range like Science, History, Literature, and others.
    - The description is optional and can be at most 1000 symbols in length.
    - The starting date, if set, makes the course visible to the students but they cannot enroll before the said date.
- Each **lecture** must have, a title, a description, a video, and an assignment.
    - The title is a string between 5 and 50 characters long.
    - The description is optional and can be at most 1000 symbols in length.
    - The video must be embedded (for example from YouTube).
    - The assignment must be a text file, saved to the file system, or optionally, could be uploaded to a cloud file hosting service.
    - Search bar, allowing the student to search for additional information in Wikipedia.

Public Section

The public part must be visible without authentication.

Anonymous users must be able to register as students or teachers.

The anonymous users must also be able to browse the catalog of available courses. The user must be able to filter the catalog by course name, course topic, teacher, and rating, as well as to sort the catalog by name and/or rating.

Anonymous users must not be able to enroll for a course.

Private Part (Students)

Must be accessible only if the student is authenticated.

Users

Users must have private section (profile page).

Users must be able to:

- View and edit their profile (names and picture).
- Change their password.
- Enroll for courses.
- See their enrolled and completed courses.

Users could be able to:

- Take notes in plain text for each lecture.

Courses

- The videos in a course must be accessible only after enrollment.
- Students must submit their work as a text file (txt, doc, docx, etc.).
- Students must be able to see the grade they've received for the assignment and their average grade for the course (formed by the average of all assignments grades).
- After receiving a grade for their last assignment for the course, if their average grade is above the passing grade, defined on per course basis, they should be able to leave a rating for the course

Private Part (Teachers)

Teachers must have access to all the functionality that students do.

However, after they are approved by an administrator, a course administration page must be accessible to them. On it they must be able to modify courses and have a section where they can search for students.

Courses must be either drafts or published. Draft courses are visible to the teachers, but not to the students. Once the teacher has prepared the course, they must be able to publish it and it becomes available to all students.
Courses could have a comments page, where students can comment on the lectures and ask questions.

Administration Part

Administrators must have access to all the functionality that teachers do.

Administrators must not be registered though the ordinary registration process. They can be given administrator rights from the UI by other administrators only.

They must be able to modify/delete courses and users, approve administrators and teachers.

Optional features

Email Verification – In order for the registration to be completed, the user must verify their email by clicking on a link sent to their email by the application. Before verifying their email, users cannot make transactions.

Refer a Friend – A user can enter the email of people, not yet registered for the application, and invite them to register. The application sends to that email a registration link.

Attendance tracker – some courses can have required attendance. Each lecture from the course should have a single attendance tracker. In the tracker, the teacher marks the absent students. The teacher should only have access to the admin part of the sessions and the enrolled students should only see their attendance record.

Enrollment confirmation – the user can receive an email confirmation after enrolling in a course with all essential information regarding the course.

Graduation Certificate – After finishing a course, the user can download and/or receive an email with the certificate in PDF format.

Advanced courses – some courses may only be available if the student has completed other courses on the same topic. There will be different expert levels on a particular topic. For example, beginner, intermediate – available if the student has completed the beginner course; advanced – student has completed the corresponding intermediate courses. You are free to add more levels if you wish.

Student groups – each course should have groups with participants enrolled in the course. Students outside the course must not be able to be added to the groups. The group will enable the Teacher to give out group tasks. If one of the students uploads a solution to the task it counts as if everyone uploaded a solution. The teacher needs to grade only 1 upload per group. The grade is the same for all students of that group.

Course Rating – each course can be rated from 1 to 5 from the students. The comment on the rating is mandatory to the rating and it should be a minimum of 75 symbols.

Entrance Exam – each course can have an entrance exam minimum grade as an enrollment requirement. The teacher should be able to make it a quiz of questions. There should be the option of MCQ (multiple-choice-question), MSQ (multiple-select-question), or OEQ (open-ended-questions) - the OEQ should be graded by the teacher.

The Entrance Exam can have different durations – the teacher should be able to set start and end time. The exam should close automatically after the time expires.

Tutoring – the teacher should be able to show in their profiles if they are available for private lessons (tutoring sessions) and when.

Additional Resources – the teacher should be able to offer under each course additional resources which are accessible only after enrollment. The enrolled students should be able to request additional resources by leaving a request (min. 100 symbols). The teacher should be able to mark the request as resolved, not relevant, or work in progress. He can also assign the request to another student.

Assistant Teacher – a student can become an assistant teacher to a specific teacher. He can then edit all courses content of the teacher. The assistant teacher can create only drafts of new courses. Each course can then have three phases – draft, to be reviewed, and published. The teacher can review it and either publish it or return it in draft for corrections to be done. If the teacher returns it in the draft a note of a minimum of 75 symbols is required.

Course Lessons Availability – each lesson of a course can be made available after a specific date or after a specific action (submit of homework). The course can also be made available after minimum participants have enrolled – in this case the course should have an explanation and enrolled participants count.

Easter eggs – Creativity is always welcome and appreciated. Find a way to add something fun and/or interesting, maybe an Easter egg or two to you project to add some variety.

REST API

To provide other developers with your service, you need to develop a REST API. It should leverage HTTP as a transport protocol and clear text JSON for the request and response payloads.

A great API is nothing without a great documentation. The documentation holds the information that is required to successfully consume and integrate with an API. You must use Swagger to document yours.

The REST API provides the following capabilities:

1. Users
    - CRUD operations (must)
    - Enroll to courses (must)
    - Search by first, last name or email (must)
2. Courses
    - CRUD operations (must)
    - Rate (if implemented) (must)
3. Lectures
    - CRUD operations (must)
    - Submit assignment file (must)
    - Add/Edit user's notes (if implemented)

External Services

The **Virtual Teacher** web application will consume a public REST service to achieve one of its main functionalities.

MediaWiki Action API

The MediaWiki software powers Wikipedia. It helps you collect and organize knowledge and make it available to people. MediaWiki has several application programming interfaces (APIs). You will be using the MediaWiki Action API. It is free of charge. For usage details please see the Appendix.

Use Cases

Main use case

Evlogi is a student who is enrolled in a series of courses available on your Virtual Teacher application. Taking a course, he watches the videos in each lection. While watching, however, he wants to find additional information on a term he comes across in the video. Evlogi searches this term in the search bar available and receives as a response the titles and a short text snippet of a certain number (your decision) of wikipedia articles. He can now read a longer extract of each article or follow the link to the original article in Wikipedia.

Technical Requirements

General

- Follow OOP principles when coding
- Follow KISS, SOLID, DRY principles when coding
- Follow REST API design best practices when designing the REST API (see Appendix)
- Use tiered project structure (separate the application in layers)
- The service layer (i.e., "business" functionality) must have at least 80% unit test code coverage
- Follow BDD when writing unit tests
- You should implement proper exception handling and propagation
- Try to think ahead. When developing something, think – “How hard would it be to change/modify this later?”

Database

The data of the application must be stored in a relational database. You need to identify the core domain objects and model their relationships accordingly. Database structure should avoid data duplication and empty data (normalize your database).

Your repository must include two scripts – one to create the database and one to fill it with data.

Git

Commits in the GitHub repository should give a good overview of how the project was developed, which features were created first and the people who contributed. Contributions from all team members must be evident through the git commit history! The repository must contain the complete application source code and any scripts (database scripts, for example).

Provide a link to a GitHub repository with the following information in the README.md file:

- Project description

- Link to the Swagger documentation (must)
- Link to the hosted project (if hosted online)
- Instructions how to setup and run the project locally
- Images of the database relations (must)

Optional Requirements

Besides all requirements marked as should and could, here are some more *optional* requirements:

- Use branching while working with Git.
- Integrate your app with a Continuous Integration server (e.g., GitHub's own) and configure your unit tests to run on each commit to the master branch.
- Host your application's backend in a public hosting provider of your choice (e.g., AWS, Azure, Heroku).

Teamwork Guidelines

Please see the Teamwork Guidelines document.

Appendix

- Guidelines for designing good REST API
- Guidelines for URL encoding
- Always prefer constructor injection
- Git commits - an effective style guide
- How to Write a Git Commit Message

MediaWiki Action API

We will use this external service to implement search and retrieve information from Wikipedia within a Lection.

Note: We're using this service because it's free. It doesn't require any payment information to be used. In our case we will use only READ operations and there is neither registration, nor need of API Key.

The point is to get familiar with consuming a REST service, understanding its domain and do some data transformations.

Integration Guidelines

1. The goal is to have a list of Java object (numerous Wiki pages) with the following data:
   - their **title**,
   - a **content snippet**, and
   - the **full url** of the article

It is your decision how many wiki pages you would like to show on each search request

2. First understand the External API. Read the documentation of the external API to understand its endpoints, request parameters, and response format.

https://www.mediawiki.org/wiki/API:Main_page

3. You need an endpoint to receive the search value of your client (i.e. the student). Using this value, you need to build a query to send to Wikipedia. The query will look like this:

https://www.wikipedia.org/w/api.php?action=query&list=search&srsearch= ***Nelson%20Mandela****&utf8=&format=json&srlimit=1*

(Search API Official docs)

The response from Wikipedia will contain a lot of information you only need:

- the title with which you will retrieve the bigger snippet as Wikitext using the Revisions API

(Revisions API Official docs)

- Pageid with which you will retrieve the full url

(Full URL endpoint)

4. In a nutshell you will have to send three requests:

- First one to receive the title, pageid and short snippet of a search
- Second one to get bigger snipper using the title
- Third one to get the full URL using the above pageid

How is a request to external API made?

We use HttpClient that sends a request (OutputStream) and receives a response (InputStream)

- OutputStream with the URI we construct using a "moving" part - the query string

- InputStream will be mapped to JsonObject that we would like to receive - Array- or JsonNode depending on the properties of the JSON we receive from the External API
- Then we need to map the JsonObject to a Java Object with only some of its properties

Suggested Approach

- Proof of Concept for work with 3rd party REST APIs
- Test out how to store incoming data
- Carefully design your Database (no user access control at this point)
- Start Implementing the algorithm
- Design and create your user access control tables in the DB
- UI
- Optional Requirements

Legend

- Must – Implement these first.
- Should – if you have time left, try to implement these.
- Could – only if you are ready with everything else give these a go.