Table of Contents

README

Features:

-Easy registration for students and teachers.

-Easy process of making an exam for teachers

-Fast and reliable way of taking exams for students

-Great „social-media” feel including users profile with photos and messages system

Prerequsities:

To install and run this project you will need:

- .Net Core SDK version >=2.1.0

- NodeJS

- Angular CLI

Installation:

First download or clone repository.

Then to launch a server, inside the CourseApp-API folder run from terminal: dotnet run / dotnet watch run

Next to launch an angular app, inside CoureseApp-SPA folder run from terminal: ng serve

Both parties will restore dependencies and install required packages so be patient.

After that open browser at http://localhost:4200/

Contribiute:

Issue tracker: https://github.com/KarolGrzesiak/CourseApp/issues

Source Code: https://github.com/KarolGrzesiak/CourseApp

Support:

If you are having issues, please let us know.

We have mailing list located at: karol.grzesiakk@gmail.com

License:

This project is licensed under the BSD license.

GENERAL INFO

CourseApplication (CAP) is a social-media type web application which allows for easy and fast way to make or take an exam.

Authorization and authentication is based on roles and JWT tokens.

At the beginning user can only register as all functionalities of website are only for the logged in users. After making an account users are automatically logged in.

Then the main navigation of the application is navbar located at the top of the presented view. It includes from the left: **Users**, **Tests**, **Messages**, **Games**, **Admin** (only for admin-user) and finally photo with a welcome message.

Users – it takes user to a view where all users are presented – presentation comes with some filters such as presenting only students or teachers. This is the view from where user can navigate to other user's profiles to look at their informations or to chat with them.
Tests – it takes user to a view where all exams are stored. Here student can enroll on test after getting a password from the author of exam, here user can take an exam or look at his/her finished exams and scores which he/she got. Teachers have one extra tabset with exams created by them where they can see statistics about students who took those exams. Also they have a little float button at the bottom of the view which allows them to create or delete tests.
Messages – it takes user to a view with inbox and outbox of messages. From here user can see all his/her messages which he/she send or received, on top of that every message is a navigation to the whole conversation. There is also an option to delete a message here but for pernament deletion both parties have to remove it.
Games – it opens a selector from which user can choose one of two games – „Snake" or „Rock,Paper,Scissors". Both links take user to respective game where user can relax. „Snake" is more complex game with highscore and three modes, whereas „Rock,Paper,Scissors" is more about checking student's luck, for example before taking an exam.
Admin – only available for the admin-user. It takes admin to a list of users where he/she can change roles – mainly used to give a new teacher permissions.
Photo with welcome message – it shows current main photo of the user and also provides logout functionality or navigation to user's profile where he/she can make some changes for example – upload new photo.

Additionaly – every user has their profile page with all theirs informations such as interests, their role, photos and tabset of messages where the conversations are being held. To navigate to this page user have to navigate first to Users from navbar and then click on the icon on someone's card. To edit this page user have to navigate from welcome message by selecting Edit Profile option.

BACKEND

Configuration:

In the appsettings.json:

Change/Add CloudinarySettings to store users photos.
Change ConntectionString to your database.
Change TokenSecret.

Helpers:

AutoMapperProfile – creates map between actual data models and DTO's.
CloudinarySettings – enables simpler usage of cloudinary settings – injected at Startup.cs
Extensions – some helpers methods (Adding Application Error to Http response, Calculating age, Adding Pagination to Http response, Creating and Veryfing Password Hash for exams)
LogUserActivity – Action filter which helps with controlling activity of users
MessageParams – information about pagination
PagedList – extension of List, adding pagination
PaginationHeader – object for http response
Seed – method invoked when database is empty – seeding users from UserSeedData.json
UserParams – information about pagination
UserSeedData.json

Seeding:

When backend is being deployed (by dotnet run/dotnet watch run from console), SeedUsersAndRoles() from Seed.cs from Helpers folder is being run which seeds users.

Models:

Answer – represents answer to question, have information about being correct and actual question navigation.
Exam – represents exam, have all the information about an exam with navigations to questions and author
Message – represents message between two users, have navigation to both sender and recipient and some additional information
Photo – represents user's photo, have navigation to user and information about being the main photo.
Question – represents question in exam, have navigations to answers and to exam.
Role – represents role of the users, linked with UserRole
User – represents application user with a lot of information and navigations to photos, userroles, messages and Exams.
UserAnswer – represents user's answer to question, have navigation to both author and question.
UserExam – represents users which are taking an exam, have navigation to both user and exam.
UserRole – links user with role.

Repositories:

IRepositoryBase – basic repository, requires Add,Delete and Update methods, every repository extends it
IRepositoryWrapper – unit of work type of repository, contains all others repositories and SaveAllAsync() method to save all work of repositiories.
IUserRepository() - requires GetUserAsync(int userId) – gets user from database by his id, GetUsersAsync(UserParams userParams) - gets paginated users from database.
IUserExamRepository – requires GetEnrolledExamsForUserAsync(int userId) – gets exams on which user enrolled, GetFinishedExamsForUserAsync(int userId) – gets exams which user finished, GetUserWithExamAsync(int userId, int examId) -gets UserExam which includes those parameters. GetUsersFromExamAsync(int examId) – gets all users from exam.
IUserAnswerRepository – requires GetUserAnswersAsync(int examId, int userId) – get users answers.
IQuestionRepository – requires GetQuestionAsync(int questionId) – gets question, GetQuestionsAsync(int examId) – gets questions
IPhotoRepository – requires GetPhotoAsync(int id) – gets photo, GetMainPhotoForUserAsync(int userId) – gets main photo for user
IMessageRepository – requires GetMessageAsync(int messageId) – gets message, GetMessagesForUserAsync(MessageParams messageParams) – gets messages for user, GetMessageThreadAsync(int userId, int recipientId) – gets whole conversation.
IExamRepository – requires GetNotEnrolledExamsForUserAsync(int userId) – gets exams for user on which he didn’t enrolled yet, GetExamAsync(int examId) – gets exam, GetCreatedExamsForUserAsync(int userId) – gets exams created by the user
IAnswerRepository – requires GetAnswerAsync(int answerId) – gets answer, getAnswersAsync(int questionId) – gets answers

Constants:

TeacherRole – represents teacher role
StudentRole – represents student role
AdminRole – represents admin role
Password – represents password used in seeded data
DaysToAddToTokenExpirationDate – represents number of days for which user token will be valid.
MinAge – represents minimum age of user
MaxAge – represents maximum age of user
MaxPageSize – represents maximum of data sent at once
PageSize – represents default number of data sent at once
RequireAdminRole, Require TeacherRole – represents strings used in policy authorization

FRONTEND

Services:

Admin – provides methods for getting users with their roles and updating their roles
Alertify – implements 3rd party library Alertify and provides nice looking pop-up informations
Auth - provides methods involving logging and registration and also stores information about current logged in user.
Exam – provides methods for creating,getting,deleting exams, answers and questions
Score – provides methods for retrieving and storing best score for the games
User – provides methods for showing,updating users and showing,deleting and creating messages and photos of users.

Components:

Admin components – involves roles modal and user management – displays information for admin in a table with tabset.
Exams components – involves table of 4 tabsets – displays information about available exams, enrolled exams, created exams and finished exams. Also involves panel for teacher to create and delete exams.
Games components – involves two games snake and rock-paper-scissors
Home components - involves welcome page for new users and registration component which displays information about registration
Members components – involves components of users – displays list of users on cards and each specific user profile
Messages component – displays inbox and outbox of messages.
Nav component – involves navigations for every component – displays navbar at the top of application with all navigations
App component – main component, involves every other component.

Resolvers:

Exam-list – gets lists of exams for exam-list component
Exam-take – gets lists of questions with answers for exam-take component
Member-detail – gets information about user for member-detail component
Member-edit – gets information about logged in user for member-edit component
Member-list – gets lists of users for member-list component
Messages – gets messages for inbox/outbox at messagas component.

Guards:

Auth – provides protection from not logged in users
Prevent-unsaved-changes – provides protection when user didn’t save changes which he made

Directives:

HasRole – checks if user is in specified role

Routes:

/ – starting page
/members – page with list of users
/**members/:id –** user's profile page
/member/edit – page for editing profile
/exams/ - page with exams
/exams/create – page to create an exam
/exams/:examId/questions/create – page to create questions
/exams/:examId/take – page to take an exam
/messages – page with outbox/inbox
/games/snake – page with „Snake" game
/games/rockpaperscissors – page with „Rock,Paper,Scissors" game
/admin – admin's page

Environments:

apiUrl = htttp://localhost:5000/api/

API:

Users: api/users

GET – returns list of users
GET /:id – returns user with specified id
PUT /:id – updates user with specified id

Stats: api/exams/stats

GET /:examId – returns scores information for user
GET /teacher/:examId – returns stats about students who took specified exam

Questions: api/exams/:examId/questions

GET /:questionId – returns specified question
GET – returns list of questions
POST – creates a question
DELETE /:questionId – deletes a question

Photos: api/users/:userId/photos

GET /:id – returns photo
POST – creates a photo
POST /:id/setMain – sets specified photo as Main
DELETE /:id – deletes a photo

Messages: api/users/:userId/messages

GET /:id – returns a message
GET – returns a list of messages
GET /thread/:recipientId – returns whole conversation
POST – creates a message
POST /:id – deletes a message
POST /:id/read – marks message as read

Exams: api/exams

GET /:examId – returns an exam
GET – returns not enrolled on list of exams for user
GET /enrolled – returns enrolled on list of exams for user
GET /created – returns list of created exams by user
GET /finished – returns list of finished exams by user
POST – creates an exam
POST /:examId/enroll/:userId – enrolls user on exam
DELETE /:examId – deletes exam

GET /:examId/:userId/answers – gets user answers
POST /:examId/take – saves user answers

Auth: api/auth

POST /register – creates a user
POST /login – generates JWT token and returns user with token

Answers: api/auth

GET /:answerId – returns an answer
GET – returns a list of answers
POST – creates an answer
POST /create – creates an list of answers

Admin: api/admin

GET /usersWithRoles – returns list of users with their roles
POST /editRoles/:userName – edits roles of user

SECURITY

Authorization:

PolicyBased – this model includes a policy composed by one or more requirements. A requirement is a collection of data parameters used by the policy to evaluate the user Identity. A handler responsible of evaluating the properties of requirements to determine if the user is authorized.
In this application every user has a role and based on that server evaluates Identity.

More here:
https://docs.microsoft.com/vi-vn/aspnet/core/security/authorization/policies?view=aspnetcore-2.0

Authentication:

JWT Authentication. Every user after logging in gets a JWT token and based on that server checks requests.

More here:
https://jwt.io/introduction/