

---

---

# Courses/Labs Management System

 <https://github.com/StrugariStefan/CLMS> 

---

---

# Summary

1. Team members
2. Project requirements
3. Technologies involved
4. Development workflow
5. Architecture
6. Microservices:
  - Authentication
  - Users
  - Courses
  - Gamification
  - Notifications

## Team members

- Birsan Ioana (B5)
- Ciulei Andrada-Teodora (B6)
- Loghin Alexandru (B4)
- Silistru Alexandru (A1)
- Strugari Mihai Stefan (B4)

## Team of 5 logo



# Project requirements

Creating a platform for the management of users, courses and laboratories that will allow:

- the registration of users who can be teachers or students
- sending notifications
- assessment of students
- uploading and downloading of courses/labs materials
- the existence of interaction between students and teachers, which consists in asking questions, providing feedback and giving answers
- authentication mechanism

# Technologies involved

**C#**

**.Net Core 2.2**

**OpenAPI Specification**

**Sql Server**

**Microsoft Azure Blob Storage**

**MSTest.TestFramework 1.3.2**

# Development workflow - Github

1. Checkout to master and pull the latest version from github locally before adding any other code. This should be done each time you want to make changes starting from the latest master changes:

- `git branch` -> current branch
- `git checkout branch-name` -> move to branch branch-name

2. Checkout to a new branch with a name that describes the feature, following the standard **name/feature**:

```
git checkout -b andrei/add-animations
```

3. Make your desired changes to the code and make commits with those changes:

```
git commit -m "message that describes commit" .
```

4. Push the changes to GitHub to a branch with the same name:

```
git push -u origin local-branch-name:desired-branch-name-on-github
```

5. Validate that the changes work as expected.
6. If your changes work as expected merge the new branch to the master branch. Merge only if the changes work as expected.

# Development workflow - Trello

## Your Team Boards

Authentication Management

Content Management

Gamification Management

Notifications Management

Users Management

Create new board...

## Users Management



CLMS Free

Team Visible

IB

AS

AC

MS

4



### OTHERS

DDD



1

Follow this example



1

netcorekit



OPEN API



Creating a simple data-driven  
CRUD microservice



Nexmo Developer | Send an SMS  
with the Messages API



2

### TO DO

Test WriteUserRepository

AS

Test ReadUserRepository

AS

Test UserValidator if necessary

AS

+ Add another card

### IN PROGRESS

Test endpoints

MS

+ Add another card

### DONE

Refactor: consult the details for  
more info



IB

Add OPEN API config



IB

Implement TeachersController



IB

Finish documentation for endpoints



IB

Create UserValidator class and  
implement



IB

Create project structure



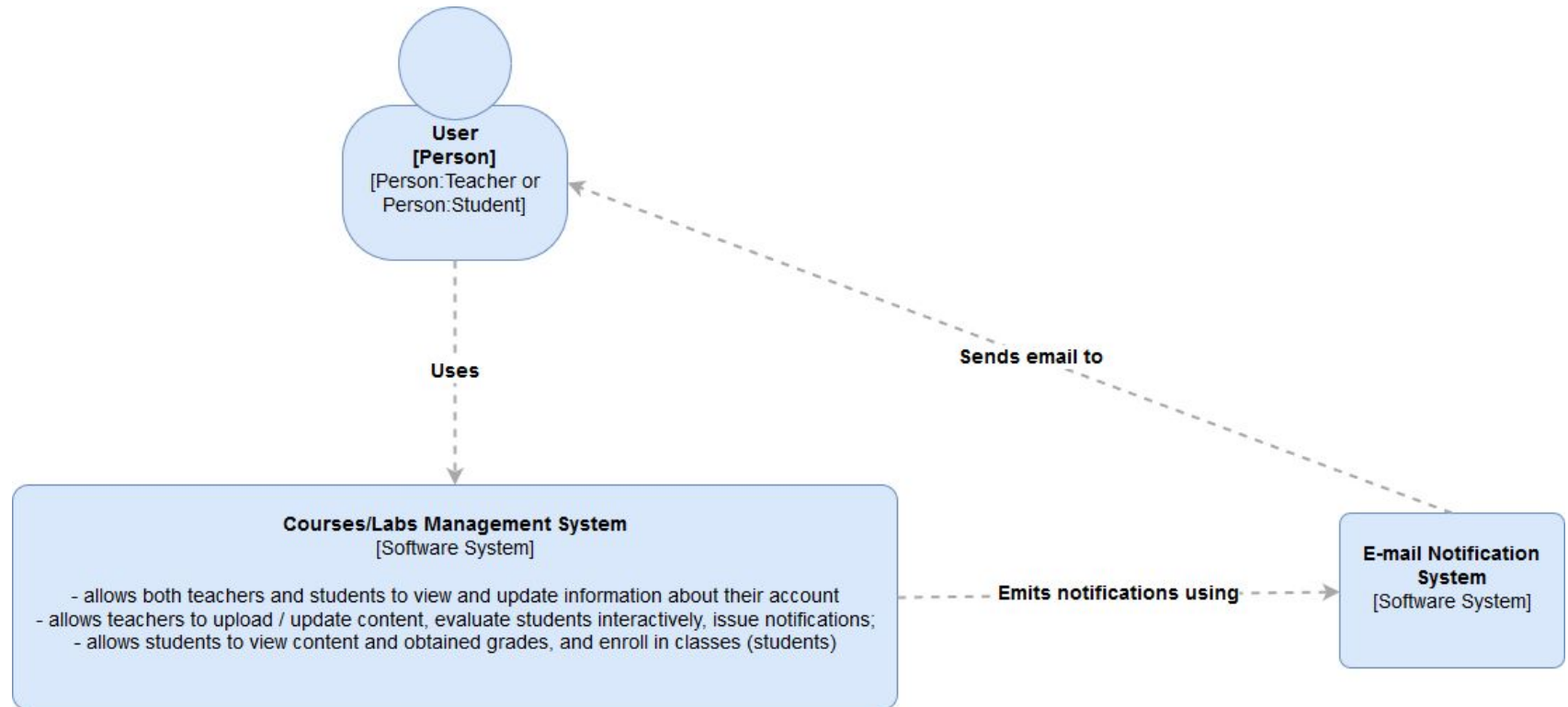
Update README with information



Implement WriteStudentRepository

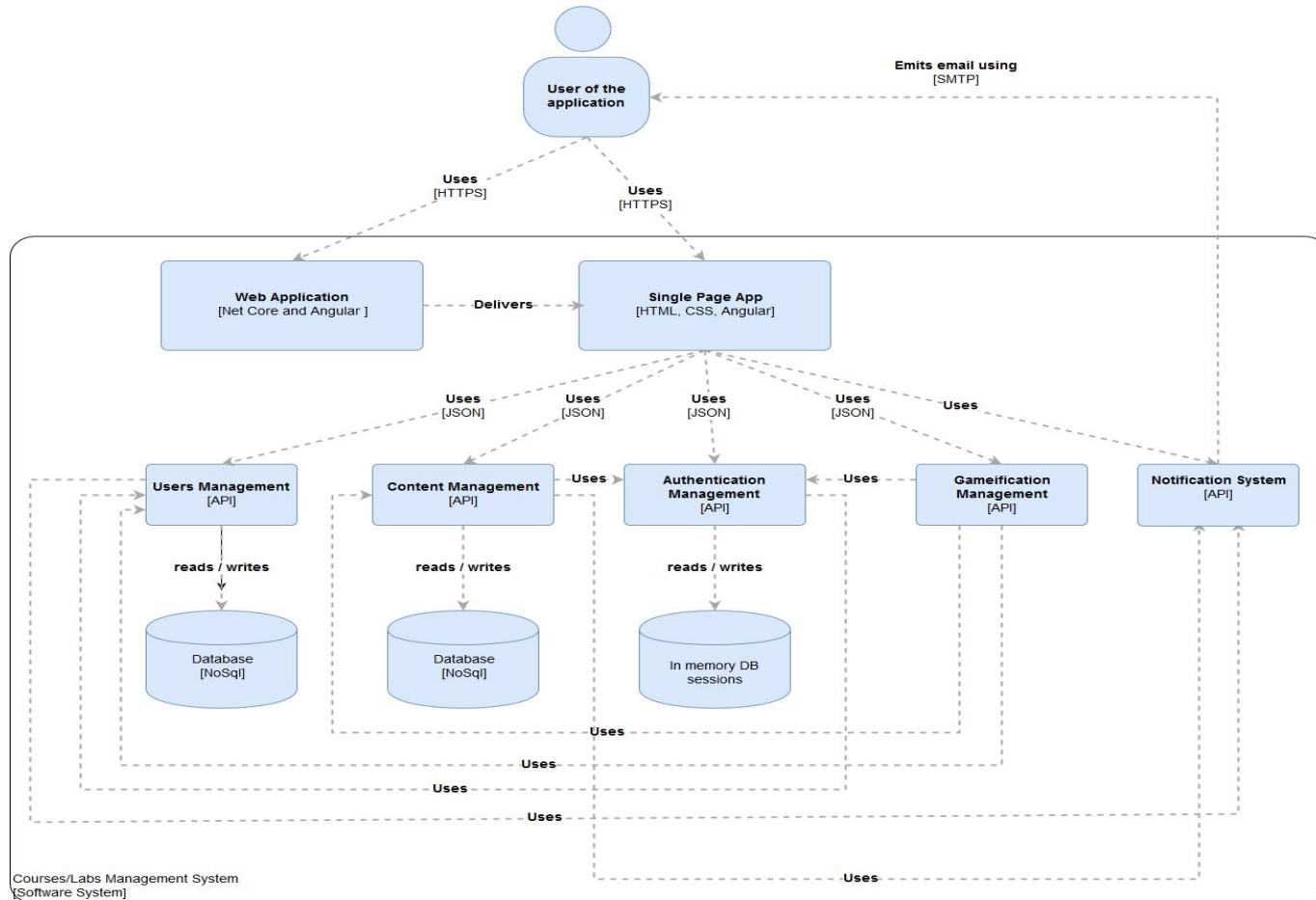
AS

# Architecture - level 1





# Architecture - level 2



# Microservices - Auth.API

## Auth

**GET** `/api/v1/auth/loggedIn/{token}` Checks if a user is logged in.

**POST** `/api/v1/auth/login` Logins a user.

**POST** `/api/v1/auth/logout` Disconnects a user.

**GET** `/api/v1/users/role/{role}` Obtains all users by role.

### Parameters

#### Name

#### Description

**role** \* required  
**integer** (\$int32)  
(path)

**AuthToken**  
**string**  
(header)

Execute

# Microservices - Users.API

## Users

**GET** `/api/v1/users/{id}` Obtains an user by id.

**DELETE** `/api/v1/users/{id}` Deletes an user by id.

**GET** `/api/v1/users/role/{role}` Obtains all users by role.

**GET** `/api/v1/users` Returns all users.

**POST** `/api/v1/users` Registers an user. Roles: 1 = student, 2 = teacher.

**POST** `/api/v1/users/registered` Checks if a user is registered.

# Microservices - Courses.API

## Courses

**GET** `/api/v1/courses` Return all courses.

**POST** `/api/v1/courses` Creates a new course.

**GET** `/api/v1/courses/{id}` Obtains course by id.

**DELETE** `/api/v1/courses/{id}` Deletes a specific Course.

**GET** `/api/v1/courses/{name}` Obtains course by name.

## ResourceFiles

**GET** `/api/v1/resourceFiles` Return all resourceFiles.

**POST** `/api/v1/resourceFiles` Creates a new resourceFile.

**GET** `/api/v1/resourceFiles/{id}` Obtains resourceFile by id.

**DELETE** `/api/v1/resourceFiles/{id}` Deletes a specific resourceFile.

**GET** `/api/v1/resourceFiles/download/{id}` Downloads a specified resource file

# Microservices - Gamification.API

## Answers

**GET****/api/v1/answers** Return all answers.**POST****/api/v1/answers** Creates a new answer.**GET****/api/v1/answers/{id}** Obtains answer by id.**DELETE****/api/v1/answers/{id}** Deletes a specific Answer.

# Microservices - Gamification.API

## Questions



**GET** /api/v1/questions Return all questions.

**POST** /api/v1/questions Creates a new question.

**GET** /api/v1/questions/{id} Obtains question by id.

**DELETE** /api/v1/questions/{id} Deletes a specific Question.

**GET** /api/v1/questions/type/{type} Obtains questions by type.

**GET** /api/v1/questions/levelOfInterest/{levelOfInterest} Obtains questions by level of interest.

# Microservices - Gamification.API

## Scores



**GET** /api/v1/scores Return all scores.

**GET** /api/v1/scores/user/{userId} Return all scores by userId.

**GET** /api/v1/scores/course/{courseId} Return all scores by courseId.

**GET** /api/v1/scores/user/{userId}/course/{courseId} Obtains score by userId and courseId.

**DELETE** /api/v1/scores/user/{userId}/course/{courseId} Deletes a specific Score.

**PUT** /api/v1/scores/user/{userId}/course/{courseId}/addpoints Adds points to score, creates new score if it doesn't exist.

## Microservices - Notifications.API

The functionality displayed by this service consists in sending emails and sms.  
As far as implementation is concerned, we have used the services outlined by:

[Mailgun API](#)

[Twilio Programmable SMS](#)

### Notifications

**POST**

`/api/v1/notifications/email` Sends an e-mail.

**POST**

`/api/v1/notifications/sms` Sends an sms.



# CLMS application characteristics

- **Flexibility and extensibility**

CLMS application can be easily modified, continued and improved

- **Modularity and scalability**

The microservices architecture gives us flexibility in terms of how we can scale the whole application.

- **Easy to use and understand by the user**