Easy Learn Community
MongoDB database project

Kornel Stefańczyk

January 13, 2021
version: 0.2

Contents

1 Introduction

Easy Learn Community is a web application that connects tutors and students and allows the user to give and receive tutoring while staying at home.

The application will offer a database of registered tutors together with basic information about them. People interested in receiving tutoring will be able to browse the offers without prior logging in, as well as contact a potential tutor. The conversation history will be recorded in the application after the interested party has registered (student profile). The application will include the necessary tools for online tutoring.

1.1 Scope, Approach and Methods

Scope of the project includes:

- Create definition of the database containing all fields needed to run the application.
- Create basement of the application working without implemented video and chat communication service.

Application will be based on the Ryan's Chenkie react app named "ReactSecurity - Orbit"(MIT licence).

The database will be developed in MongoDB technology that is NoSQL database technology often used in webapps.

1.2 System Overview

This section provides an overview of the entire web application. The application will be explained and discussed in the context of its application.

The application assumes the main system functionalities:

- Announcement of tutors by providing their contact details visible to all system users
- Tutoring online through a dedicated online chat application
- Communication with any of the tutors via chat
- Visible online availability of tutors and possible immediate tutoring by available tutors
- The sorting of the tutors on the lists is closely linked to the ranking of tutors, which is determined by the opinion

2 Defining the data structure

Definition of data structure and Rest points was done in OpenAPI Specification - Version 3.0.0. Full file is in the Appendix A.

Basic data model of *UserBasicData* can be seen at the listing below.

```
565     UserBasicData:
566       type: object
567       properties:
568         _id:
569           type: string
570         firstName:
571           type: string
572           example: "John"
573         lastName:
574           type: string
575           example: "Rambo"
576         email:
577           type: string
578           example: "test@test.com"
579         phone:
580           type: string
581           example: "+48555666333"
582         dateOfBirth:
583           type: string
584           format: date-time
585           example: "2000-12-02T00:00:00.000Z"
586         educationLevel:
587           $ref: "#/components/schemas/EducationLevelValues"
588         gender:
589           type: string
```

Basic data model are often part of the more complex structure like definition of *User* structure.

```
559     User:
560       allOf:
561         - $ref: "#/components/schemas/UserBasicData"
562         - $ref: "#/components/schemas/UserStatusData"
563         - $ref: "#/components/schemas/UserGetPut"
564         - $ref: "#/components/schemas/UserPost"
```

Some of structures can be used to modify data in the database or get data from the database. Manipulating database was done by RestAPI. Example of usage data structures is described in GET, PUT and DELETE requests. GET is used to get data from the database, PUT is used to update data existing in database and DELETE is used to remove data object from database.

```
76    /api/v1/users:
77      get:
78        tags:
```

```
79          - users
80        description: Get data of current logged in user
81        summary: Get current logged in user
82        operationId: getLoggedInUser
83        responses:
84          "200":
85            description: successful operation
86            content:
87              application/json:
88                schema:
89                  $ref: "#/components/schemas/UserGetPut"
90      put:
91        tags:
92          - users
93        summary: Updated current logged in user
94        description: This can only be done by the logged in user.
95        operationId: updateLoggedInUser
96        responses:
97          "200":
98            description: successful operation
99          "404":
100           description: User not found
101       requestBody:
102         content:
103           application/json:
104             schema:
105               $ref: "#/components/schemas/UserGetPut"
106         description: Updated user object
107         required: true
108     delete:
109       tags:
110         - users
111       summary: Delete current logged in user
112       description: This can only be done by the logged in user.
113       operationId: deleteLoggedInUser
114       responses:
115         "401":
116           description: Unauthorized
117         "404":
118           description: User not found
```

POST method is used to creating not existing data object in database. Rest point */api/v1/signup* allows new user to create new account.

```
16    /api/v1/signup:
17      post:
18        tags:
19          - users
20        summary: Create user
21        description: Sign up user. Create new user account.
22        operationId: createUser
23        responses:
24          "200":
25            description: successful operation
26          "400":
27            description: "Bad request"
28        requestBody:
29          content:
```

```
30            application/json:
31              schema:
32                $ref: "#/components/schemas/UserPost"
33          description: Created user object
34          required: true
```

Security and authentication of the rest points was done with usage of the bearerAuth. The bearer token is a cryptic string, generated by the server in response to a login request. The client must send this token in the Authorization header when making requests to protected resources. Cryptic string contains user data. They are used to setup frontend according to user constraints and preferences. If string containing user data would be changed then bearer token would change and there would be not possible to authenticate these token by server.

```
35    /api/v1/users/login:
36      post:
37        tags:
38          - users
39        summary: Logs user into the system
40        operationId: loginUser
41        responses:
42          "200":
43            description: successful operation
44            headers:
45              X-Rate-Limit:
46                description: calls per hour allowed by the user
47                schema:
48                  type: integer
49                  format: int32
50              X-Expires-After:
51                description: date in UTC when token expires
52                schema:
53                  type: string
54                  format: date-time
55            content:
56              application/json:
57                schema:
58                  $ref: "#/components/schemas/UserGetPut"
59          "400":
60            description: Invalid email/password supplied
61        requestBody:
62          content:
63            application/json:
64              schema:
65                type: object
66                properties:
67                  email:
68                    description: The email for login
69                    type: string
70                    format: email
71                    example: "test@test.com"
72                  password:
73                    description: The password for login in clear text
74                    type: string
75                    example: "test"
```

3 Server Implementation

Server was based on NodeJS and was written in JavaScript. It contains implementation of the OpenAPI definition of RestAPI.

Database was set up in external online database server and tools *MongoDB Atlas*. *MongoDB Compass* tool was used to view and modify dataset stored in MongoDB database. Screenshot of the tool is visible at the figure nr 1.

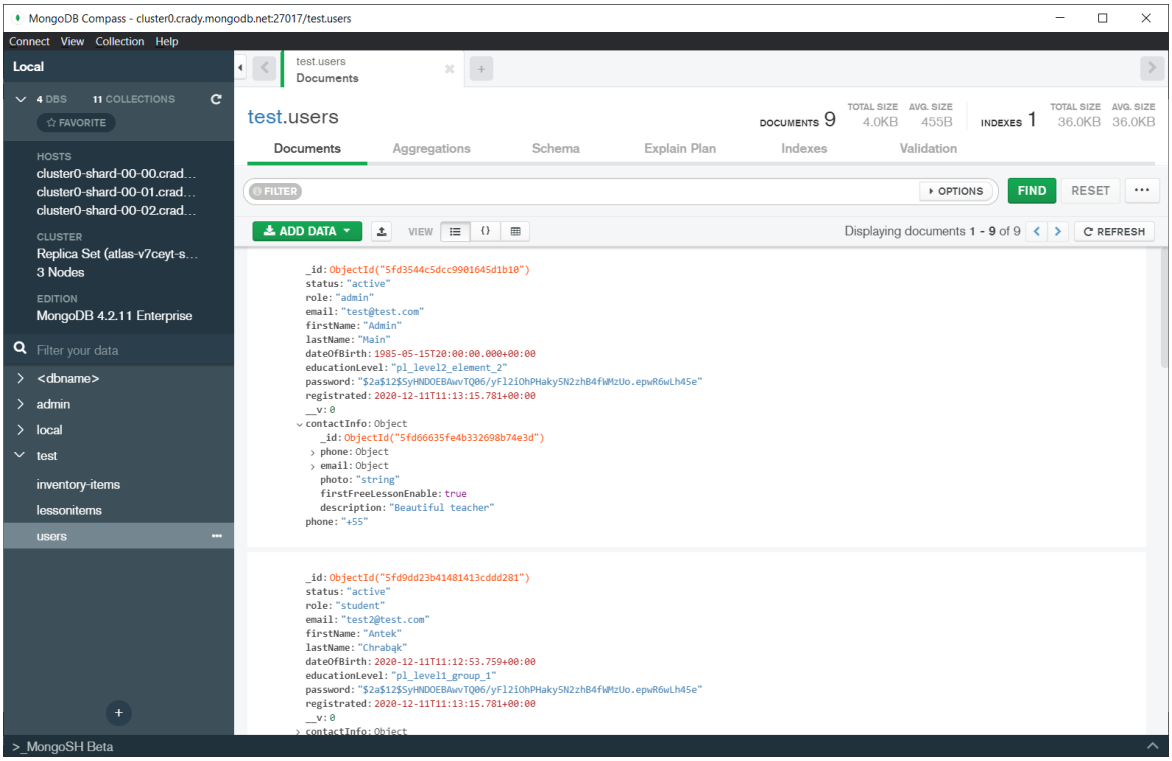

Figure 1: MongoDB Compass

To use connect to the MongoDB database import of the mongoose liblary was needed.

```
1 const mongoose = require('mongoose');
```

To manipulate data in the database mongoose Schema of the collections should be defined. My project contains two collections, so both are defined as separate mongoose Schemas.

Definition of LessonItem object:

```
1  const mongoose = require('mongoose');
2  const Schema = mongoose.Schema;
3
4  const lessonFeedbackModel = new Schema({
5      feedbackPostDate: { type: "date", required: true },
6      feedbackEditDate: { type: "date", required: true },
7      description: { type: String, required: true},
8      rating: { type: Number, required: true },
9      studentId: { type: mongoose.Types.ObjectId, required: true },
10     valuableFeedback: { type: [Schema.Types.ObjectId], required: false },
11     notValuableFeedback: { type: [Schema.Types.ObjectId], required: false
           }
12 });
13
14 const lessonItemModel = new Schema({
```

```
15      teacherId: { type: Schema.Types.ObjectId, required: true },
16      educationLevel: { type: [String], required: true },
17      places: { type: [String], required: true },
18      teachingLanguages: { type: [String], required: true },
19      hourlyRate: { type: Number, required: true },
20      onlineModeEnable: { type: Boolean, required: true },
21      faceToFaceModeStudentPlaceEnable: { type: Boolean, required: true },
22      faceToFaceModeTeacherPlaceEnable: { type: Boolean, required: true },
23      description: { type: String, required: false},
24      fieldOfStudy: { type: String, required: true},
25      lessonFeedback: { type: [lessonFeedbackModel], required: false },
26      numberOfFeedbacks: { type: Number, required: false },
27      rating: { type: Number, required: false },
28      phoneNumber: { type: String, required: false},
29      email: { type: String, required: false},
30      firstFreeLessonEnable: { type: Boolean, required: false}
31  });
32
33  module.exports = mongoose.model('lessonItem', lessonItemModel);
```

Definition of User object:

```
1   const mongoose = require('mongoose');
2   const Schema = mongoose.Schema;
3
4   const userContactDataModel = new Schema({
5     phone: new Schema(
6       {number: {type: String, required: false},
7       visible: {type: Boolean, required: false}
8     }),
9     email: new Schema(
10      {email: {type: String, required: false},
11      visible: {type: Boolean, required: false}
12    }),
13    photo: { type: String, required: false },
14    firstFreeLessonEnable: {type: Boolean, required: false},
15    description: { type: String, required: false }
16  });
17
18
19  const userModel = new Schema({
20    firstName: { type: String, required: true },
21    lastName: { type: String, required: true },
22    email: { type: String, required: true },
23    phone: { type: String, required: false },
24    password: { type: String, required: true },
25    dateOfBirth: { type: "date", required: true },
26    educationLevel: { type: String, required: true },
27    gender: { type: String, required: false },
28    registered: { type: "date", required: true },
29    status: { type: String, required: true, default: 'active' },
30    contactInfo: { type: userContactDataModel, required: false },
31
32    role: { type: String, required: true, default: 'user' },
33    bio: { type: String, required: false }
34  });
35
36  module.exports = mongoose.model('user', userModel);
```

In the next step Mongoose Schemas can be imported to server source code:

```
const User = require('./data/User');
const LessonItem = require('./data/LessonItem');
```

One of the most important operations is POST request, that create object in database. Below listing consist operation of adding new lesson item to system.

```
app.post('/api/v1/lessons', requireAuth, async (req, res) => {
  // operationId: addLessonItem
  errorMessage = 'There was a problem creating your lesson item';
  try {
    const { teacherId,
      educationLevel,
      places,
      teachingLanguages,
      hourlyRate,
      onlineModeEnable,
      faceToFaceModeStudentPlaceEnable,
      faceToFaceModeTeacherPlaceEnable,
      description,
      fieldOfStudy
    } = req.body;

    const lessonsData = {
      teacherId,
      educationLevel,
      places,
      teachingLanguages,
      hourlyRate,
      onlineModeEnable,
      faceToFaceModeStudentPlaceEnable,
      faceToFaceModeTeacherPlaceEnable,
      description,
      fieldOfStudy
    };
    if (req.user.role !== 'admin') {
      lessonsData.teacherId = req.user.id;
    }

    const existingLesson = await LessonItem.findOne({
      teacherId: lessonsData.teacherId,
      fieldOfStudy: lessonsData.fieldOfStudy
    }).lean();

    if (existingLesson) {
      return res
        .status(400)
        .json({ message: 'Lesson already exists' });
    }

    const newLesson = new LessonItem(lessonsData);
    const savedLesson = await newLesson.save();

    if (savedLesson) {
      await updateUserRole(savedLesson.teacherId);
      await updateLessonItemsUserInfoByUserId(savedLesson.teacherId);
      returnLessonItem = await prepareDataLessonItemGet(savedLesson.
          toObject());
```

```
51        return res.json(returnLessonItem);
52      } else {
53        return res.status(400).json({
54          message: errorMessage
55        });
56      }
57    } catch (err) {
58      console.log(err);
59      return res.status(400).json({
60        message: errorMessage
61      });
62    }
63  });
```

GET, PUT and DELETE request can be often takes less lines of code than POST request.

```
1  app.get('/api/v1/lessons/:id', async (req, res) => {
2      // operationId: getLessonItemId
3      try {
4          const lessonItem = await LessonItem.findById(req.params.id).lean
               ();
5          res.json(lessonItem);
6      } catch (err) {
7          return res.status(400).json({
8          message: 'There was a problem getting the lesson item'
9          });
10     }
11     });
12
13     app.put('/api/v1/lessons/:id', requireAuth, async (req, res) => {
14     // operationId: updateLessonItemById
15     try {
16         const lessonItem = await LessonItem.findById(req.params.id).lean
               ();
17         if (isUserOwnerOrAdmin(lessonItem.teacherId, req.user.id)) {
18         await LessonItem.findOneAndUpdate(
19             { _id: req.params.id },
20             req.body
21         );
22         res.json({
23             message:
24             'Lesson item updated.'
25         });
26         } else {
27         return res.status(401).json({ message: 'Unauthorized' });
28         }
29     } catch (err) {
30         console.log(err);
31         return res.status(400).json({
32         message: 'There was a problem updating the lesson item'
33         });
34     }
35     });
36
37     app.delete('/api/v1/lessons/:id', requireAuth, async (req, res) => {
38     // operationId: deleteLessonItemById
39     try {
40         const lessonItem = await LessonItem.findById(req.params.id).lean
```

```
            ();
41          if (isUserOwnerOrAdmin(lessonItem.teacherId, req.user.id)) {
42          const lessonItem = await LessonItem.findByIdAndRemove(req.params.
                id);
43          await updateUserRole(lessonItem.teacherId);
44          res.json({
45              message: "Lesson item deleted"
46          });
47          } else {
48          return res.status(401).json({ message: 'Unauthorized' });
49          }
50      } catch (err) {
51          console.log(err);
52          return res.status(400).json({
53          message: 'There was a problem deleting the lesson item'
54          });
55      }
56      });
```

One of the exception can be GET request that is used to find elements with constrains:

```
1   app.get('/api/v1/lessons', async (req, res) => {
2       // operationId: findLessonItemByStatus
3       // firstFreeLesson: req.query.firstFreeLesson
4       // availableOnline: true
5       searchValue = {};
6
7       if (typeof req.query.fieldOfStudy === 'string' || req.query.
            fieldOfStudy instanceof String) {
8           searchValue.fieldOfStudy = req.query.fieldOfStudy;
9       }
10
11      if (typeof req.query.lessonPlaceMode === 'string' || req.query.
            lessonPlaceMode instanceof String) {
12          searchValue.places = req.query.lessonPlaceMode;
13      } else if (req.query.educationLevel instanceof Object) {
14          searchValue.places = { $in: req.query.lessonPlaceMode };
15      }
16
17      if (typeof req.query.educationLevel === 'string' || req.query.
            educationLevel instanceof String) {
18          searchValue.educationLevel = req.query.educationLevel;
19      } else if (req.query.educationLevel instanceof Object) {
20          searchValue.educationLevel = { $in: req.query.educationLevel };
21      }
22
23      if (typeof req.query.teachingLanguages === 'string' || req.query.
            teachingLanguages instanceof String) {
24          searchValue.teachingLanguages = req.query.teachingLanguages;
25      } else if (req.query.teachingLanguages instanceof Object) {
26          searchValue.teachingLanguages = { $in: req.query.
                teachingLanguages };
27      }
28
29      if (req.query.hourlyRateMin !== undefined && req.query.hourlyRateMax
            !== undefined) {
30          searchValue.hourlyRate = { $gte: req.query.hourlyRateMin, $lte:
                req.query.hourlyRateMax };
```

```
    } else if (req.query.hourlyRateMin !== undefined) {
        searchValue.hourlyRate = { $gte: req.query.hourlyRateMin };
    } else if (req.query.hourlyRateMax !== undefined) {
        searchValue.hourlyRate = { $lte: req.query.hourlyRateMax };
    }

    // console.log(searchValue);

    try {
        const lessonItems = await LessonItem.find(searchValue).lean();
        res.json(lessonItems);
    } catch (err) {
        console.log(err);
        return res.status(400).json({
        message: 'There was a problem getting the lesson item'
        });
    }
    });
```

Example data of users exported from database. First object is student user:

```
{
    "_id":{
        "$oid":"5fd9dd23b41481413cddd281"
    },
    "status":"active",
    "role":"student",
    "email":"test2@test.com",
    "firstName":"Antek",
    "lastName":"Chrabak",
    "dateOfBirth":{
        "$date":"2020-12-11T11:12:53.759Z"
    },
    "educationLevel":"pl_level1_group_1",
    "password":"$2a$12$SyHNDOEBAwvTQ06/yFl2iOhPHaky5N2zhB4fWMzUo.
        epwR6wLh45e",
    "registrated":{
        "$date":"2020-12-11T11:13:15.781Z"
    },
    "__v":0,
    "contactInfo":{
        "_id":{
            "$oid":"5ff1a3c854ce411748ee12a4"
        },
        "phone":{
            "_id":{
                "$oid":"5ff1a3c854ce411748ee12a5"
            },
            "number":"111111111",
            "visible":true
        },
        "email":{
            "_id":{
                "$oid":"5ff1a3c854ce411748ee12a6"
            },
            "email":"1111@aa",
            "visible":true
        },
```

```
37          "firstFreeLessonEnable":true,
38          "description":"Beautiful teacher"
39      }
40  }
```

Second example is teacher user:

```
1       {
2       "_id":{
3           "$oid":"5fdbd2f59b290b21e80697c9"
4       },
5       "status":"active",
6       "role":"teacher",
7       "email":"test3@test.com",
8       "firstName":"Antoni",
9       "lastName":"Andrzejek",
10      "dateOfBirth":{
11          "$date":"2018-12-12T22:00:00.000Z"
12      },
13      "educationLevel":"pl_level3_element_1",
14      "password":"$2a$12$7.xqpqVsqsFRH6KWlfOYueN5OSbR/WlMAY8.fBZk.
            eMFnCWIyVVgu",
15      "registrated":{
16          "$date":"2020-12-11T11:13:15.781Z"
17      },
18      "__v":0,
19      "phone":"+46111",
20      "contactInfo":{
21          "_id":{
22              "$oid":"5ff1fe63116c8417700a140d"
23          },
24          "phone":{
25              "_id":{
26                  "$oid":"5ff1fe63116c8417700a140e"
27              },
28              "number":"+118000",
29              "visible":true
30          },
31          "email":{
32              "_id":{
33                  "$oid":"5ff1fe63116c8417700a140f"
34              },
35              "email":"niepiszdomnie@cc.ok",
36              "visible":true
37          },
38          "firstFreeLessonEnable":true,
39          "description":"Lorem Ipsum is simply dummy text of the printing and
                typesetting industry. Lorem Ipsum has been the industry's
               standard dummy text ever since the 1500s, when an unknown
               printer took a galley of type and scrambled it to make a type
               specimen book. It has survived not only five centuries, but
               also the leap into electronic typesetting, remaining
               essentially unchanged. It was popularised in the 1960s with the
                release of Letraset sheets containing Lorem Ipsum passages,
               and more recently with desktop publishing software like Aldus
               PageMaker including versions of Lorem Ipsum."
40      }
41  }
```

Example below presents Lesson Item object stored in database:

```
{
    "_id":{
        "$oid":"5fef6ce79a838e2e9823f02f"
    },
    "educationLevel":[
        "pl_level3_element_1"
    ],
    "places":[

    ],
    "teachingLanguages":[
        "teach_lan_spanish"
    ],
    "teacherId":{
        "$oid":"5fdbd2f59b290b21e80697c9"
    },
    "hourlyRate":0,
    "onlineModeEnable":true,
    "faceToFaceModeStudentPlaceEnable":true,
    "faceToFaceModeTeacherPlaceEnable":true,
    "description":"asdasdasd",
    "fieldOfStudy":"lan_french",
    "lessonFeedback":[
        {
            "valuableFeedback":[

            ],
            "notValuableFeedback":[

            ],
            "_id":{
                "$oid":"5ff2266e3b7b0c3fac6cf922"
            },
            "description":"lkajsdlkamdlsk",
            "rating":3,
            "studentId":{
                "$oid":"5ff224fc68be3343182fe527"
            },
            "feedbackPostDate":{
                "$date":"2021-01-03T20:17:50.725Z"
            }
        }
    ],
    "__v":0,
    "email":"niepiszdomnie@cc.ok",
    "firstFreeLessonEnable":true,
    "phoneNumber":"+118000",
    "numberOfFeedbacks":1,
    "rating":3
}
```

4 Frontend Implementation

Frontend was written in JavaScript language using React library.

4.1 Communication with backend

Submit of the forms action was putting saveLessonItem(data parsed form fields in a form) and forward them to database using POST and PUT requests.

```
const { data } = lessonItem === undefined ? (
    await fetchContext.authAxios.post(
      `/v1/lessons`,
      saveLessonItem
    )
  ) : (
      await fetchContext.authAxios.put(
        `/v1/lessons/` + saveLessonItem._id,
        saveLessonItem
      )
    );
```

Receiving data from backend was realized as getting data from database and setting variables with received data.

```
const { data } = await fetchContext.authAxios.get(
    '/v1/users/lessons'
  );
setLessonItems(data);
```

4.2 User interface description

Basic user operations as *Login* and *Sign Up* was shown at the figures 2 and 3.

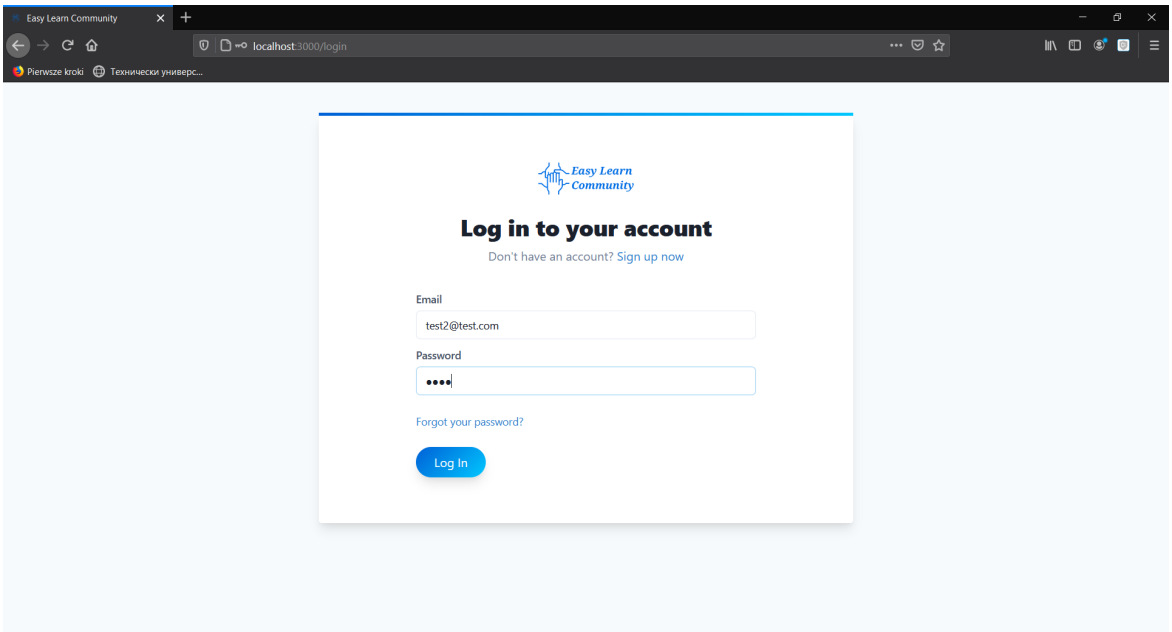

Figure 2: Login view

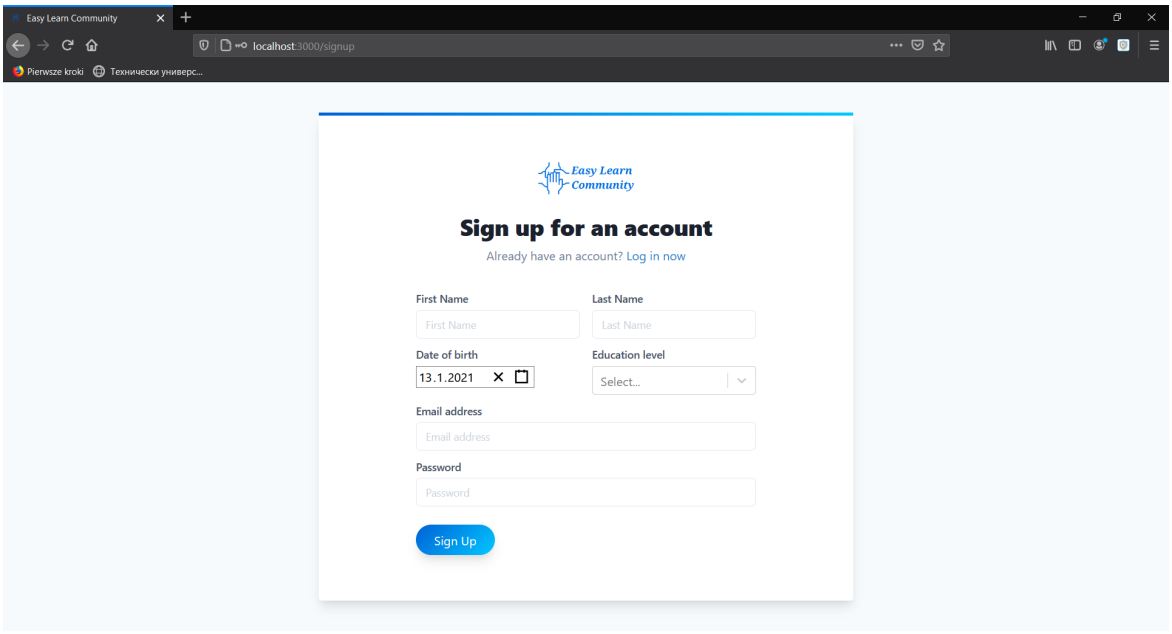

Figure 3: Sign up view

Newly created user is set as student from default.
Student and Teacher can find a lesson(figure nr 4), so they can improve their skills.

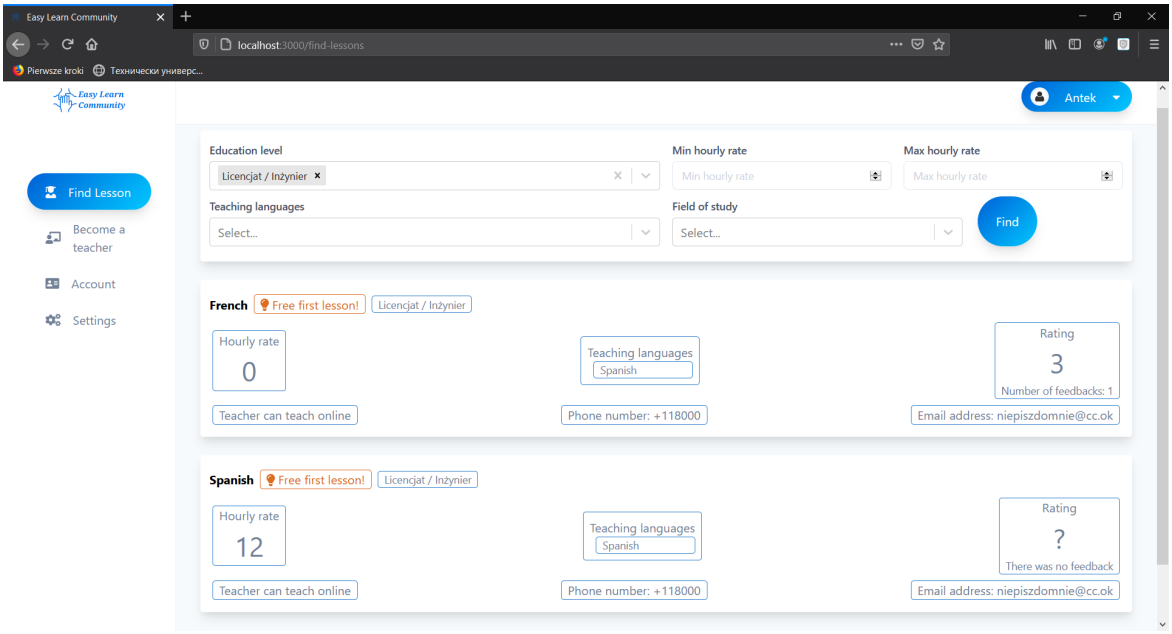

Figure 4: Find lesson view

Student can become a teacher by selecting *Become a teacher* tab, filling contact data(figure nr 5) and adding new lesson item(figure nr 6).

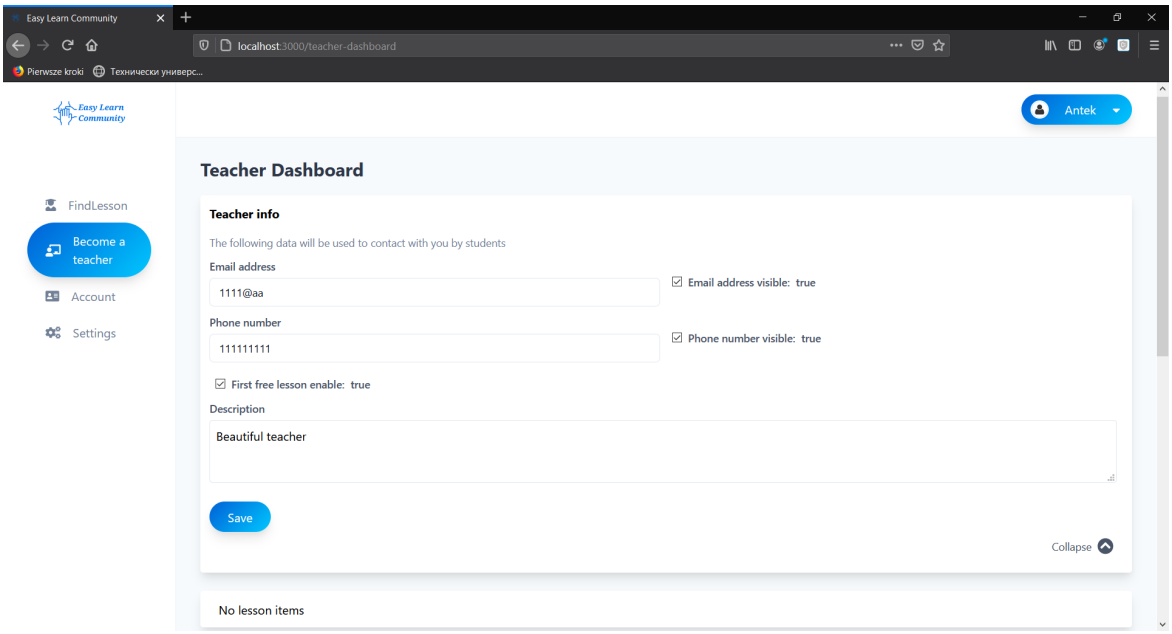

Figure 5: Become a teacher - fill contact data

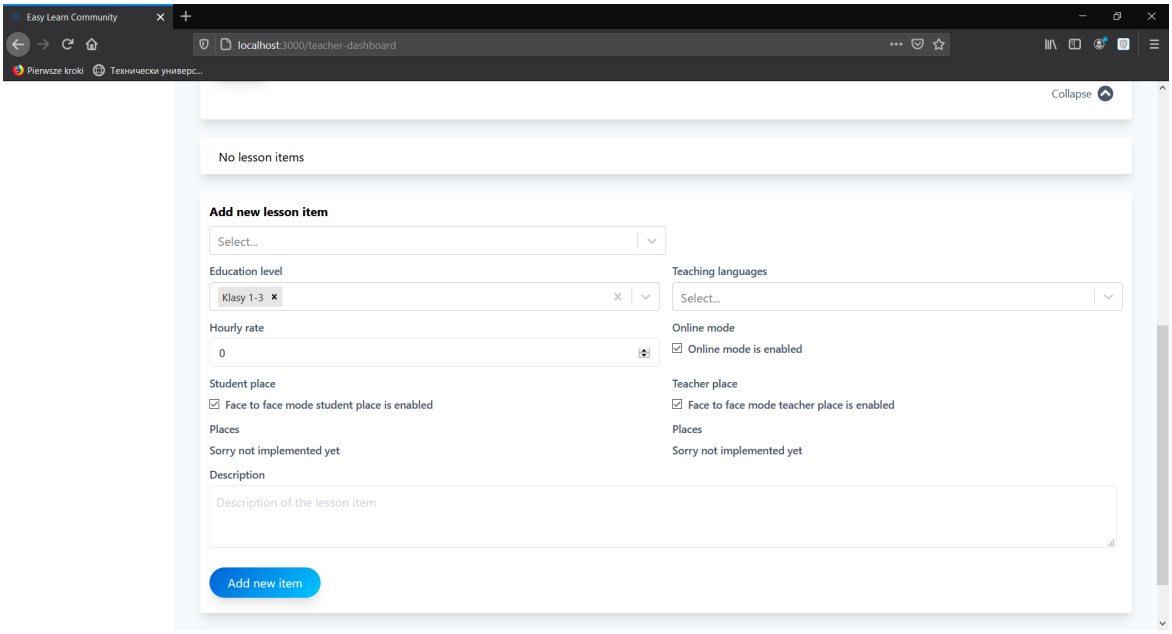

Figure 6: Become a teacher - add new lesson item

After logout and login user role is updated to *Teacher* and teacher lesson items can be managed from section *Your Lesson Items*(figure nr 7)

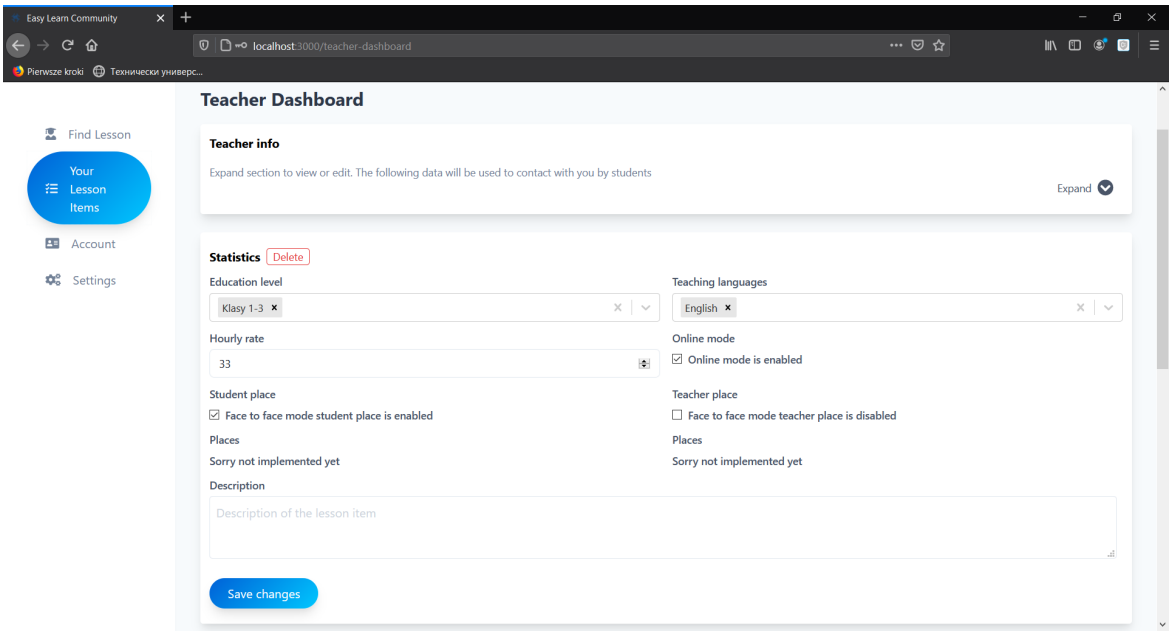

Figure 7: Teacher dashboard

Admin account allows to manage all lesson items *Lesson Items Administration* and users accounts *Users*(figure nr 8)

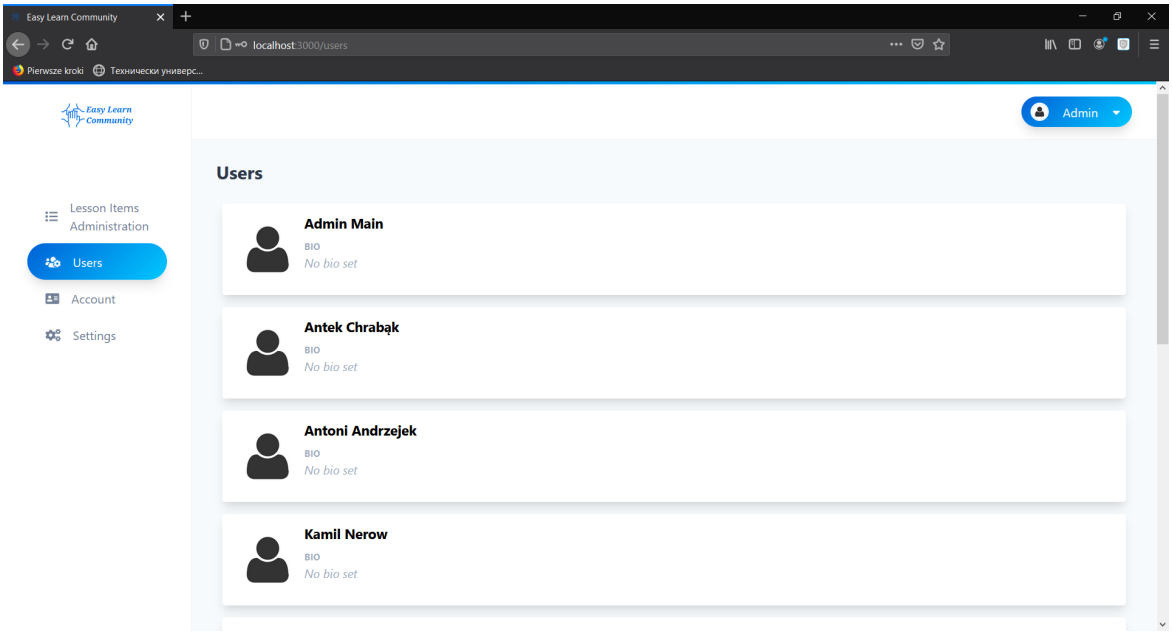

Figure 8: Become a teacher - add new lesson item

Every user is allowed to change *Personal Info*, *Sensitive data* and set new password *Change password*(figure nr 9)

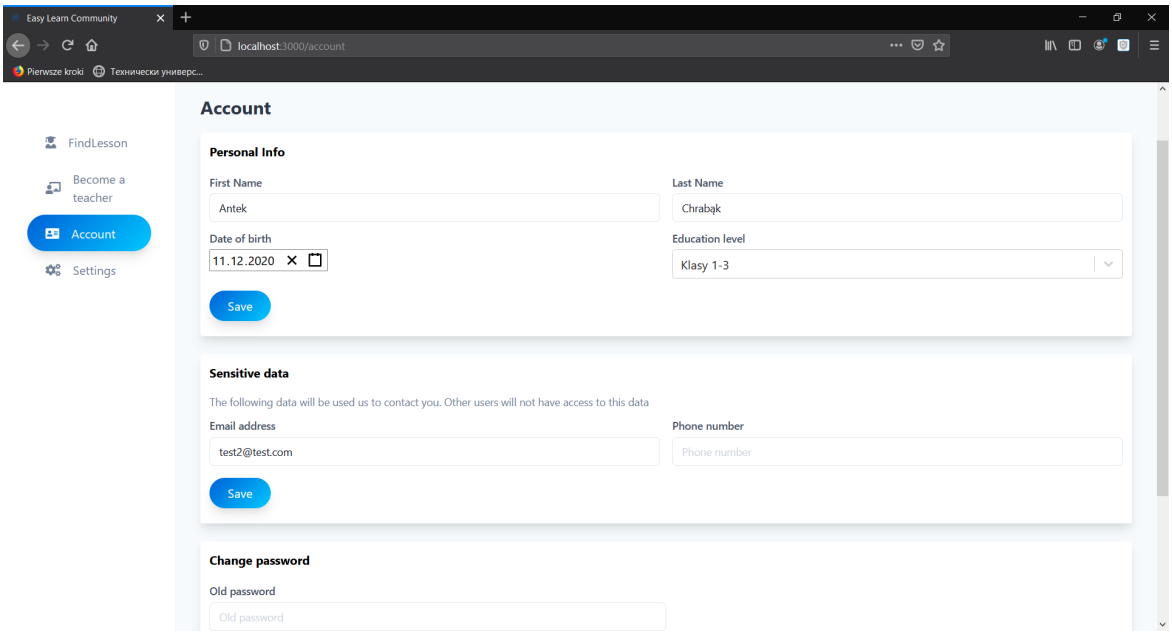

Figure 9: Account settings

5 Appendix A

```
openapi: 3.0.0
info:
  description: Easy Learn Community
  version: 1.0.1
  title: Easy Learn Community
  contact:
    email: kornelstefanczyk@wp.pl
tags:
  - name: users
    description: Operations about users
  - name: lessons
    description: Everything about your Lessons
  - name: report
    description: Everything about report
paths:
  /api/v1/signup:
    post:
      tags:
        - users
      summary: Create user
      description: Sign up user. Create new user account.
      operationId: createUser
      responses:
        "200":
          description: successful operation
        "400":
          description: "Bad request"
      requestBody:
        content:
          application/json:
```

```
31              schema:
32                $ref: "#/components/schemas/UserPost"
33          description: Created user object
34          required: true
35    /api/v1/users/login:
36      post:
37        tags:
38          - users
39        summary: Logs user into the system
40        operationId: loginUser
41        responses:
42          "200":
43            description: successful operation
44            headers:
45              X-Rate-Limit:
46                description: calls per hour allowed by the user
47                schema:
48                  type: integer
49                  format: int32
50              X-Expires-After:
51                description: date in UTC when token expires
52                schema:
53                  type: string
54                  format: date-time
55            content:
56              application/json:
57                schema:
58                  $ref: "#/components/schemas/UserGetPut"
59          "400":
60            description: Invalid email/password supplied
61        requestBody:
62          content:
63            application/json:
64              schema:
65                type: object
66                properties:
67                  email:
68                    description: The email for login
69                    type: string
70                    format: email
71                    example: "test@test.com"
72                  password:
73                    description: The password for login in clear text
74                    type: string
75                    example: "test"
76    /api/v1/users:
77      get:
78        tags:
79          - users
80        description: Get data of current logged in user
81        summary: Get current logged in user
82        operationId: getLoggedInUser
83        responses:
84          "200":
85            description: successful operation
86            content:
87              application/json:
```

```
88                schema:
89                  $ref: "#/components/schemas/UserGetPut"
90      put:
91        tags:
92          - users
93        summary: Updated current logged in user
94        description: This can only be done by the logged in user.
95        operationId: updateLoggedInUser
96        responses:
97          "200":
98            description: successful operation
99          "404":
100           description: User not found
101       requestBody:
102         content:
103           application/json:
104             schema:
105               $ref: "#/components/schemas/UserGetPut"
106         description: Updated user object
107         required: true
108     delete:
109       tags:
110         - users
111       summary: Delete current logged in user
112       description: This can only be done by the logged in user.
113       operationId: deleteLoggedInUser
114       responses:
115         "401":
116           description: Unauthorized
117         "404":
118           description: User not found
119   /api/v1/users/password:
120     put:
121       tags:
122         - users
123       summary: Updates password of current logged in user
124       description: This can only be done by the logged in user.
125       operationId: updateLoggedInUserPassword
126       responses:
127         "200":
128           description: successful operation
129         "404":
130           description: User not found
131       requestBody:
132         content:
133           application/json:
134             schema:
135               $ref: "#/components/schemas/UserChangePasswordPut"
136         description: Updated user object
137         required: true
138   /api/v1/users/lessons:
139     get:
140       tags:
141         - users
142       summary: Get lesson items of current logged in user
143       description: This can only be done by the logged in user.
144       operationId: getLessonItemsLoggedInUser
```

```
145       responses:
146         "200":
147           description: successful operation
148           content:
149             application/json:
150               schema:
151                 type: array
152                 items:
153                   $ref: "#/components/schemas/LessonItemGet"
154         "400":
155           description: Invalid status value
156   "/api/v1/users/{id}":
157     get:
158       tags:
159         - users
160       description: Admin role required
161       summary: Get user by id
162       operationId: getUserById
163       parameters:
164         - name: id
165           in: path
166           description: The Id that needs to be fetched.
167           required: true
168           schema:
169             type: string
170       responses:
171         "200":
172           description: successful operation
173           content:
174             application/json:
175               schema:
176                 $ref: "#/components/schemas/UserGetPut"
177         "401":
178           description: Unauthorized
179         "404":
180           description: User not found
181     put:
182       tags:
183         - users
184       summary: Updated user by id
185       description: Admin role required
186       operationId: updateUserById
187       parameters:
188         - name: id
189           in: path
190           description: The Id that needs to be fetched.
191           required: true
192           schema:
193             type: string
194       responses:
195         "200":
196           description: successful operation
197         "401":
198           description: Unauthorized
199         "404":
200           description: User not found
201       requestBody:
```

```
202         content:
203           application/json:
204             schema:
205               $ref: "#/components/schemas/UserGetPut"
206         description: Updated user object
207         required: true
208       delete:
209         tags:
210           - users
211         summary: Delete user by id
212         description: Admin role required
213         operationId: deleteUserById
214         parameters:
215           - name: id
216             in: path
217             description: The Id that needs to be deleted
218             required: true
219             schema:
220               type: string
221         responses:
222           "200":
223             description: successful operation
224           "401":
225             description: Unauthorized
226           "404":
227             description: User not found
228     "/api/v1/users/{id}/password":
229       put:
230         tags:
231           - users
232         summary: Updated password of user by user id
233         description: Admin role required
234         operationId: updateUserPasswordByUserId
235         parameters:
236           - name: id
237             in: path
238             description: The Id of user
239             required: true
240             schema:
241               type: string
242         responses:
243           "200":
244             description: successful operation
245           "404":
246             description: User not found
247         requestBody:
248           content:
249             application/json:
250               schema:
251                 $ref: "#/components/schemas/UserChangePasswordPut"
252           description: Updated user object
253           required: true
254     "/api/v1/users/{id}/info":
255       get:
256         tags:
257           - users
258         description: User info will be returned only if user is teacher
```

```
259       summary: Get teacher user info
260       operationId: getTeacherInfoById
261       parameters:
262         - name: id
263           in: path
264           description: The Id that needs to be fetched.
265           required: true
266           schema:
267             type: string
268       responses:
269         "200":
270           description: successful operation
271           content:
272             application/json:
273               schema:
274                 $ref: "#/components/schemas/UserInfoGet"
275         "400":
276           description: Invalid id supplied
277         "401":
278           description: Unauthorized
279         "404":
280           description: User not found
281   /api/v1/lessons:
282     post:
283       tags:
284         - lessons
285       summary: Add a new lesson item
286       operationId: addLessonItem
287       responses:
288         "200":
289           description: successful operation
290           content:
291             application/json:
292               schema:
293                 $ref: "#/components/schemas/LessonItemGet"
294         "405":
295           description: Invalid input
296       requestBody:
297         content:
298           application/json:
299             schema:
300               $ref: "#/components/schemas/LessonItemPost"
301     get:
302       tags:
303         - lessons
304       summary: Finds lesson items by params
305       description: Multiple status values can be provided with comma
            separated strings
306       operationId: findLessonItemByStatus
307       parameters:
308         - name: educationLevel
309           in: query
310           description: Education level values that need to be considered
                for filter
311           required: false
312           explode: true
313           schema:
```

```
314             type: array
315             items:
316               $ref: "#/components/schemas/EducationLevelValues"
317               example: "pl_level1_group_1"
318         - name: teachingLanguages
319           in: query
320           description: Education level values that need to be considered
                for filter
321           required: true
322           explode: true
323           schema:
324             type: array
325             items:
326               $ref: "#/components/schemas/TeachingLanguagesValues"
327         - name: fieldOfStudy
328           in: query
329           description: Field of study values that need to be considered
                for filter
330           required: true
331           explode: true
332           schema:
333             $ref: "#/components/schemas/FieldOfStudyValues"
334         - name: hourlyRateMin
335           in: query
336           description: Minimum hourly rate value that need to be
                considered for filter
337           required: false
338           schema:
339             type: number
340             format: float
341         - name: hourlyRateMax
342           in: query
343           description: Maximum hourly rate value that need to be
                considered for filter
344           required: false
345           schema:
346             type: number
347             format: float
348         - name: firstFreeLesson
349           in: query
350           description: First free lesson
351           required: false
352           schema:
353             type: boolean
354         - name: lessonPlaceMode
355           in: query
356           description: Place mode lesson
357           required: false
358           explode: true
359           schema:
360             type: array
361             items:
362               $ref: "#/components/schemas/LessonPlaceModeValues"
363         - name: availableOnline
364           in: query
365           description: Teacher have to has online status
366           required: false
```

```
367           schema:
368             type: boolean
369       responses:
370         "200":
371           description: successful operation
372           content:
373             application/json:
374               schema:
375                 type: array
376                 items:
377                   $ref: "#/components/schemas/LessonItemGet"
378         "400":
379           description: Invalid status value
380   "/api/v1/lessons/{id}":
381     get:
382       tags:
383         - lessons
384       summary: Get lesson item by lesson item id
385       operationId: getLessonItemId
386       parameters:
387         - name: id
388           in: path
389           description: The id that needs to be fetched.
390           required: true
391           schema:
392             type: string
393       responses:
394         "200":
395           description: successful operation
396           content:
397             application/json:
398               schema:
399                 $ref: "#/components/schemas/LessonItemGet"
400         "400":
401           description: Invalid id supplied
402         "404":
403           description: Lesson item not found
404     put:
405       tags:
406         - lessons
407       summary: Update an existing lesson item
408       description: This can only be done by the logged in user.
409       operationId: updateLessonItemById
410       parameters:
411         - name: id
412           in: path
413           description: id of lesson item that need to be updated
414           required: true
415           schema:
416             type: string
417       responses:
418         "200":
419           description: successful operation
420         "400":
421           description: Invalid ID supplied
422         "404":
423           description: Lesson item not found
```

```
        "405":
          description: Validation exception
      requestBody:
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/LessonItemPut"
        description: Updated user object
        required: true
    delete:
      tags:
      - lessons
      summary: Delete lesson item
      description: This can only be done by the logged in user.
      operationId: deleteLessonItemById
      parameters:
      - name: id
        in: path
        description: The id of the lesson item that needs to be deleted
        required: true
        schema:
          type: string
      responses:
        "200":
          description: successful operation
        "400":
          description: Invalid id supplied
        "404":
          description: Lesson item not found
  "/api/v1/lessons/{id}/feedbacks":
    post:
      tags:
      - lessons
      summary: Add a new feedback for lesson item
      operationId: addLessonItemFeedback
      parameters:
      - name: id
        in: path
        description: The id that needs to be fetched.
        required: true
        schema:
          type: string
      responses:
        "200":
          description: successful operation
        "405":
          description: Invalid input
      requestBody:
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/LessonFeedbackPostPut"
    get:
      tags:
      - lessons
      summary: Get lesson item feedbacks by lesson item id
      operationId: getLessonItemFeedbackId
```

```
    parameters:
      - name: id
        in: path
        description: The id that needs to be fetched.
        required: true
        schema:
          type: string
    responses:
      "200":
        description: successful operation
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: "#/components/schemas/LessonFeedbackGet"
      "400":
        description: Invalid id supplied
      "404":
        description: Lesson item not found
  put:
    tags:
      - lessons
    summary: Update an existing lesson item feedback
    description: This can only be done by the logged in user.
    operationId: updateLessonItemFeedbackIdById
    parameters:
      - name: id
        in: path
        description: id of lesson item that need to be updated
        required: true
        schema:
          type: string
    responses:
      "200":
        description: successful operation
      "400":
        description: Invalid ID supplied
      "404":
        description: Lesson item not found
      "405":
        description: Validation exception
    requestBody:
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/LessonFeedbackPostPut"
      description: Updated user object
      required: true
  delete:
    # deprecated: true
    tags:
      - lessons
    summary: Delete lesson item feedback
    description: This can only be done by the logged in user.
    operationId: deleteLessonItemFeedbackIdById
    parameters:
```

```
538         - name: id
539           in: path
540           description: The id of the lesson item
541           required: true
542           schema:
543             type: string
544         - name: studentId
545           in: query
546           description: The id of the user that feedback item needs to be
                deleted. Available only for admin
547           required: false
548           schema:
549             type: string
550       responses:
551         "200":
552           description: successful operation
553         "400":
554           description: Invalid id supplied
555         "404":
556           description: Lesson item not found
557 components:
558   schemas:
559     User:
560       allOf:
561         - $ref: "#/components/schemas/UserBasicData"
562         - $ref: "#/components/schemas/UserStatusData"
563         - $ref: "#/components/schemas/UserGetPut"
564         - $ref: "#/components/schemas/UserPost"
565     UserBasicData:
566       type: object
567       properties:
568         _id:
569           type: string
570         firstName:
571           type: string
572           example: "John"
573         lastName:
574           type: string
575           example: "Rambo"
576         email:
577           type: string
578           example: "test@test.com"
579         phone:
580           type: string
581           example: "+48555666333"
582         dateOfBirth:
583           type: string
584           format: date-time
585           example: "2000-12-02T00:00:00.000Z"
586         educationLevel:
587           $ref: "#/components/schemas/EducationLevelValues"
588         gender:
589           type: string
590     UserContactData:
591       type: object
592       properties:
593         phone:
```

```
        properties:
          number:
            type: string
          visible:
            type: boolean
      email:
        properties:
          email:
            description: As default email is the same as user email
            type: string
          visible:
            type: boolean
      photo:
        description: Feature not implemented yet
        type: string
      firstFreeLessonEnable:
        description: The teacher declare the first lesson will be for
          free
        type: boolean
      description:
        type: string
  UserStatusData:
    type: object
    properties:
      registered:
        type: string
        format: date-time
        example: "2020-06-11T11:12:53.759Z"
      status:
        type: string
        description: Order Status
        enum:
          - active
          - suspended
          - inactive
        example: "active"
      role:
        type: string
        description: Order Status
        enum:
          - user
          - student
          - teacher
          - admin
        example: "student"
  UserPost:
    allOf:
      - $ref: "#/components/schemas/UserBasicData"
      - $ref: "#/components/schemas/UserStatusData"
      - type: object
        required:
          - password
        properties:
          password:
            type: string
            example: "test"
  UserGetPut:
```

```
650         allOf:
651           - $ref: "#/components/schemas/UserBasicData"
652           - $ref: "#/components/schemas/UserStatusData"
653           - type: object
654             properties:
655               contactInfo:
656                 $ref: "#/components/schemas/UserContactData"
657       UserInfoGet:
658         allOf:
659           - type: object
660             properties:
661               registered:
662                 type: string
663                 format: date-time
664               _id:
665                 type: string
666               firstName:
667                 type: string
668               lastName:
669                 type: string
670               email:
671                 type: string
672               phone:
673                 type: string
674               firstFreeLessonEnable:
675                 description: The teacher declare the first lesson will be
                        for free
676                 type: boolean
677               educationLevel:
678                 $ref: "#/components/schemas/EducationLevelValues"
679               photo:
680                 description: Feature not implemented yet
681                 type: string
682               userDescription:
683                 type: string
684               availableOnline:
685                 description: The information about the current availability
                        of the tutor
686                 type: boolean
687               lessonItems:
688                 type: array
689                 items:
690                   allOf:
691                     - $ref: "#/components/schemas/LessonItemBasicData"
692                     - type: object
693                       properties:
694                         rating:
695                           description: Rating will be described as number
                                of stars
696                           type: number
697                           format: float
698                         numberOfFeedbacks:
699                           type: number
700       UserChangePasswordPut:
701         type: object
702         properties:
703           oldPassword:
```

```
704             type: string
705             example: "test"
706           newPassword:
707             type: string
708             example: "test"
709       LessonItem:
710         allOf:
711           - $ref: "#/components/schemas/LessonItemBasicData"
712           - $ref: "#/components/schemas/LessonItemPost"
713           - type: object
714             properties:
715               lessonFeedback:
716                 type: array
717                 items:
718                   $ref: "#/components/schemas/LessonFeedback"
719       LessonItemPost:
720         allOf:
721           - $ref: "#/components/schemas/LessonItemBasicData"
722           - type: object
723             properties:
724               fieldOfStudy:
725                 $ref: "#/components/schemas/FieldOfStudyValues"
726       LessonItemPut:
727         allOf:
728           - $ref: "#/components/schemas/LessonItemBasicData"
729       LessonItemGet:
730         allOf:
731           - $ref: "#/components/schemas/LessonItemBasicData"
732           - type: object
733             properties:
734               fieldOfStudy:
735                 $ref: "#/components/schemas/FieldOfStudyValues"
736               rating:
737                 description: Rating will be described as number of stars
738                 type: number
739                 format: float
740               numberOfFeedbacks:
741                 type: number
742               availableOnline:
743                 description: The information about the current availability
                        of the tutor
744                 type: boolean
745               phoneNumber:
746                 type: string
747               email:
748                 type: string
749               firstFreeLessonEnable:
750                 description: The teacher declare the first lesson will be
                        for free
751                 type: boolean
752       LessonItemBasicData:
753         type: object
754         properties:
755           _id:
756             type: string
757           teacherId:
758             type: string
```

```
        educationLevel:
          type: array
          items:
            $ref: "#/components/schemas/EducationLevelValues"
        places:
          type: array
          items:
            type: object
            format: text
        teachingLanguages:
          type: array
          items:
            $ref: "#/components/schemas/TeachingLanguagesValues"
        hourlyRate:
          type: number
          format: float
        onlineModeEnable:
          description: The possibility of online classes
          type: boolean
        faceToFaceModeStudentPlaceEnable:
          description: The possibility of teaching at the student's place
          type: boolean
        faceToFaceModeTeacherPlaceEnable:
          description: The possibility of teaching at the teacher's place
          type: boolean
        description:
          type: string
    LessonFeedback:
      allOf:
        - $ref: "#/components/schemas/LessonFeedbackBasicData"
        - type: object
          properties:
            valuableFeedback:
              type: array
              items:
                type: object
                properties:
                  userId:
                    type: string
            notValuableFeedback:
              type: array
              items:
                type: object
                properties:
                  userId:
                    type: string
    LessonFeedbackBasicData:
      type: object
      properties:
        feedbackPostDate:
          type: string
          format: date-time
          example: "2020-12-02T00:00:00.000Z"
        feedbackEditDate:
          type: string
          format: date-time
          example: "2020-12-06T00:00:00.000Z"
```

```
816         description:
817           type: string
818         rating:
819           description: Rating will be described as number of stars
820           type: number
821           format: float
822         studentId:
823           type: string
824     LessonFeedbackGet:
825       allOf:
826         - $ref: "#/components/schemas/LessonFeedbackBasicData"
827         - type: object
828           properties:
829             valuableFeedback:
830               type: number
831             notValuableFeedback:
832               type: number
833     LessonFeedbackPostPut:
834       allOf:
835         - $ref: "#/components/schemas/LessonFeedbackBasicData"
836     NotificationForm:
837       type: object
838       properties:
839         _id:
840           type: string
841           description: my event id
842         event:
843           type: string
844           description: my event type
845         data:
846           type: object
847     EducationLevelValues:
848       type: string
849       enum:
850         - "pl_level1_group_1"
851         - "pl_level1_group_2"
852         - "pl_level2_element_1"
853         - "pl_level2_element_2"
854         - "pl_level3_element_1"
855         - "pl_level3_element_2"
856         - "pl_level3_element_3"
857         - "pl_level4_element_1"
858       example: "pl_level1_group_1"
859     FieldOfStudyValues:
860       type: string
861       enum:
862         - "lan_english"
863         - "lan_german"
864         - "lan_french"
865         - "lan_russian"
866         - "lan_spanish"
867         - "lan_italian"
868         - "lan_polish"
869         - "lan_polish_for_foreigners"
870         - "lan_other"
871         - "hum_history"
872         - "hum_civics"
```

```
          - "hum_art_history"
          - "hum_other"
          - "sci_mathematics"
          - "sci_chemistry"
          - "sci_physics"
          - "sci_biology"
          - "sci_geography"
          - "sci_statistics"
          - "sci_other"
          - "compsci_informatics"
          - "compsci_computer_use"
          - "compsci_programming"
          - "compsci_other"
          - "art_musical_instruments"
          - "art_music"
          - "art_drawing"
          - "art_other"
          - "other_primary_learning"
          - "other_other"
        example: "compsci_informatics"
      TeachingLanguagesValues:
        type: string
        enum:
          - "teach_lan_polish"
          - "teach_lan_english"
          - "teach_lan_german"
          - "teach_lan_spanish"
          - "teach_lan_italian"
          - "teach_lan_french"
          - "teach_lan_russian"
          - "teach_lan_ukrainian"
      LessonPlaceModeValues:
        type: string
        enum:
          - "place_mode_online"
          - "place_mode_teacher"
          - "place_mode_student"
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
security:
  - bearerAuth: []
externalDocs:
  description: Swagger Petstore editor example
  url: "https://editor.swagger.io/?docExpansion=none"
servers:
  - url: "http://localhost:3001"
```