Lexicon LMS

The project you will work on during the final module is a learning platform, a so-called LMS1 *(Learning Management System),* adapted for Lexicon's extension training.
An LMS simplifies and centralizes communication between
teacher, university and student by collecting schedules, course materials, other information, exercises and submissions in one and the same place.

You must produce the system from scratch with database, back-end functionality and a well-thought-out front-end. This is called a "full-stack project" and aims to demonstrate your understanding of all parts of a web application and modern systems. The project intends to test the breadth of your understanding and that you have a foundation to stand on, regardless of future orientation within .NET.

Product description

The main task and goal of the system we will build is to make course materials and schedules easily available to students. It must also function as a collection point for submissions. For this to be possible, we also need flexible functionality for teachers to be able to easily administer these classes, students, schedules and documents. Because if it is not easy for the teacher to use the tool, the students will never have the chance to use it.

The finished system is intended above all to cover basic functionality, but in a well-thought-out and elaborate way. *Less is more* is often true when it comes to this one
type of applications to be used daily. Unfortunately, in order to reach as wide a market as possible, most LMS available today are enormously heavy and overloaded with all imaginable functionality that is rarely used - you must change this! Less is more does not necessarily have to refer to pure functionality, but rather about experience
complexity. There may be deep functionality, but the user should not need 14 options in every choice he makes.

Framework and techniques

The application must have been built in Blazor WebAssembly. The database must be built with Entity Framework Core according to *the code* first approach. Frontend must use Bootstrap 5.

1 Mer information: https://sv.wikipedia.org/wiki/Lärplattform

Entities, relationships and attributes (basic form)

The entities and attributes described below are a minimum, not an absolute description. Above all, the attributes will need to be expanded when you plan the system in more detail.

User

The application must manage users in the roles of students and teachers, these must all have logins and accounts in the application. The users should be saved with at least one name and one email address.

Course

All students belong to **a course and only one,** which in turn has a course name, a description and a start date. Example of course name: ".NET 2023".

Module

Each *course* reads one or more *modules,* these have module names, a description, start date and end date. Examples of modules: "Database design", "Javascript", etc.
Modules may not overlap or go outside the course.

Activities

The modules in turn have *activities,* these activities can be e-learning sessions, lectures, practice sessions, assignments or other. The activities have a type, a name, a start/end time and a description.
Activities must not overlap or go outside the module.

Document

All entities above can hold documents.

Examples of documents:

Submissions from the students, module documents, general information documents for the course, lecture materials or exercises linked to the activities.

The *document entity(ies)* should have a name, a description, a timestamp of the upload occasion, and information about which user uploaded the file.

Use cases (minimum requirement)

These use cases are not comprehensive; depending on the implementation, more detailed cases must be developed to cover all practical functionality.

A non-logged-in visitor must be able to: • Log in

A student must be able to:

- See which course he is studying and who the other course participants are • See which modules he is studying
- See the activities for a specific module (module schedule).

A teacher must be able to:

- See all courses
- See all modules included in a course • See all activities a module contains
- Create and edit users (teachers and students) • Create and edit courses • Create and edit modules • Create and edit activities

Use cases (desirable)

A student must be able to:

- See if a specific module or activity has any documents linked to it and i in that case download these.
- See which submission information he has received, if it has already been submitted, when must be submitted at the latest and if it is late.
- Upload files as submissions

A teacher must be able to:

- Upload documents for courses/modules/activities • Receive submissions

Use cases (extra if there is time)

A non-logged-in visitor must be able to:

- Request a new password

A student must be able to:

- Share documents with your course or module
- Receive notifications when a teacher has uploaded new documents for the course, e.g. one new document
- Receive feedback on submissions
- Receive notifications when a teacher has provided feedback on an assignment
- Register yourself after receiving an invitation via email
- Must be able to remove oneself from the system and delete all information about oneself that exists saved according to GDPR

A teacher must be able to:

- Provide feedback on submissions

API

To make the system more flexible, we put all data access in an API. This means that we can easily reuse that implementation if we need to create other types of applications such as a mobile app or replace our frontend with, for example, React.

The application retrieves its data from the API via HttpClient.

Frontend – Blazor

“Blazor WebAssembly Standalone”-templaten med “ASP.NET Core Hosted”-
option selected. Then we get an API template in the bargain. We also add authentication in the form of "Individual Accounts".

The desire is that the frontend visually has a uniform appearance. The interface must be built with Bootstrap. Look at the components that are ready!

In addition to these purely aesthetic wishes, the remaining frontend focus must be directed towards the user experience and reducing the user's cognitive friction.

The system must be easy to use. And it must be clear how it works.
The application must be responsive. Bonus points for a well-functioning mobile version.

Working

Scrum

The project must be carried out in a group with a scrum-based working method. We will work with two sprints. A new sprint begins with *sprint planning* where you set up a sprint backlog,
distributes the work and updates your *task board.*

Each day begins with a *standup (daily scrum)* where you deal briefly, one by one

1. What have you done since the last *standup,*
2. What you plan to do until the next and
3. If there is something blocking planned work.

You hold the meeting in the Teams app in the group's channel. You must have your task board in front of you so you can visually see how you are doing.

When the sprint is finished, you end it with a sprint demo in front of the other groups followed by
a retrospective.

Git - Versionshantering

The project must be version managed with git and GitHub.
You must at least each have a personal branch as well as a master and a development. We recommend that in this project you start working with feature branches instead of a personal branch.

Test (desirable)

Feel free to try writing tests. Not a requirement but can be incredibly educational.
Plus for writing the tests first according to TDD.

Reconciliation points and deliverables

During the course of the project, you are expected to report certain points before continuing. This because avoid dead ends and maximize your effective development time.

- *Product backlog* must be approved before you start implementation.
- ER-Diagram must be approved before you start an implementation.
- Wireframes for some key views must be approved before you start one implementation.
- *Sprint backlog* must be approved before you start a new sprint.
- At the end of the sprint, all delivery-ready changes must be demonstrated at the sprint demo.

Planning

During the start-up of the project, you must start working on planning the work. You must produce documentation according to the points in reconciliation points and deliverables.

Think about how the system should work for a teacher. What is the primary thing a teacher is interested in? How is a good flow created to create new courses with everything such a course consists of?

What are you as students most interested in when you log in?
What information is most relevant to access directly?
Here it is only about how you present the data that is stored in the system.
How will you navigate to all functionality, etc.

Accounting

Detailed information about the accounting phase will come later

Good luck!