Projektowanie Aplikacji Internetowych 2 – projekt

3ID12A

Zespół:

Katarzyna Dawiec

Adrian Jakubczyk

Dominik Kaliszewski

Karol Kołodziejczyk

Temat: USOS

**Dokumentacja**

1. Przedstawienie aplikacji

USOS – Uniwersytecki System Obsługi Studiów – system ułatwiający zarządzanie studiami w szkole wyższej. Aplikacja jest wykorzystywana między innymi do zarządzania efektami kształcenia studentów, stypendiami, płatnościami za usługi edukacyjne.

Istnieją cztery grupy użytkowników, gdzie każda grupa ma określone uprawnienia:

* Rektor
  + Dodawanie nowych użytkowników
  + Usuwanie istniejących użytkowników
  + Podpinanie wykładowcy pod przedmiot
  + Ostateczne zaakceptowanie wniosku o stypendium
  + Zamykanie systemu i uczelni z powodu Koronavirusa
  + Dostęp do listy płatności
* Student
  + Wgląd do swoich ocen
  + Wgląd do wykazu przydzielonych stypendiów
  + Wnioskowanie o stypendium
  + Wgląd do decyzji
  + Opłacenie usług edukacyjnych
  + Opłacenie semestru (w przypadku studiów niestacjonarnych)
* Dziekan
  + Podpięcie studenta pod przedmiot
  + Usunięcie studenta z przedmiotu
  + Wstępne zaakceptowanie wniosku o stypendium
  + Dodawanie ogłoszeń
  + Otwieranie terminów
  + Zamykanie terminów
  + Dodawanie studenta do grupy dziekańskiej
  + Dodawanie studenta do grupy laboratoryjnej
* Wykładowca
  + Dodawanie nowych ocen
  + Zmiana istniejących ocen
  + Usunięcie istniejących ocen
  + Tworzenie syllabusa do przedmiotu

Do każdej roli dodawani są podstawowi użytkownicy:

* Rektor
  + Login: rector  
    hasło: rector
* Student
  + Login: student  
    hasło: student
* Dziekan
  + Login: dean  
    hasło: dean
* Wykładowca
  + Login: lecturer  
    hasło: lecturer

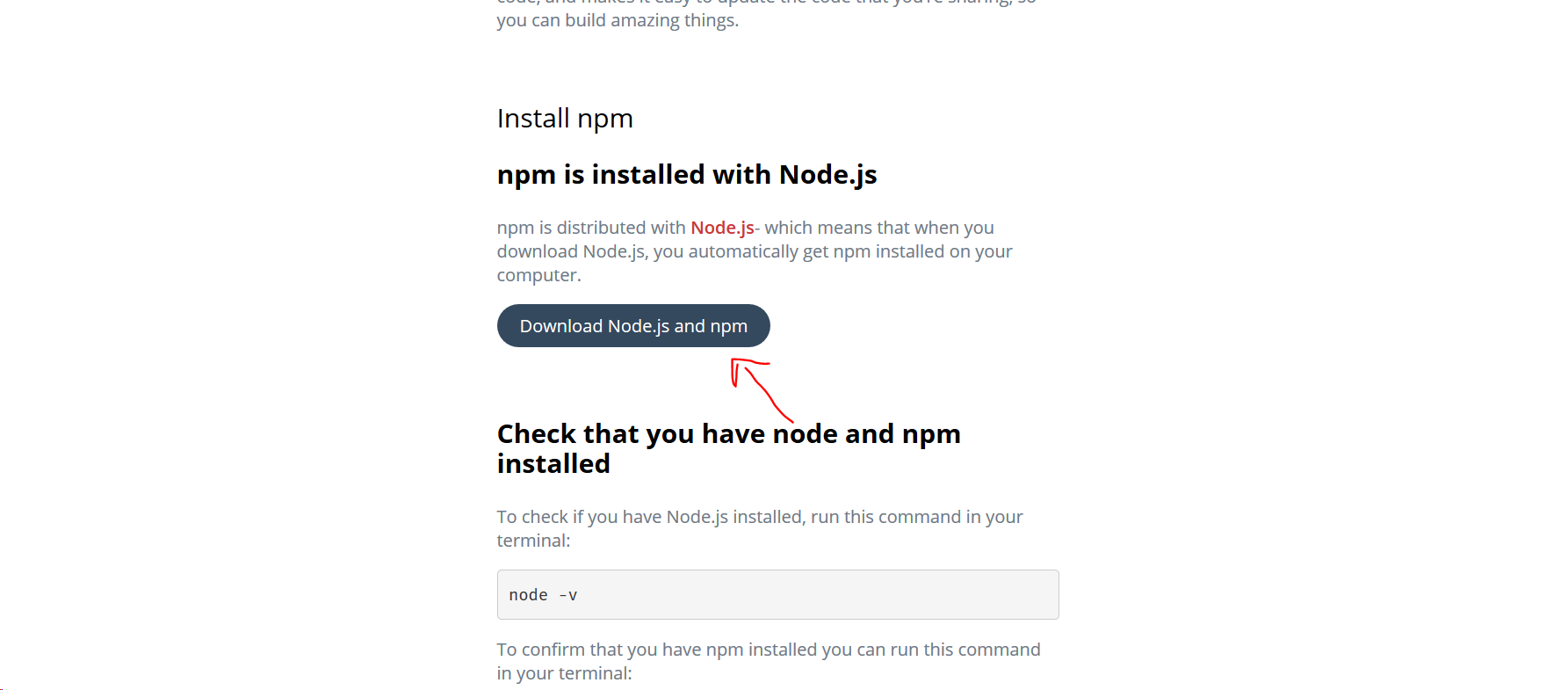
1. Wykorzystane technologie:

* Frontend:
  + Vue (Vue + Vuex + Vuetify)
  + Axios – zapytania HTTP
* Backend:
  + Java
  + MySQL
  + Spring
  + JUnit
  + Selenium

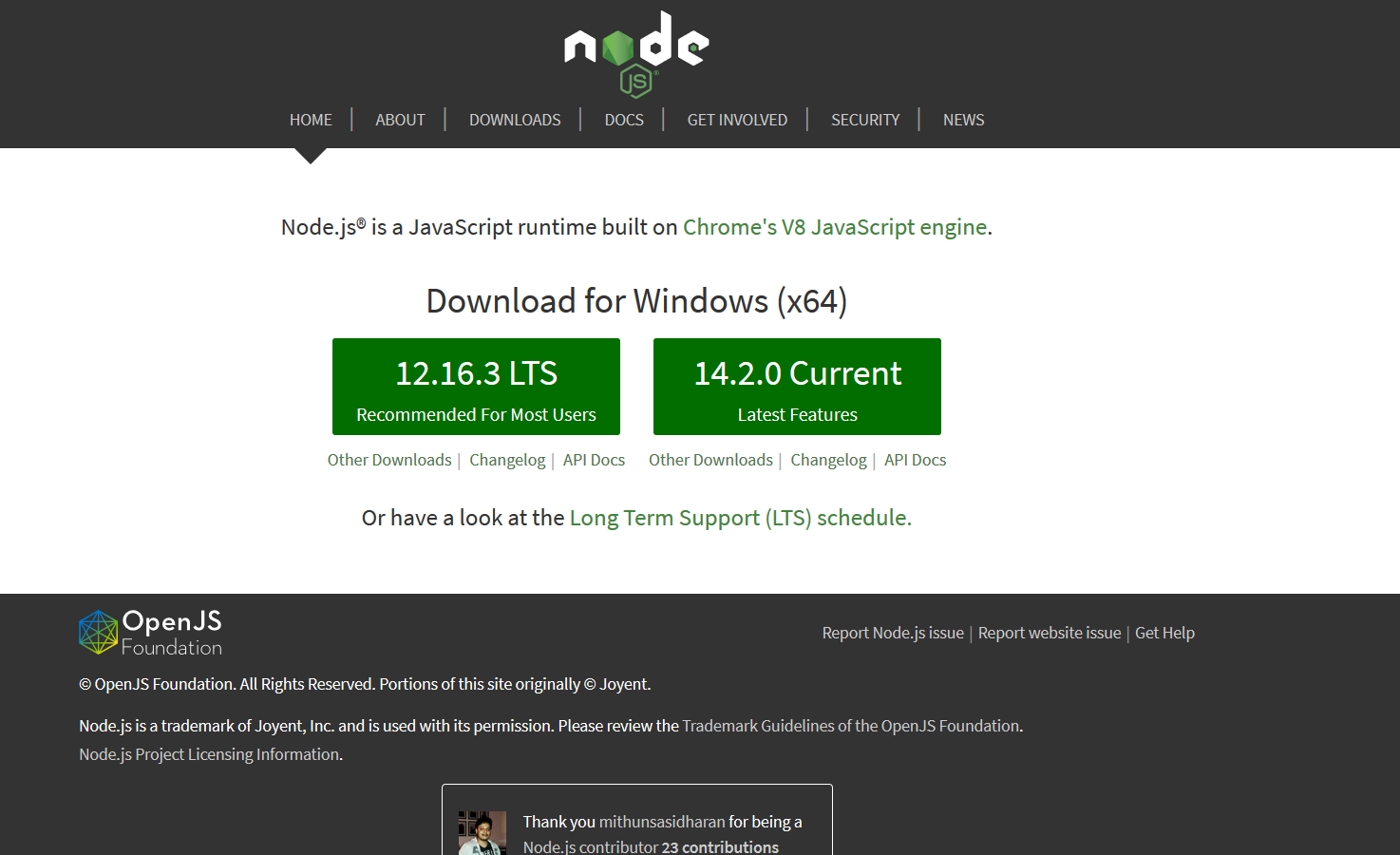
1. Instalacja potrzebnych narzędzi:

* npm + Node.js – domyślny manager pakietów dla środowiska Node.js

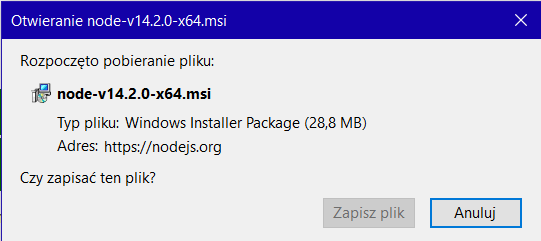
Aby zainstalować npm należy skorzystać ze strony: <https://www.npmjs.com/get-npm>, następnie w sekcji „Install npm” kliknąć na przycisk „Download Node.js and npm”



Ukaże się następująca strona:

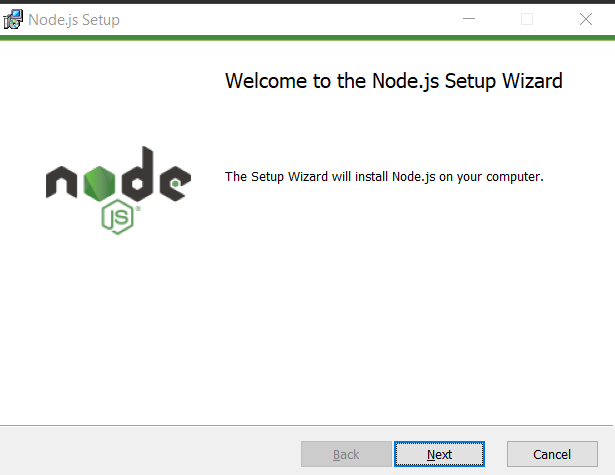


Po kliknięciu na przycisk „14.2.0 Current” pojawi się okno dialogowe:



Należy kliknąć „Zapisz plik”

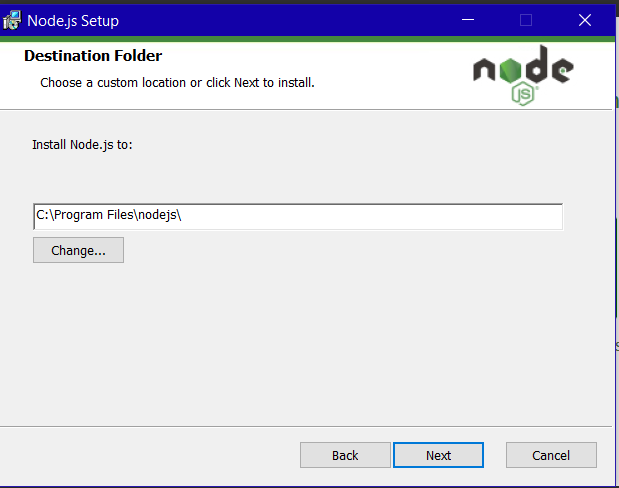
Po zakończeniu pobierania pojawi się okno instalacji:



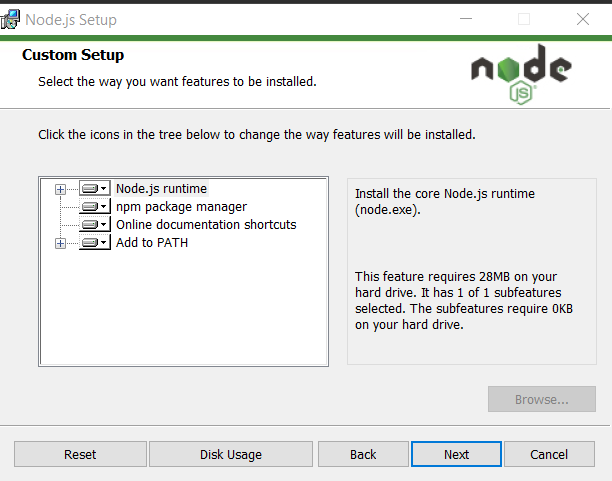
Należy zaakceptować postanowienia licencyjne

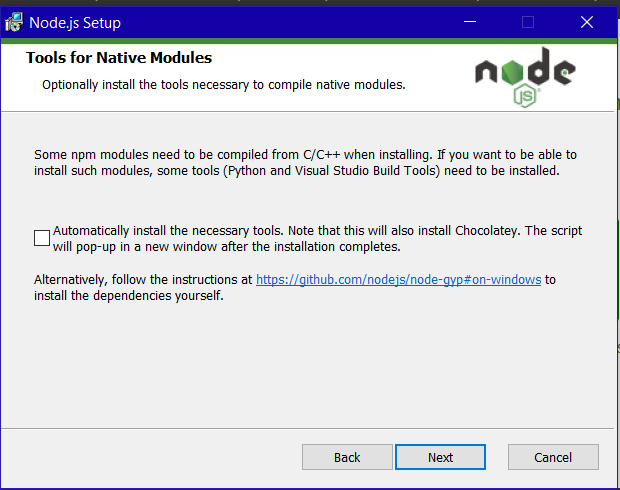


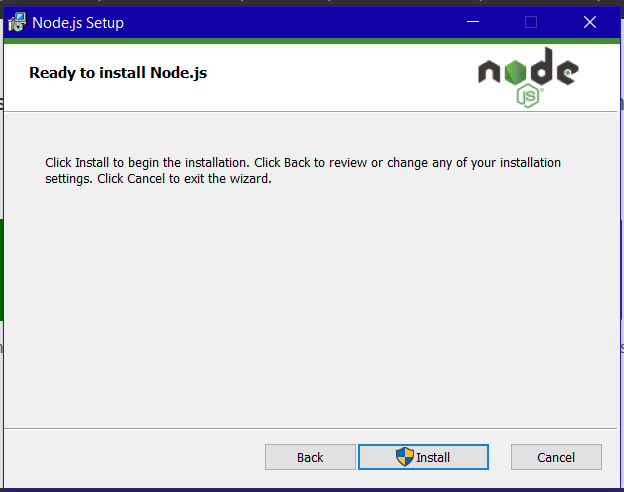
Wybrać lokalizację instalacji



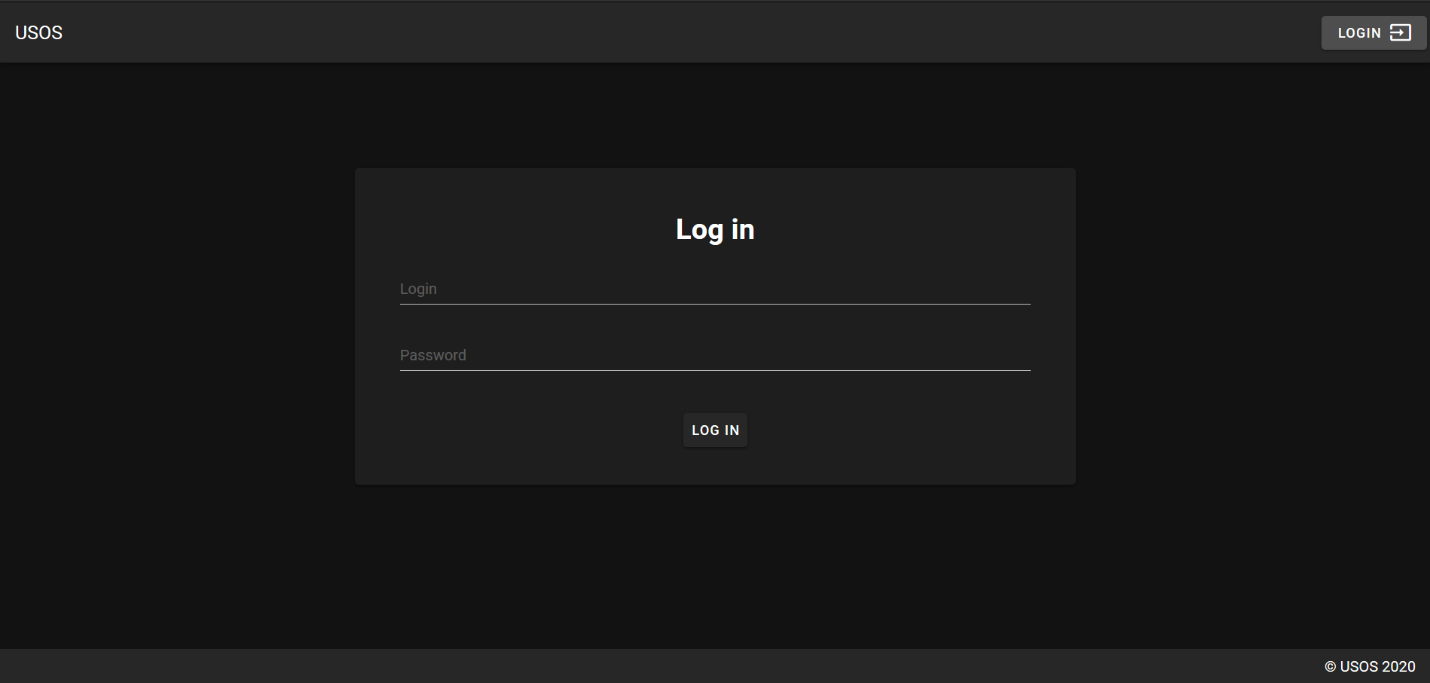
Wybrać co powinno zostać zainstalowane

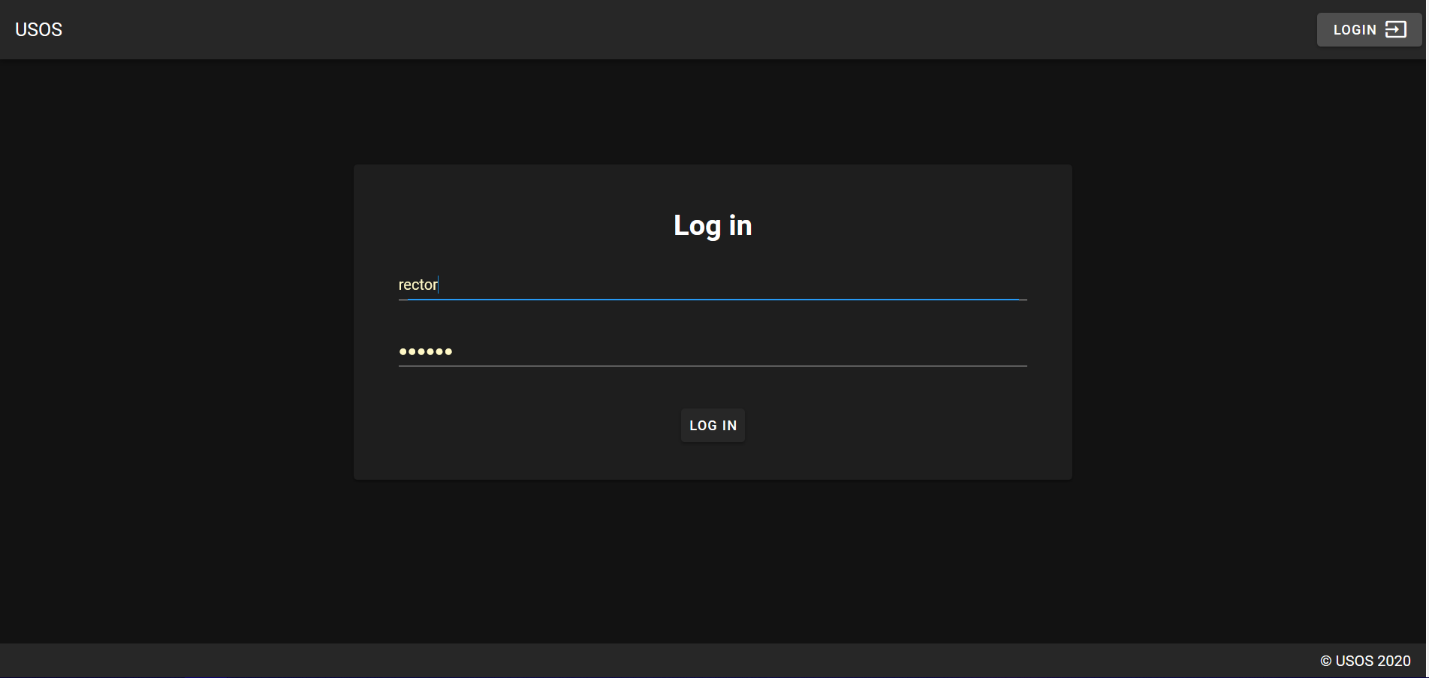
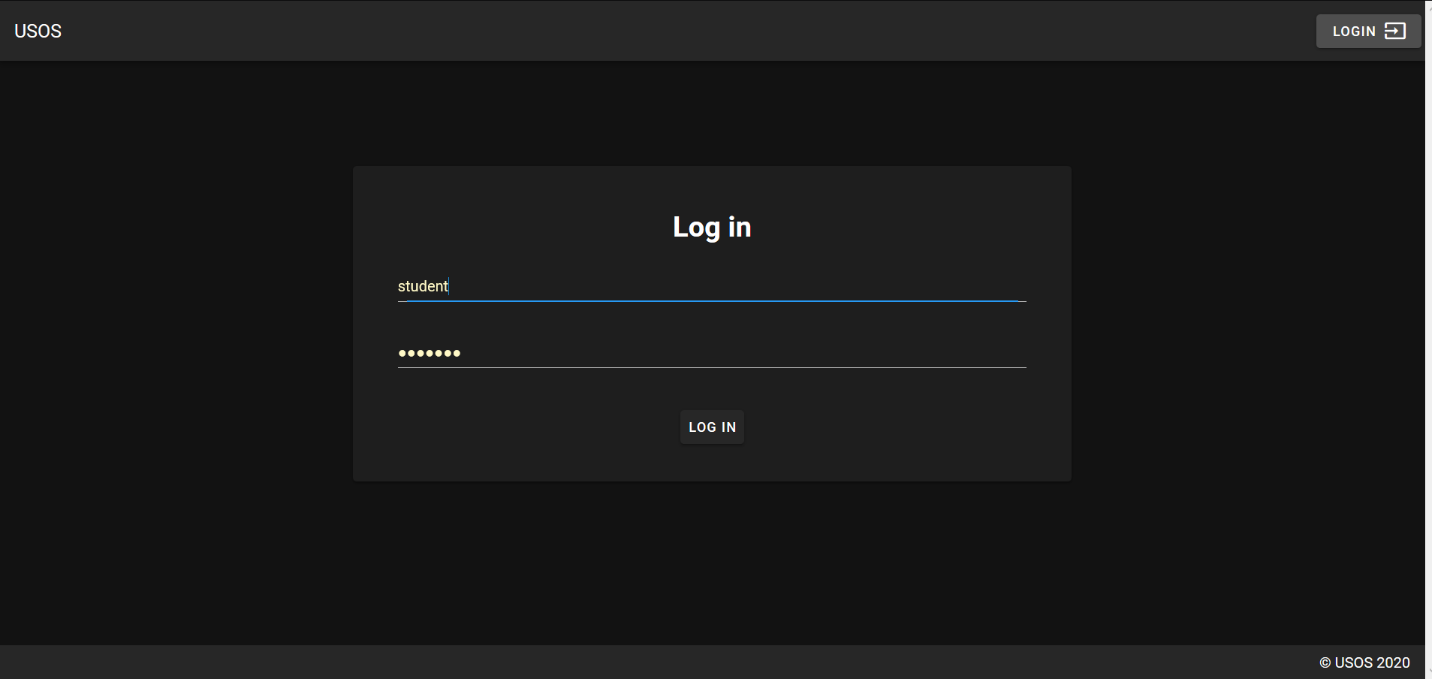


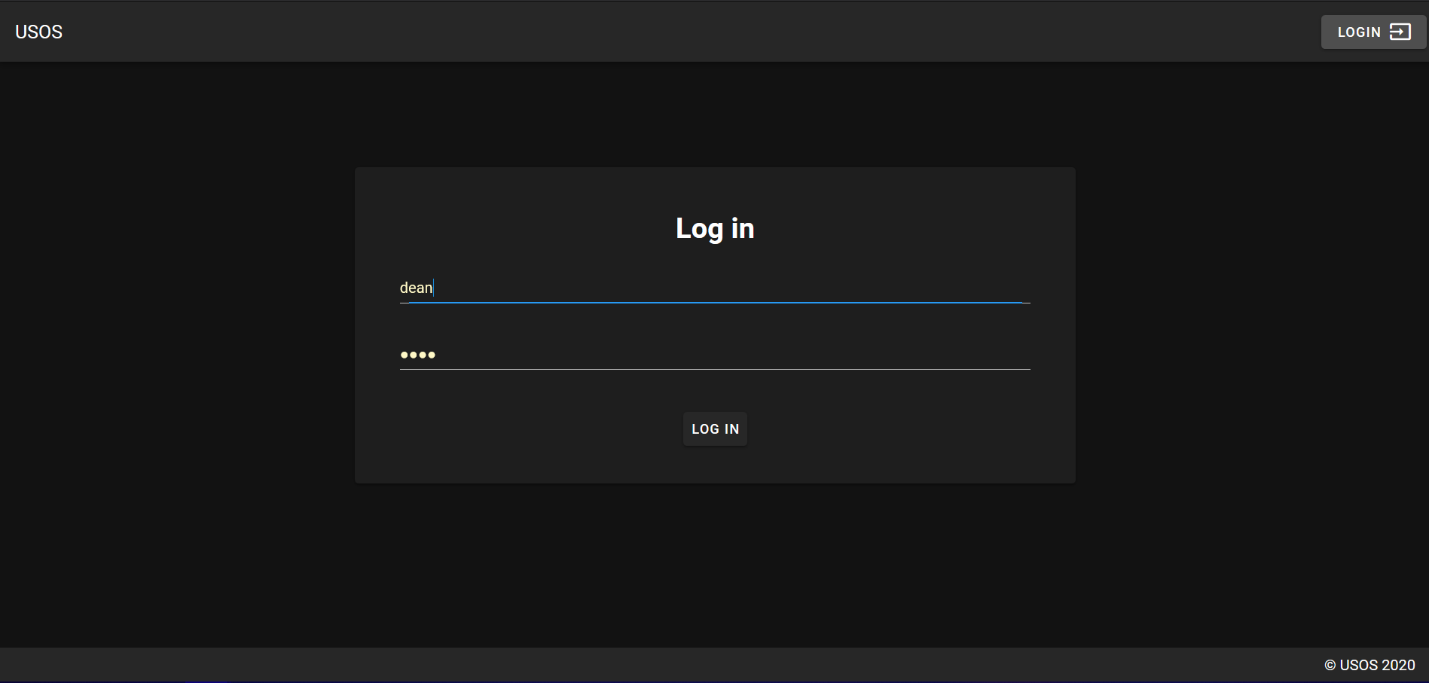
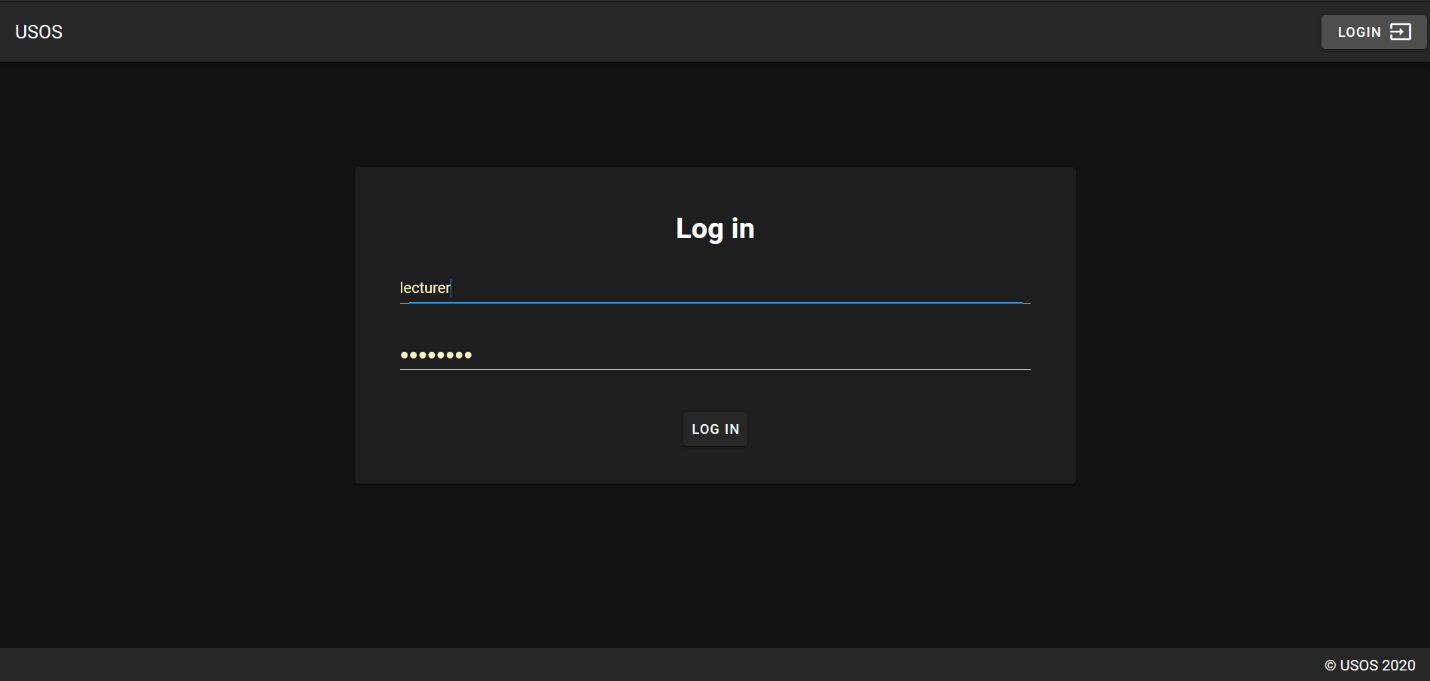




Kliknięcie przycisku „Install” spowoduje poproszenie o uprawnienia administratora, oraz nastapi przejście do instalacji.

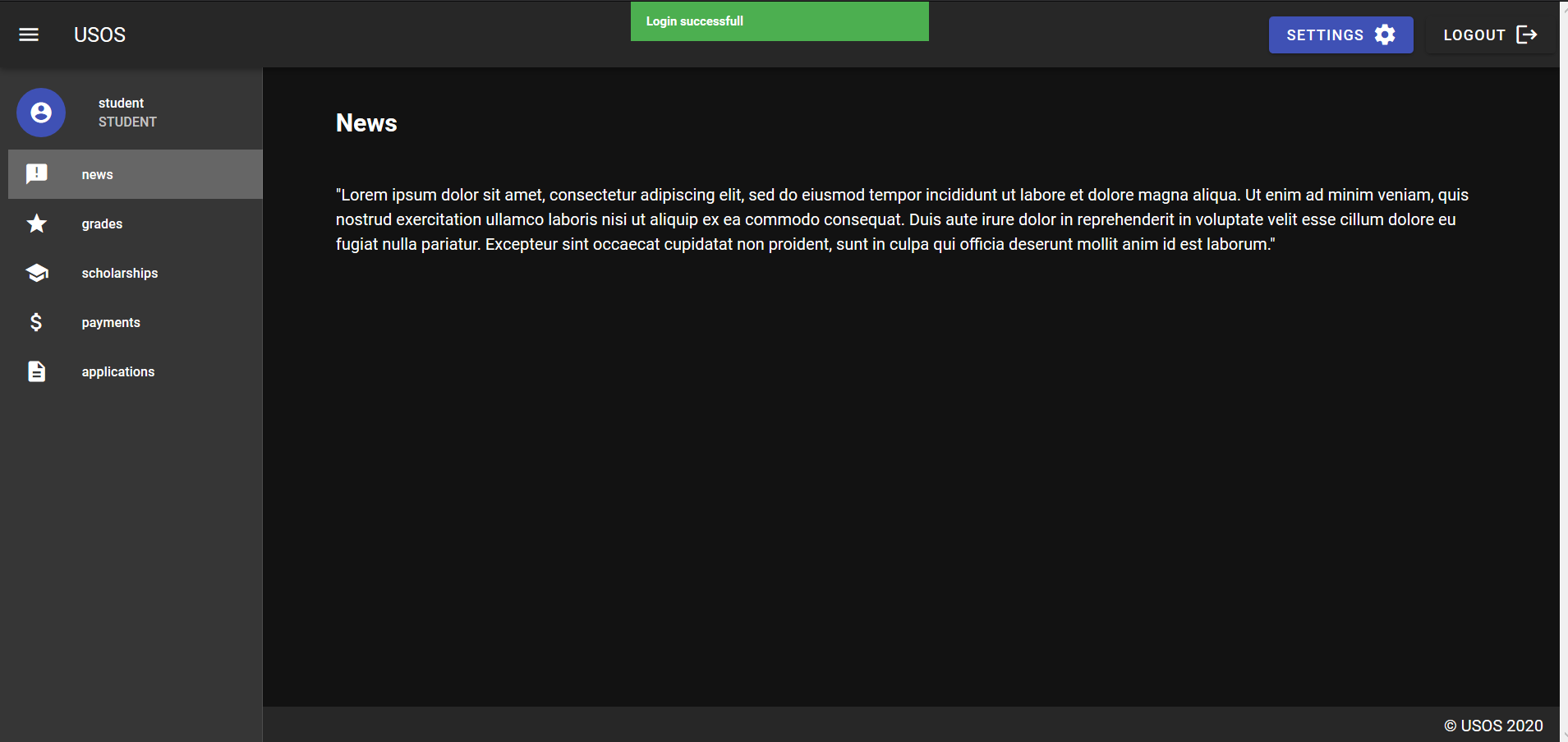
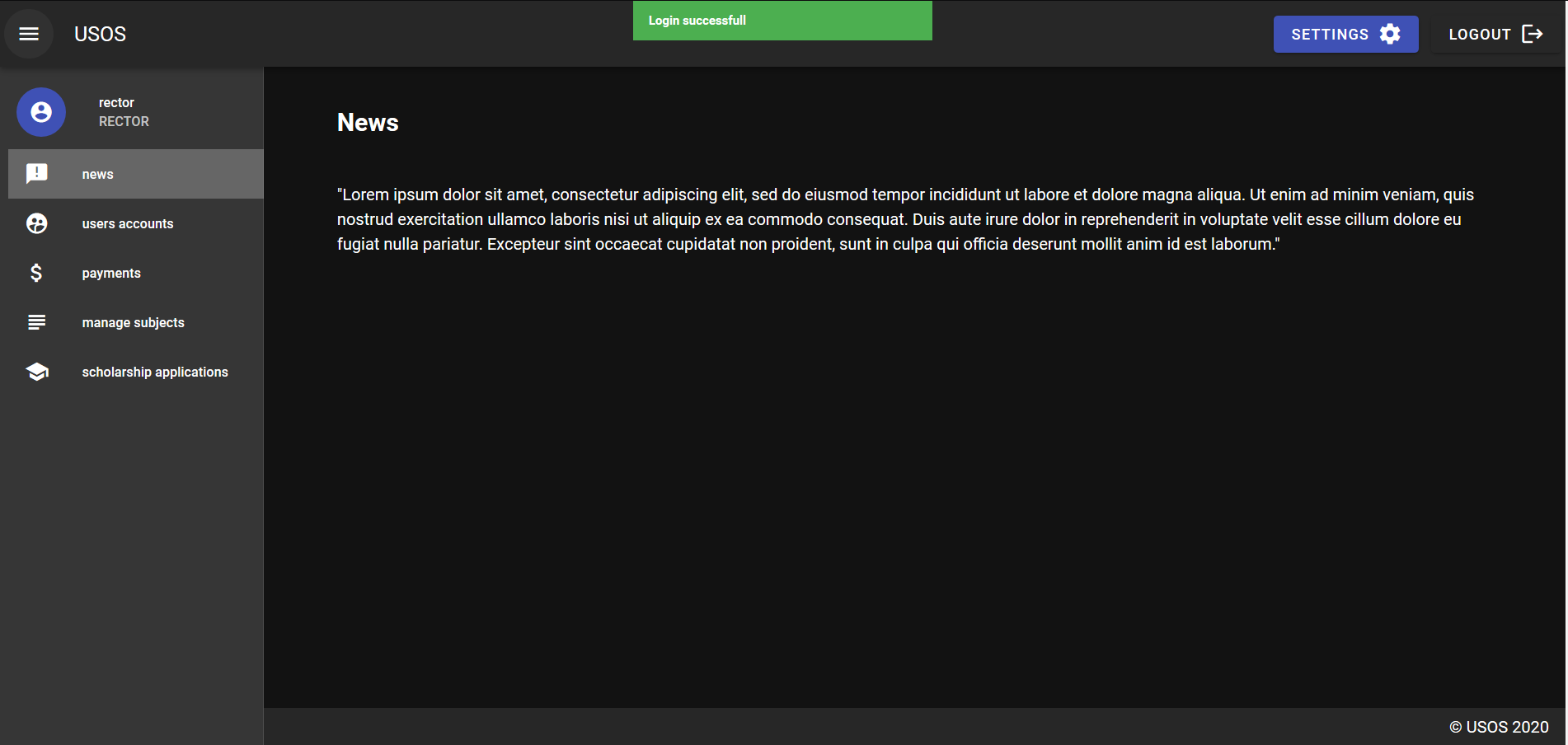
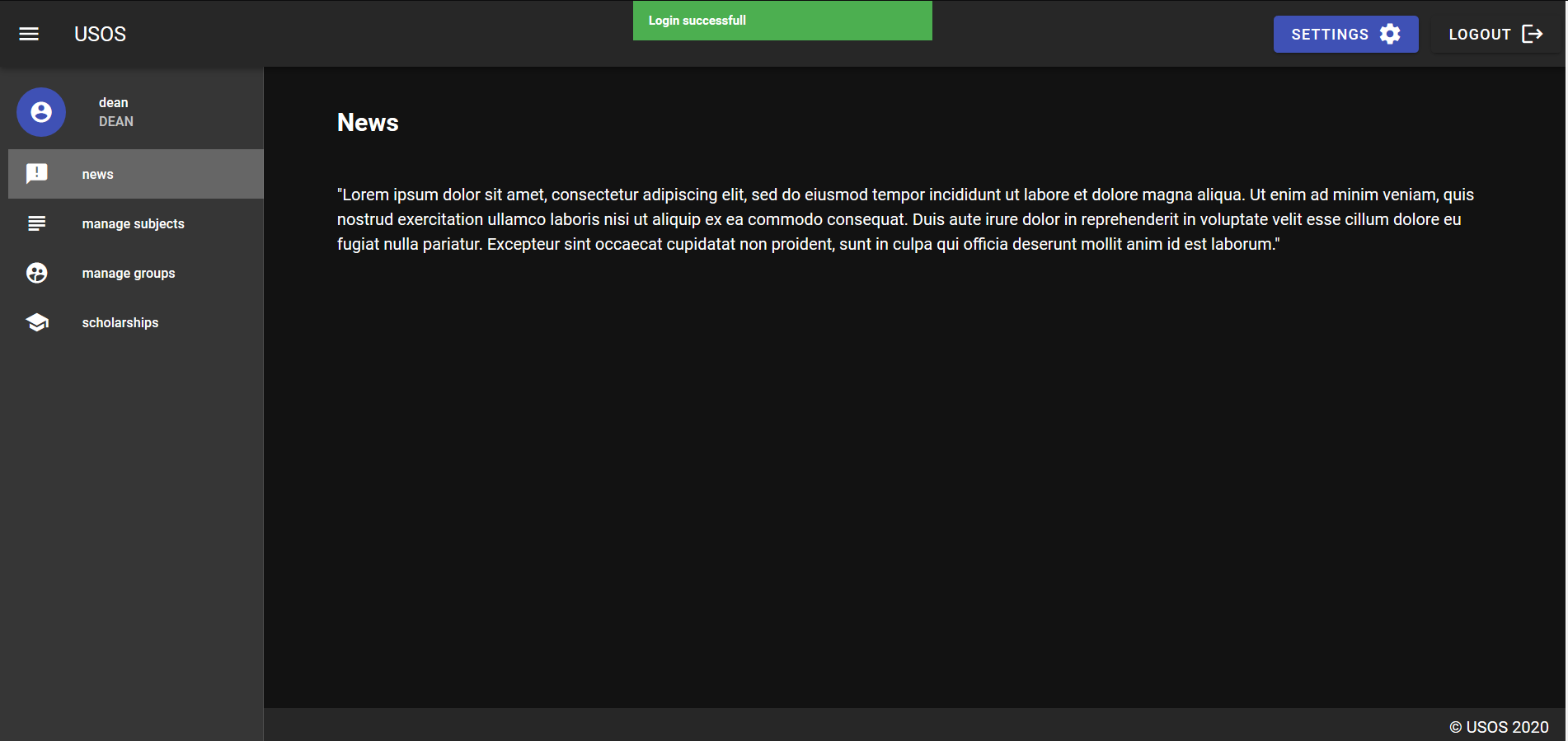
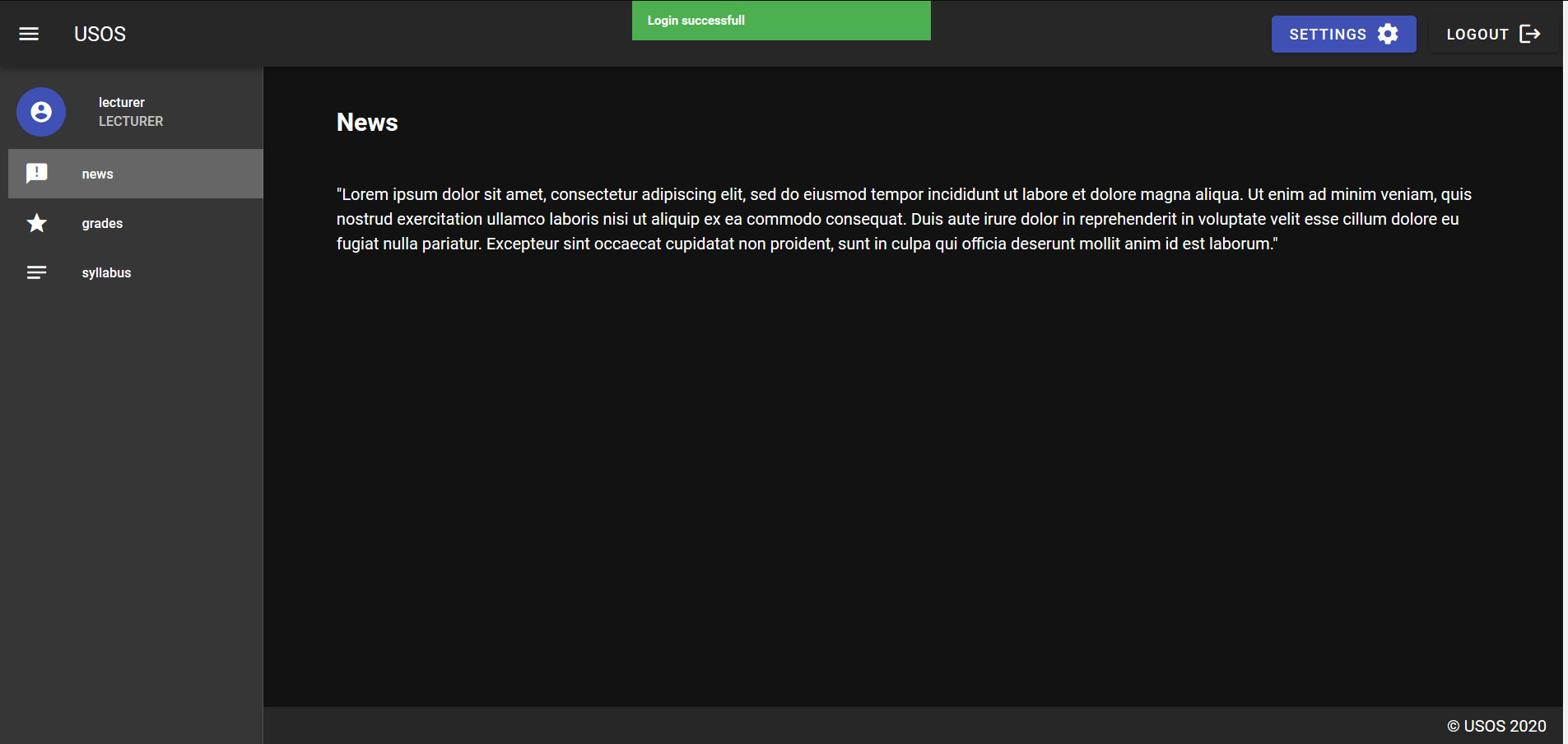
1. Przedstawienie aplikacji:
2. Ekran logowania do systemu

Logowanie jako student, rektor, dziekan oraz wykładowca:



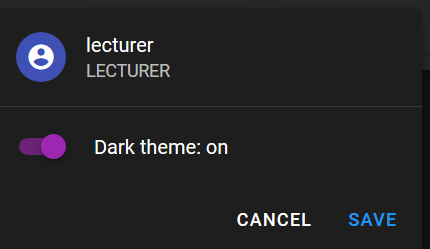
Kliknięcie przycisku „Log in” spowoduje zalogowanie się na określonego użytkownika i przejście do strony głównej

1. Strona główna i widok strony z poziomu poszczególnych ról

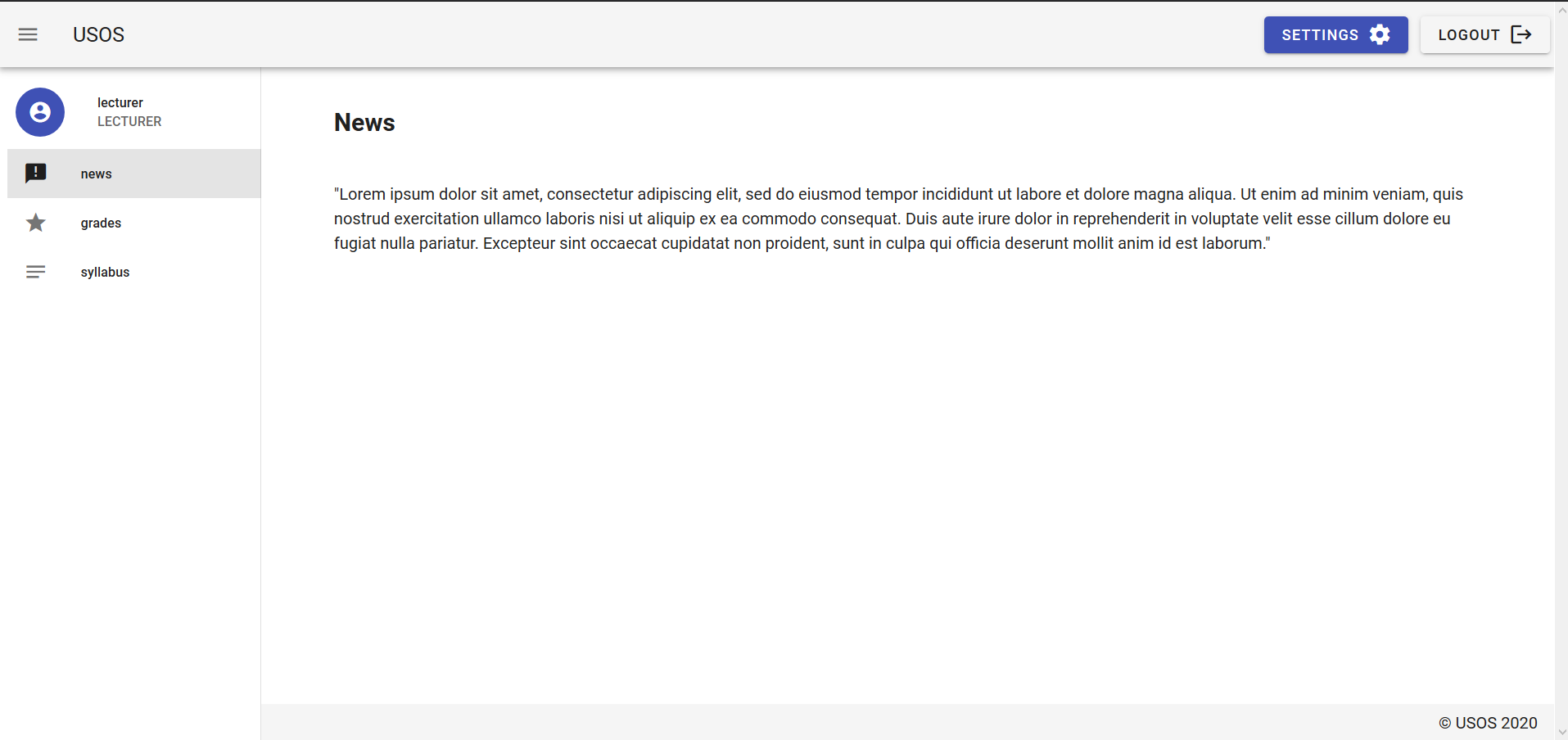
* Student
* Rektor
* Dziekan
* Wykładowca

O pomyślnym zalogowaniu użytkownik jest informowany za pomocą wysuwanej belki z informacją na zielonym tle.

Każdy użytkownik ma również możliwość zmiany koloru tła poprzez kliknięcie w przycisk „Settings” i zmianę suwaka „Dark theme: on” z poziomu on na off oraz nacisnąć przycisk „SAVE”.

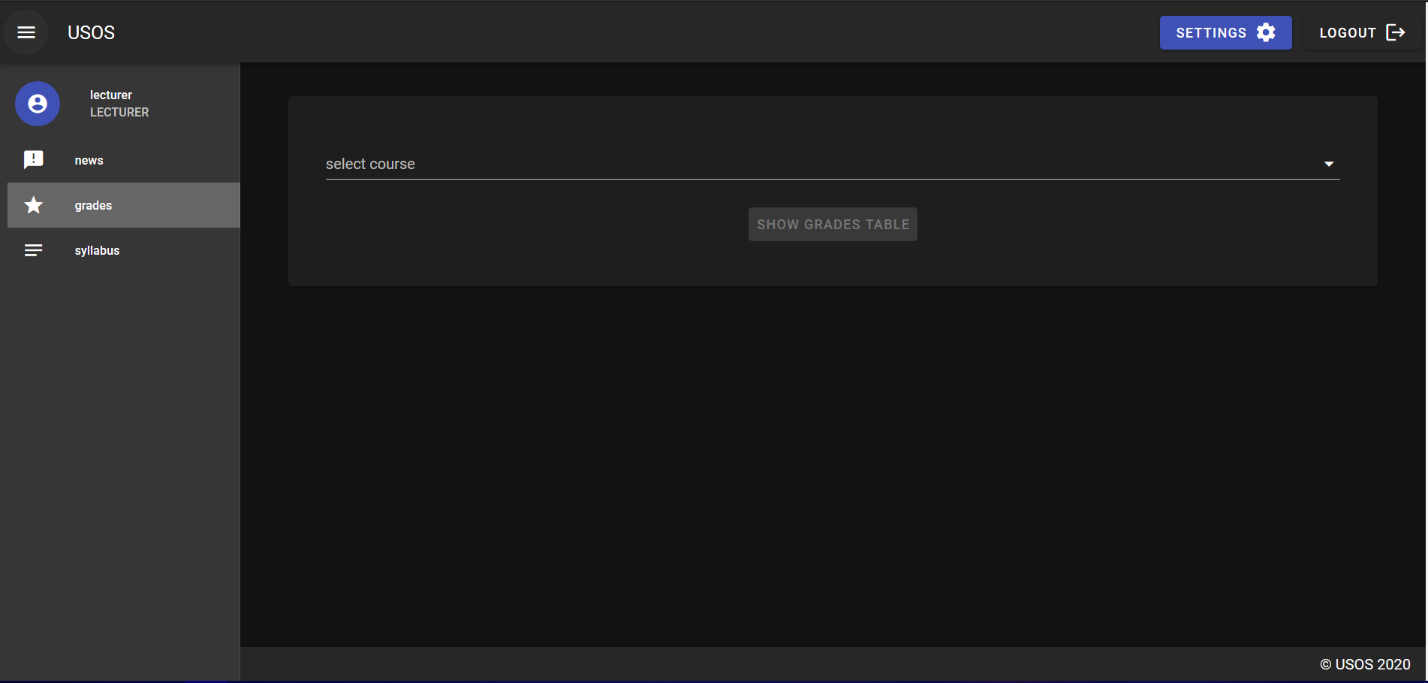


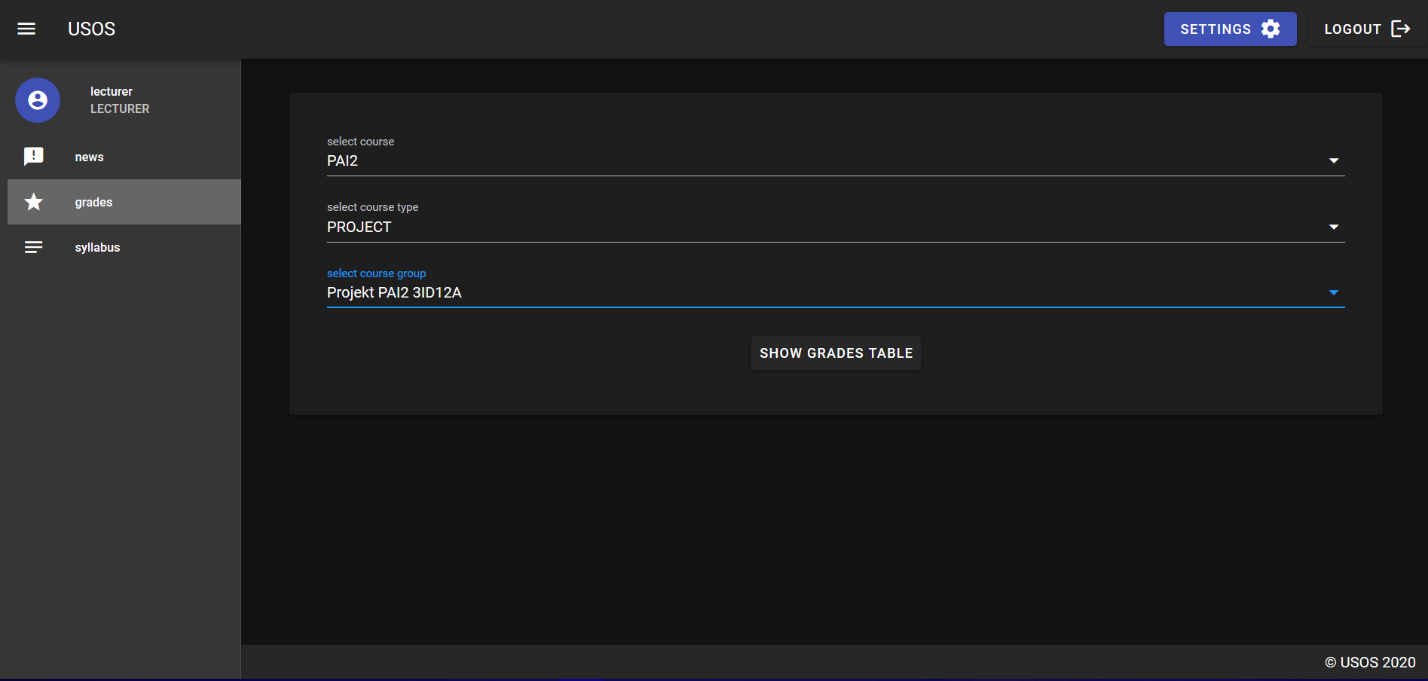
Widok systemu po zmianie koloru tła:



1. Omówienie trzech warstw aplikacji na przykładzie dodania nowej oceny dla studenta z poziomu wykładowcy

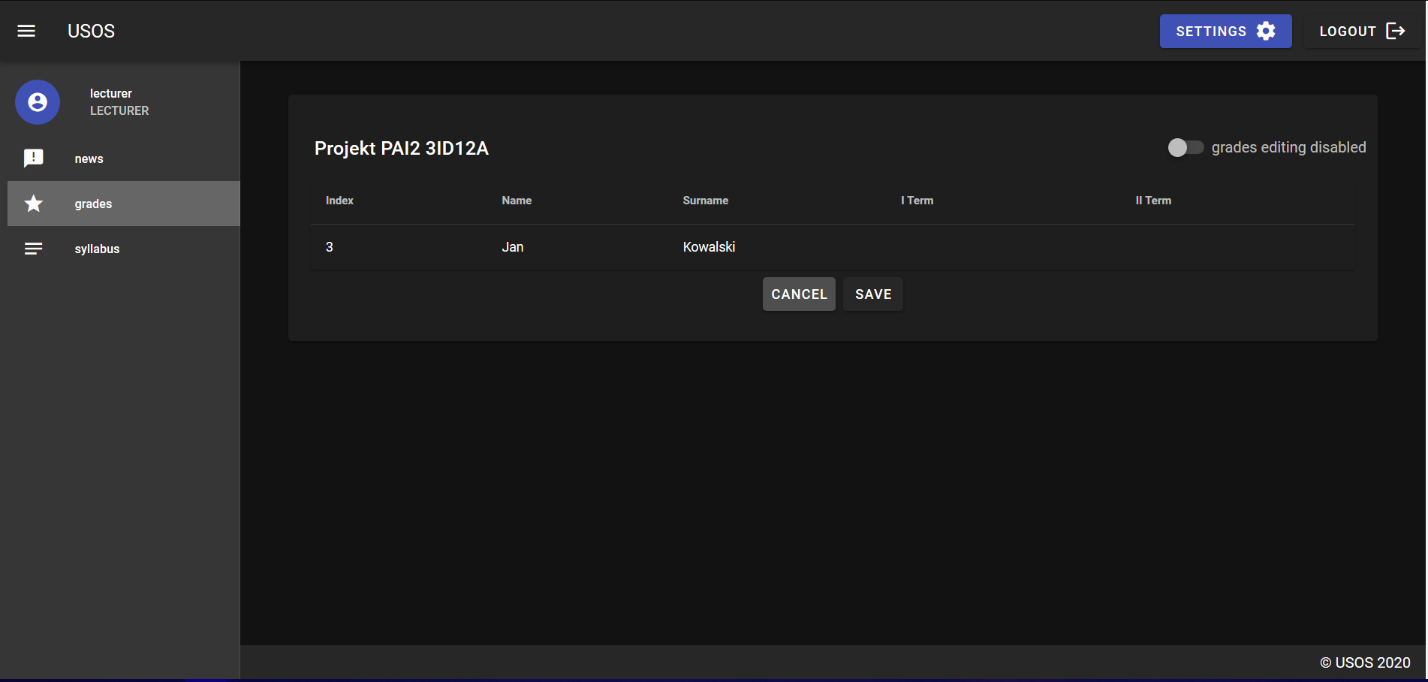
* Warstwa prezentacji:

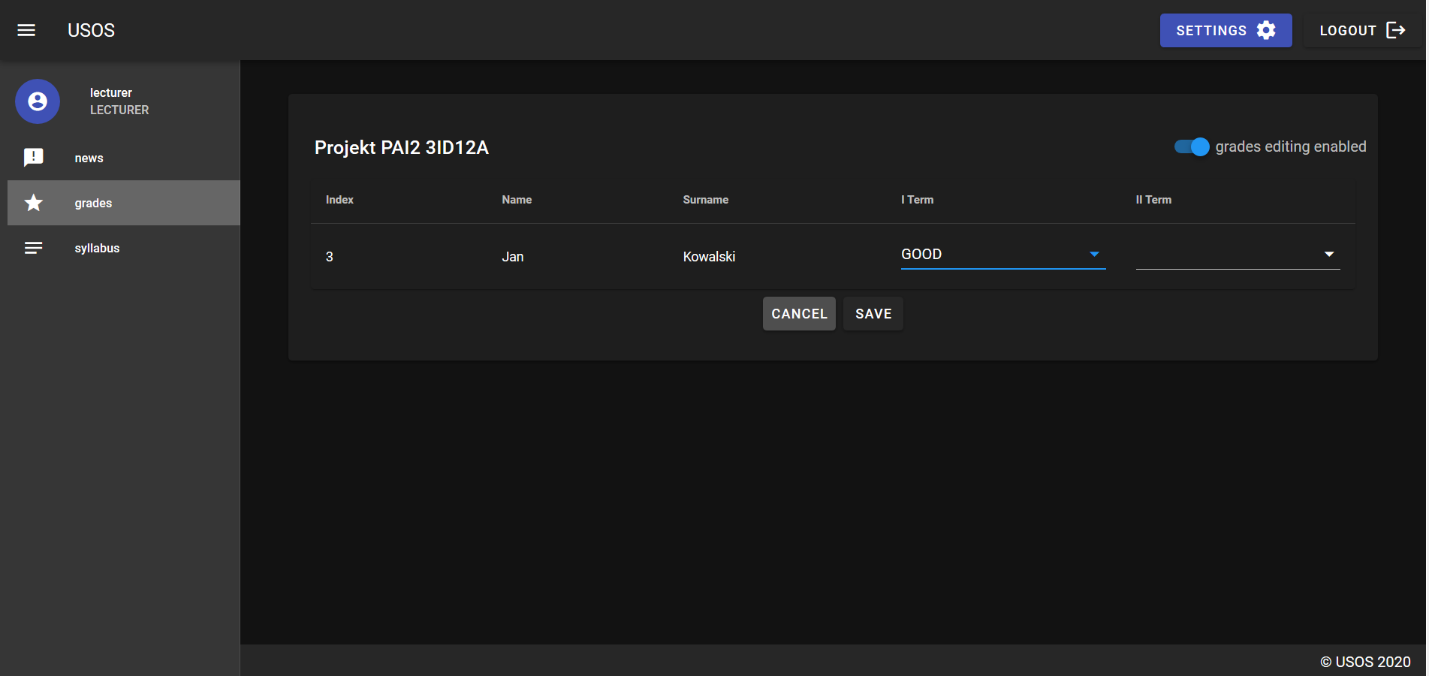
Wygląd strony przed ustawieniem kursu, typu zajęć oraz grupy zajęciowej:

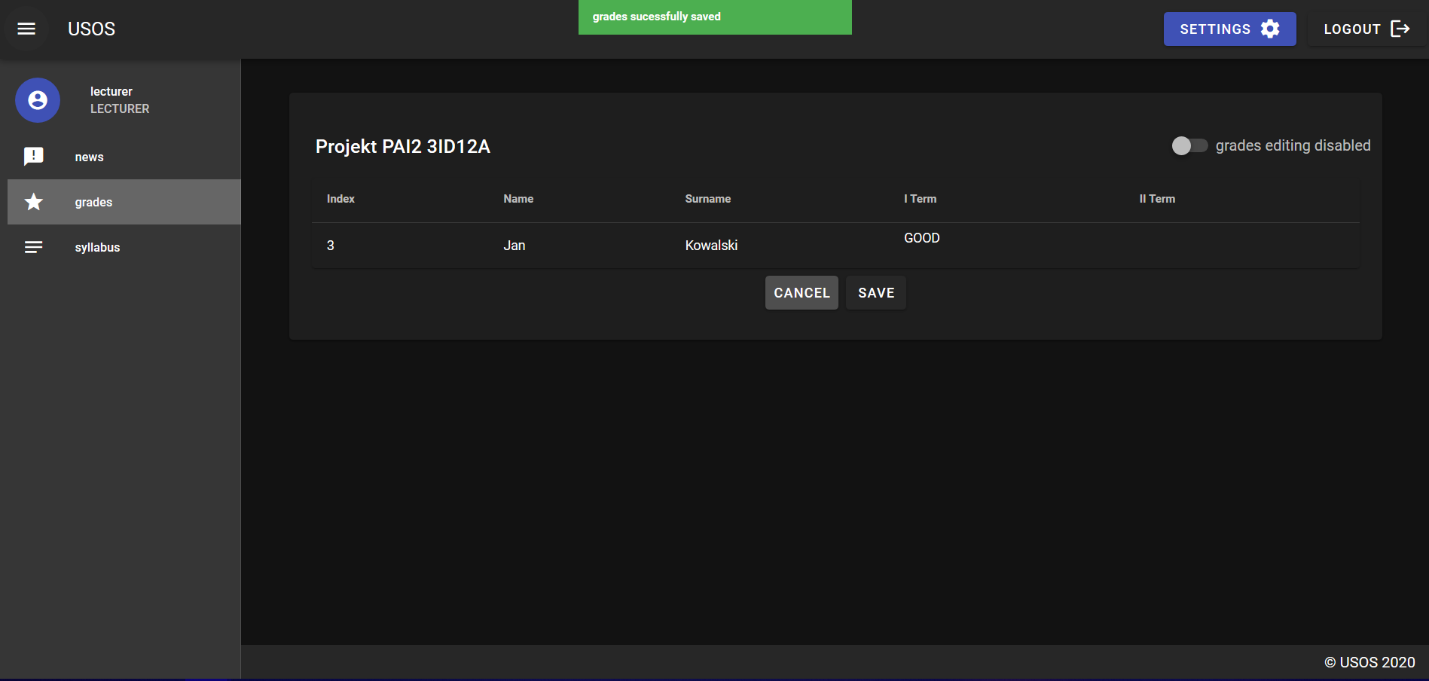
Wygląd strony po uzupełnieniu pól:

Kod odpowiadający za ten fragment strony:

|  |
| --- |
| LecturerGradesPanel.vue |
| <template>  <v-container>  <v-form @submit.prevent="showGradesTable()">  <v-card class="pa-10">  <v-select v-model="course" :items="courses" label="select course"></v-select>  <v-select v-if="course" v-model="courseType" :items="courseTypes" label="select course type"></v-select>  <v-select v-if="courseType" v-model="courseGroup" :items="courseGroups" label="select course group"></v-select>  <v-card-actions class="justify-center">  <v-btn type="submit" :disabled="!course || !courseType || !courseGroup">Show grades table</v-btn>  </v-card-actions>  </v-card>  </v-form>  </v-container> </template>  <script> import { ***mapActions***, ***mapGetters*** } from "vuex";  export default {  data() {  return {  courses: [],  courseTypes: [],  courseGroups: [],  course: "",  courseType: "",  courseGroup: ""  };  },  computed: ***mapGetters***(["courseData"]),  methods: {  ...***mapActions***(["getCourseData"]),  showGradesTable() {  this.$router.push(`/lecturer/grades/${this.courseGroup}`);  }  },  async created() {  await this.getCourseData();  this.courses = this.courseData.map(course => course.course.name);  },  watch: {  course() {  ***console***.log(this.course)  if (this.course)  this.courseTypes = this.courseData  .filter(course => course.course.name === this.course)  .map(course => course.courseType);  },  courseType() {  if (this.courseType)  this.courseGroups = this.courseData  .filter(course => course.course.name === this.course && course.courseType === this.courseType)  .map(course => course.name);  }  } }; </script>  <style scoped> </style> |

Strona po kliknięciu “Show Grades Table”

Widok dodania nowej oceny dla studenta

Zapisanie oceny dla studenta

|  |
| --- |
| GradesTable.vue |
| <template>  <v-container>  <v-card class="pa-6">  <v-row>  <v-card-title>{{this.courseFullName}}</v-card-title>  <v-spacer></v-spacer>  <v-switch v-model="edit" :label="'grades editing ' + (edit ? 'enabled' : 'disabled')"></v-switch>  </v-row>  <v-data-table  hide-default-footer  :dark="$vuetify.theme.dark"  :headers="headers"  :items="students"  class="elevation-1"  >  <template v-slot:item.firstTerm="{item}">  <p v-if="!edit">{{item.firstTerm}}</p>  <v-select v-model="item.firstTerm" :items="grades" v-else></v-select>  </template>  <template v-slot:item.secondTerm="{item}">  <p v-if="!edit">{{item.secondTerm}}</p>  <v-select v-model="item.secondTerm" :items="grades" v-else></v-select>  </template>  </v-data-table>  <v-card-actions class="justify-center">  <v-btn to="/lecturer/grades">cancel</v-btn>  <v-btn @click="saveGrades">save</v-btn>  </v-card-actions>  </v-card>  </v-container> </template>  <script> /\* eslint-disable \*/ import { ***mapGetters***, ***mapActions*** } from "vuex"; export default {  data() {  return {  headers: [  { text: "Index", value: "id" },  { text: "Name", value: "firstName" },  { text: "Surname", value: "lastName" },  { text: "I Term", value: "firstTerm", width: "250" },  { text: "II Term", value: "secondTerm", width: "250" }  ],  students: [],  studentsConst: [],  edit: false,  grades: [],  courseFullName: "",  courseId: ""  };  },  computed: ***mapGetters***(["courseData", "gradesValues"]),  methods: {  ...***mapActions***(["getCourseData", "addGrade", "getGradesValues"]),  async saveGrades() {  const firstTermStudents = [],  secondTermStudents = [];  this.studentsConst.forEach(studentConst => {  this.students.forEach(student => {  if (studentConst.firstTerm !== student.firstTerm)  firstTermStudents.push(student);  if (studentConst.secondTerm !== student.secondTerm)  secondTermStudents.push(student);  });  });   const params = {  description: "",  courseGroupId: this.courseId,  createdById: ***localStorage***.getItem("userId")  };   if (!firstTermStudents.length && !secondTermStudents.length)  this.$notify({  group: "foo",  title: "grades not changed"  });  else {  if (firstTermStudents.length)  await ***Promise***.all(  firstTermStudents.map(student => {  params.examDateType = "FIRST";  params.assignedUserId = student.id;  params.value = student.firstTerm;  return this.addGrade(params);  })  );   if (secondTermStudents.length)  await ***Promise***.all(  secondTermStudents.map(student => {  params.examDateType = "SECOND";  params.assignedUserId = student.id;  params.value = student.secondTerm;  return this.addGrade(params);  })  );  //UNCOMMENT THIS WHEN GRADES WILL BE RETRIEVED FROM BACKEND  // await this.getCourseData();  // this.retrieveData();  //   //DELETE THIS WHEN GRADES WILL BE RETRIEVED FROM BACKEND  this.edit =false  //    this.$notify({  group: "foo",  type: "success",  title: "grades sucessfully saved"  });  }  },  retrieveData() {  this.courseFullName = this.$route.params.course;  const course = this.courseData.find(  course => course.name === this.courseFullName  );  this.students = course.students;  //DELETE THIS WHEN GRADES WILL BE RETRIEVED FROM BACKEND  this.students.forEach(student => {  student.firstTerm = "";  student.secondTerm = "";  });  //  this.studentsConst = ***JSON***.parse(***JSON***.stringify(this.students));  this.courseId = course.id;  this.edit=false;  }  },  async created() {  await this.getCourseData();  this.retrieveData();  await this.getGradesValues();  this.grades = this.gradesValues;  } }; </script>  <style scoped> </style> |

* Warstwa logiki

|  |
| --- |
| GradeController.java |
| package org.fibi.usos.controller.course.grade;  import org.fibi.usos.annotation.RequireRole; import org.fibi.usos.dto.grade.GradeChangeRequestDto; import org.fibi.usos.dto.grade.GradeDto; import org.fibi.usos.model.course.grade.GradeModel; import org.fibi.usos.model.user.UserRole; import org.fibi.usos.service.course.grade.GradeService; import org.springframework.context.annotation.Role; import org.springframework.http.ResponseEntity; import org.springframework.security.access.annotation.Secured; import org.springframework.security.access.prepost.PreAuthorize; import org.springframework.web.bind.annotation.\*;  import javax.annotation.security.RolesAllowed; import java.util.Collection; import java.util.LinkedList; import java.util.List; import java.util.Optional;  @RestController @RequestMapping("/grade") public class GradeController {   private GradeService gradeService;   public GradeController(GradeService gradeService) {  this.gradeService = gradeService;  }    @RequireRole({UserRole.*LECTURER*,UserRole.*DEAN*,UserRole.*RECTOR*})  @PostMapping("/add")  public ResponseEntity<GradeDto> createGrade(@RequestBody GradeChangeRequestDto dto) {  GradeDto res = new GradeDto();  Optional<GradeModel> model = gradeService.createOrUpdate(dto);  if(model.isPresent()){  res = model.get().MapToDto();  }  return ResponseEntity.*ok*(res);  }   @RequireRole({UserRole.*STUDENT*})  @GetMapping("/student/{id}")  public ResponseEntity<Collection<GradeDto>> getGradeByAssignedUserId(@PathVariable(name = "id")Long assignedUserId){  List<GradeDto> responseDtos = new LinkedList<>();  Optional<Collection<GradeModel>> models = gradeService.findGradesByAssignedUserId(assignedUserId);  models.ifPresent( model -> model.forEach(m -> responseDtos.add(m.MapToDto())));  return ResponseEntity.*ok*(responseDtos);  }   @RequireRole({UserRole.*DEAN*,UserRole.*LECTURER*,UserRole.*RECTOR*})  @GetMapping("/lecturer/{id}")  public ResponseEntity<Collection<GradeDto>> getGradeByLecturerUserId(@PathVariable(name = "id")Long lecturerUserId){  List<GradeDto> responseDtos = new LinkedList<>();  Optional<Collection<GradeModel>> models = gradeService.findGradesByCreatedByUserId(lecturerUserId);  models.ifPresent( model -> model.forEach(m -> responseDtos.add(m.MapToDto())));  return ResponseEntity.*ok*(responseDtos);  } } |

* Warstwa dostępu do danych

|  |
| --- |
| GradeModel.java |
| package org.fibi.usos.controller.course.grade;  import org.fibi.usos.annotation.RequireRole; import org.fibi.usos.dto.grade.GradeChangeRequestDto; import org.fibi.usos.dto.grade.GradeDto; import org.fibi.usos.model.course.grade.GradeModel; import org.fibi.usos.model.user.UserRole; import org.fibi.usos.service.course.grade.GradeService; import org.springframework.context.annotation.Role; import org.springframework.http.ResponseEntity; import org.springframework.security.access.annotation.Secured; import org.springframework.security.access.prepost.PreAuthorize; import org.springframework.web.bind.annotation.\*;  import javax.annotation.security.RolesAllowed; import java.util.Collection; import java.util.LinkedList; import java.util.List; import java.util.Optional;  @RestController @RequestMapping("/grade") public class GradeController {   private GradeService gradeService;   public GradeController(GradeService gradeService) {  this.gradeService = gradeService;  }    @RequireRole({UserRole.*LECTURER*,UserRole.*DEAN*,UserRole.*RECTOR*})  @PostMapping("/add")  public ResponseEntity<GradeDto> createGrade(@RequestBody GradeChangeRequestDto dto) {  GradeDto res = new GradeDto();  Optional<GradeModel> model = gradeService.createOrUpdate(dto);  if(model.isPresent()){  res = model.get().MapToDto();  }  return ResponseEntity.*ok*(res);  }   @RequireRole({UserRole.*STUDENT*})  @GetMapping("/student/{id}")  public ResponseEntity<Collection<GradeDto>> getGradeByAssignedUserId(@PathVariable(name = "id")Long assignedUserId){  List<GradeDto> responseDtos = new LinkedList<>();  Optional<Collection<GradeModel>> models = gradeService.findGradesByAssignedUserId(assignedUserId);  models.ifPresent( model -> model.forEach(m -> responseDtos.add(m.MapToDto())));  return ResponseEntity.*ok*(responseDtos);  }   @RequireRole({UserRole.*DEAN*,UserRole.*LECTURER*,UserRole.*RECTOR*})  @GetMapping("/lecturer/{id}")  public ResponseEntity<Collection<GradeDto>> getGradeByLecturerUserId(@PathVariable(name = "id")Long lecturerUserId){  List<GradeDto> responseDtos = new LinkedList<>();  Optional<Collection<GradeModel>> models = gradeService.findGradesByCreatedByUserId(lecturerUserId);  models.ifPresent( model -> model.forEach(m -> responseDtos.add(m.MapToDto())));  return ResponseEntity.*ok*(responseDtos);  } } |

|  |
| --- |
| GradeValueType.java |
| package org.fibi.usos.model.course.grade;  import org.fibi.usos.annotation.RegisterEnum;  @RegisterEnum public enum GradeValueType {  *UNCLASSIFIED*,  *ALLOWING*,  *SUFFICIENT*,  *GOOD*,  *VERY\_GOOD*,  *EXCELLENT* } |

1. Testy jednostkowe

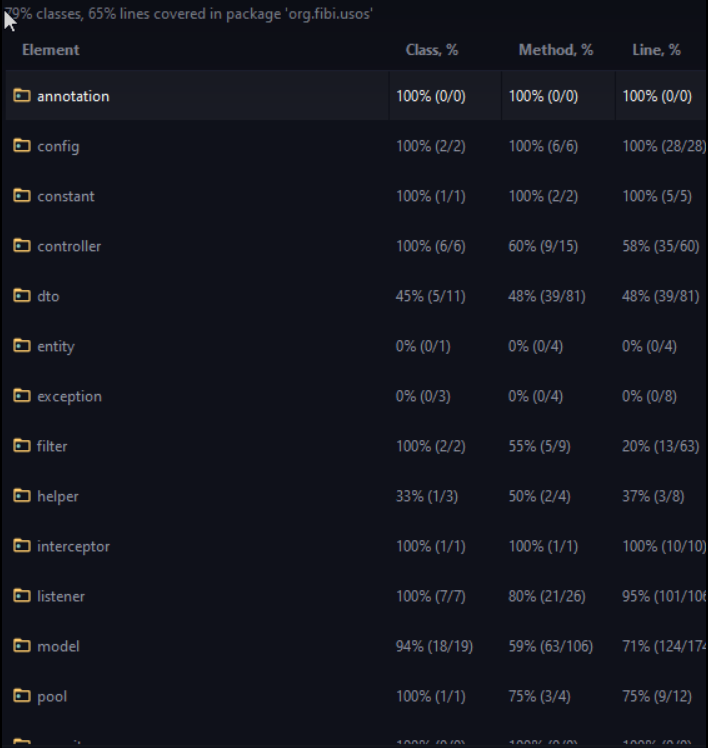
|  |
| --- |
| GradeControllerTests.java |
| package org.fibi.usos.controller.course.grade;  import com.fasterxml.jackson.databind.ObjectMapper; import org.fibi.usos.App; import org.fibi.usos.dto.grade.GradeChangeRequestDto; import org.fibi.usos.model.course.grade.GradeValueType; import org.fibi.usos.model.exam.ExamDateType; import org.junit.Before; import org.junit.Test; import org.junit.runner.RunWith; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc; import org.springframework.boot.test.context.SpringBootTest; import org.springframework.http.MediaType; import org.springframework.security.test.context.support.WithUserDetails; import org.springframework.test.context.junit4.SpringRunner; import org.springframework.test.web.servlet.MockMvc;  import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*get*; import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*post*; import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;   @RunWith(SpringRunner.class) @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.*MOCK*, classes = {App.class}) @AutoConfigureMockMvc public class GradeControllerTests {  @Autowired  private MockMvc mockMvc;  private ObjectMapper objectMapper;   @Before  public void before(){  objectMapper = new ObjectMapper();  }    @Test  @WithUserDetails("lecturer")  public void addNewGradeWithValidDtoReturnModelAndUserWithAccess() throws Exception{  GradeChangeRequestDto dto = new GradeChangeRequestDto();  dto.setDescription("Test");  dto.setValue(GradeValueType.*EXCELLENT*);  dto.setExamDateType(ExamDateType.*FIRST*);  dto.setCourseGroupId(9);  dto.setAssignedUserId(3);  dto.setCreatedById(5);  mockMvc.perform(*post*("/grade/add")  .contentType(MediaType.*APPLICATION\_JSON*).content(objectMapper.writeValueAsString(dto)))  .andExpect(*status*().isOk());  }    @Test  @WithUserDetails("student")  public void addNewGradeWithValidDtoReturnModelAndUserWithoutAccess() throws Exception{  GradeChangeRequestDto dto = new GradeChangeRequestDto();  dto.setDescription("Test");  dto.setValue(GradeValueType.*EXCELLENT*);  dto.setExamDateType(ExamDateType.*FIRST*);  dto.setCourseGroupId(9);  dto.setAssignedUserId(3);  dto.setCreatedById(5);  mockMvc.perform(*post*("/grade/add")  .contentType(MediaType.*APPLICATION\_JSON*).content(objectMapper.writeValueAsString(dto)))  .andExpect(*status*().isUnauthorized());  }   @Test  @WithUserDetails("student")  public void getGradesForStudentWithValidRole() throws Exception{  mockMvc.perform(*get*("/grade/student/3"))  .andExpect(*status*().isOk())  .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  .andExpect(*jsonPath*("$").isArray());  }    @Test  @WithUserDetails("student")  public void getGradesForStudentWithValidRoleButNoGrades() throws Exception{  mockMvc.perform(*get*("/grade/student/666"))  .andExpect(*status*().isOk())  .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  .andExpect(*jsonPath*("$").isEmpty());  }    @Test  @WithUserDetails("lecturer")  public void getGradesForStudentWithNotValidRole() throws Exception{  mockMvc.perform(*get*("/grade/student/3"))  .andExpect(*status*().isUnauthorized());  }   @Test  @WithUserDetails("lecturer")  public void getGradesForLecturerWithValidRole() throws Exception{  mockMvc.perform(*get*("/grade/lecturer/5"))  .andExpect(*status*().isOk())  .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  .andExpect(*jsonPath*("$").isArray());  }    @Test  @WithUserDetails("lecturer")  public void getGradesForLecturerWithValidRoleButNoGrades() throws Exception{  mockMvc.perform(*get*("/grade/lecturer/666"))  .andExpect(*status*().isOk())  .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  .andExpect(*jsonPath*("$").isEmpty());  }    @Test  @WithUserDetails("student")  public void getGradesForLecturerWithNotValidRole() throws Exception{  mockMvc.perform(*get*("/grade/lecturer/5"))  .andExpect(*status*().isUnauthorized());  } } |

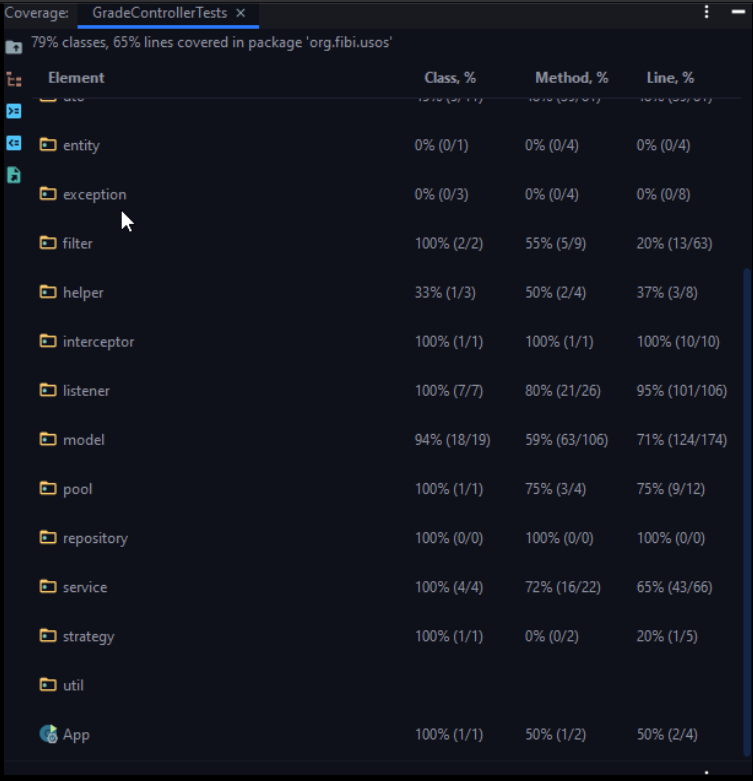
|  |
| --- |
| UtilControllerTests.java |
| package org.fibi.usos.controller;   import org.fibi.usos.App; import org.fibi.usos.model.exam.ExamDateType; import org.hamcrest.Matchers; import org.junit.jupiter.api.Test; import org.junit.runner.RunWith; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc; import org.springframework.boot.test.context.SpringBootTest; import org.springframework.http.MediaType; import org.springframework.test.context.junit4.SpringRunner; import org.springframework.test.web.servlet.MockMvc;  import java.lang.reflect.Field; import java.util.Arrays;  import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*get*; import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;  @RunWith(SpringRunner.class) @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.*MOCK*, classes = {App.class}) @AutoConfigureMockMvc public class UtilControllerTests {   @Autowired  private MockMvc mockMvc;   @Test  public void enumUtilWithValidNameReturnCorrectValues() throws Exception {  mockMvc.perform(*get*("/util/enum/ExamDateType/values")  .accept(MediaType.*APPLICATION\_JSON*))  .andExpect(*status*().isOk())  .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  .andExpect(*jsonPath*("$.message").value("Enum found."))  .andExpect(*jsonPath*("$.enumValues").isArray())  .andExpect(*jsonPath*("$.enumValues", Matchers.*containsInAnyOrder*(Arrays  .*stream*(ExamDateType.class.getFields())  .map(Field::getName)  .distinct()  .toArray())));  }   @Test  public void enumUtilWithNotValidNameReturn403() throws Exception {  mockMvc.perform(*get*("/util/enum/Test/values"))  .andExpect(*status*().isNoContent());  } } |

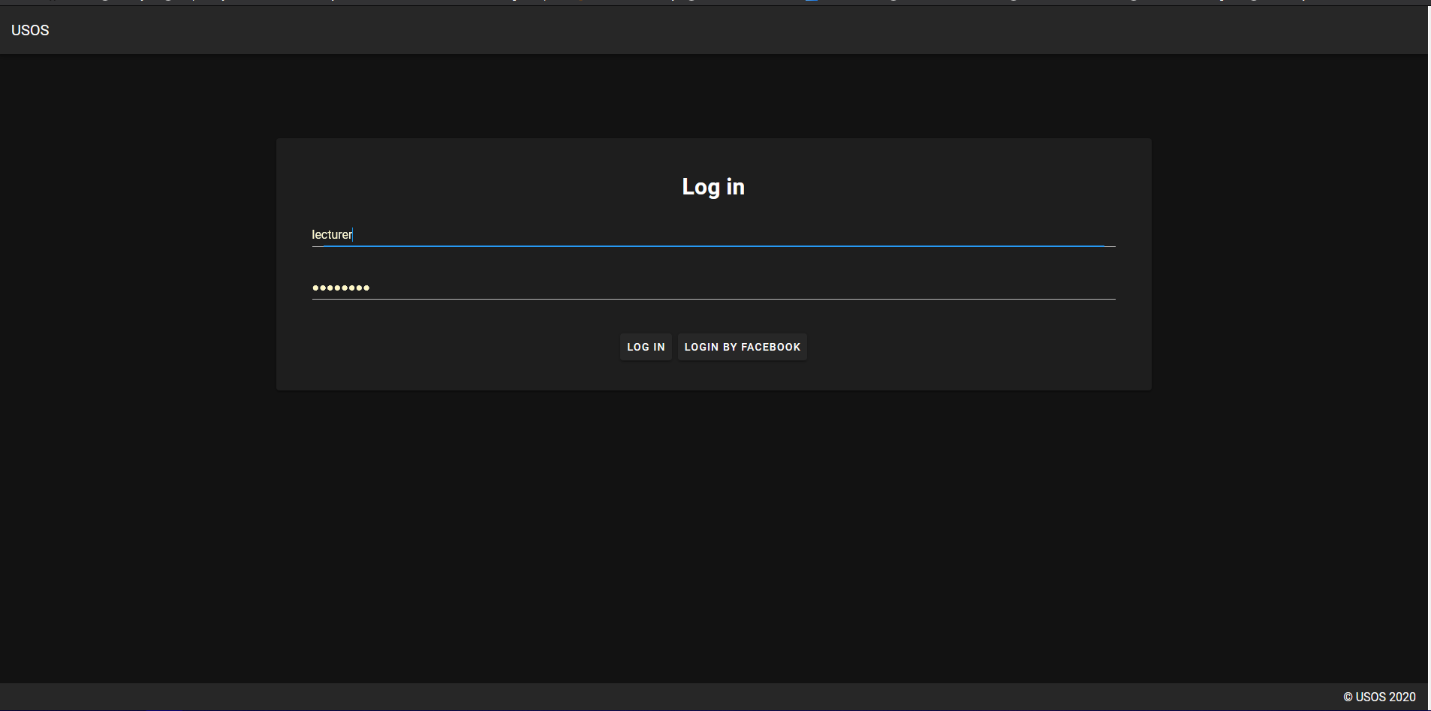
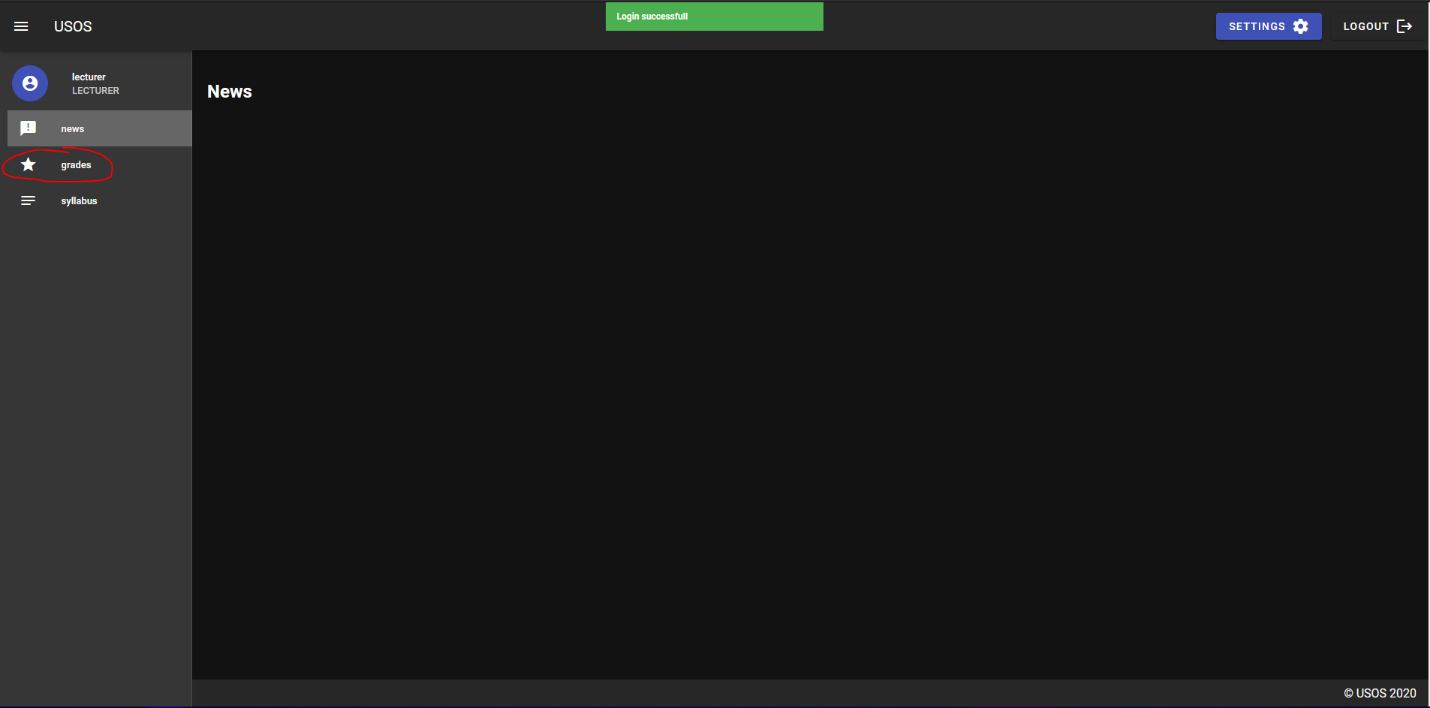
|  |
| --- |
| UserControllerTests.java |
| package org.fibi.usos.controller.user;  import org.fibi.usos.App; import org.junit.Test; import org.junit.runner.RunWith; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc; import org.springframework.boot.test.context.SpringBootTest; import org.springframework.http.MediaType; import org.springframework.security.test.context.support.WithUserDetails; import org.springframework.test.context.junit4.SpringRunner; import org.springframework.test.web.servlet.MockMvc;  import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*get*; import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;  @RunWith(SpringRunner.class) @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.*MOCK*, classes = {App.class}) @AutoConfigureMockMvc public class UserControllerTests {   @Autowired  private MockMvc mockMvc;    @Test  @WithUserDetails("rector")  public void getAllUserUsingUserWithValidRole() throws Exception {  mockMvc.perform(*get*("/user/all"))  .andExpect(*status*().isOk())  .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  .andExpect(*jsonPath*("$").isArray())  .andExpect(*jsonPath*("$").isNotEmpty());  }   @Test  @WithUserDetails("student")  public void getAllUserUsingUserWithNotValidRole() throws Exception {  mockMvc.perform(*get*("/user/all"))  .andExpect(*status*().isUnauthorized());  }    @Test  @WithUserDetails("rector")  public void getAllUserInSpecificRoleUsingUserWithValidRole() throws Exception {  mockMvc.perform(*get*("/user/DEAN/all"))  .andExpect(*status*().isOk())  .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  .andExpect(*jsonPath*("$").isArray())  .andExpect(*jsonPath*("$").isNotEmpty())  .andExpect(*jsonPath*("$[\*].role").value("DEAN"));  }   @Test  @WithUserDetails("rector")  public void getAllUserInSpecificNotValidRoleUsingUserWithValidRole() throws Exception {  mockMvc.perform(*get*("/user/AAA/all"))  .andExpect(*status*().isBadRequest());  }   @Test  @WithUserDetails("student")  public void getAllUserInSpecificRoleUsingUserWithNotValidRole() throws Exception {  mockMvc.perform(*get*("/user/DEAN/all"))  .andExpect(*status*().isUnauthorized());  } } |

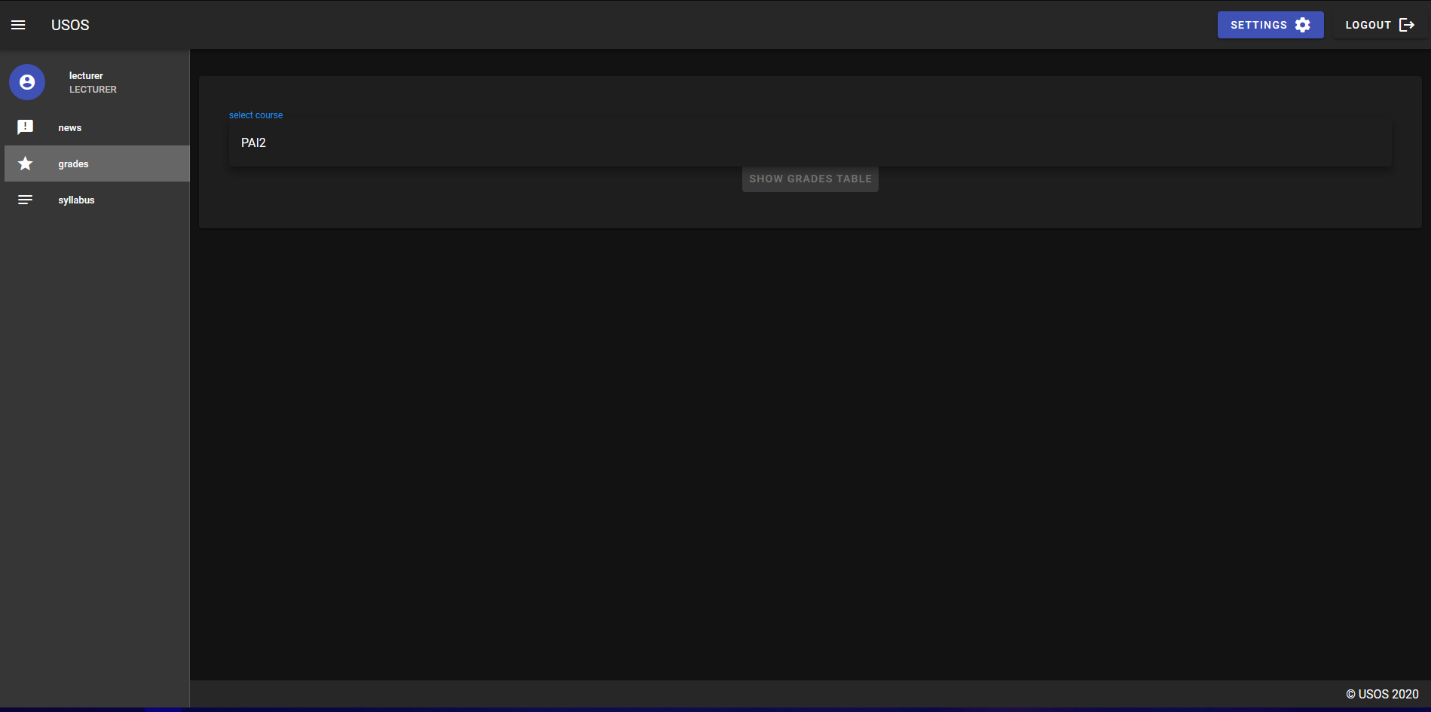
|  |
| --- |
| AppTests.java |
| package org.fibi.usos;  import org.junit.jupiter.api.Test; import org.springframework.boot.test.context.SpringBootTest;  @SpringBootTest class AppTests {   @Test  void contextLoads() {  }  } |

1. Sprawdzenie pokrycia kodu

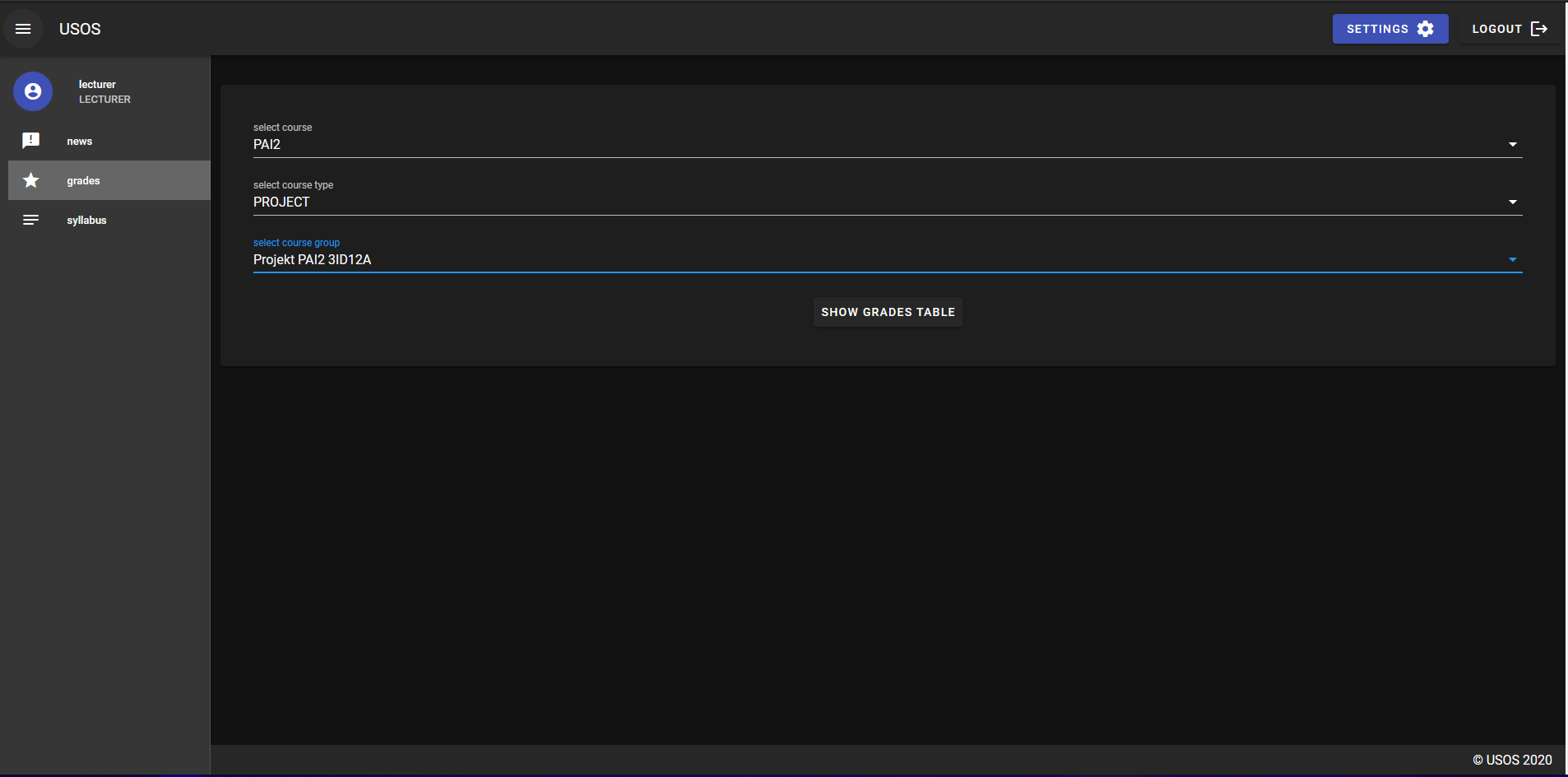




1. Instrukcja obsługi systemu na przykładzie zmiany oceny studenta
2. Należy zalogować się na panel wykładowcy:
3. Na stronie głównej w menu po prawej stronie należy wybrać opcję „grades”
4. Po ukazaniu się panelu „grades” należy wybrać nazwę przedmiotu, typ przedmiotu oraz grupę

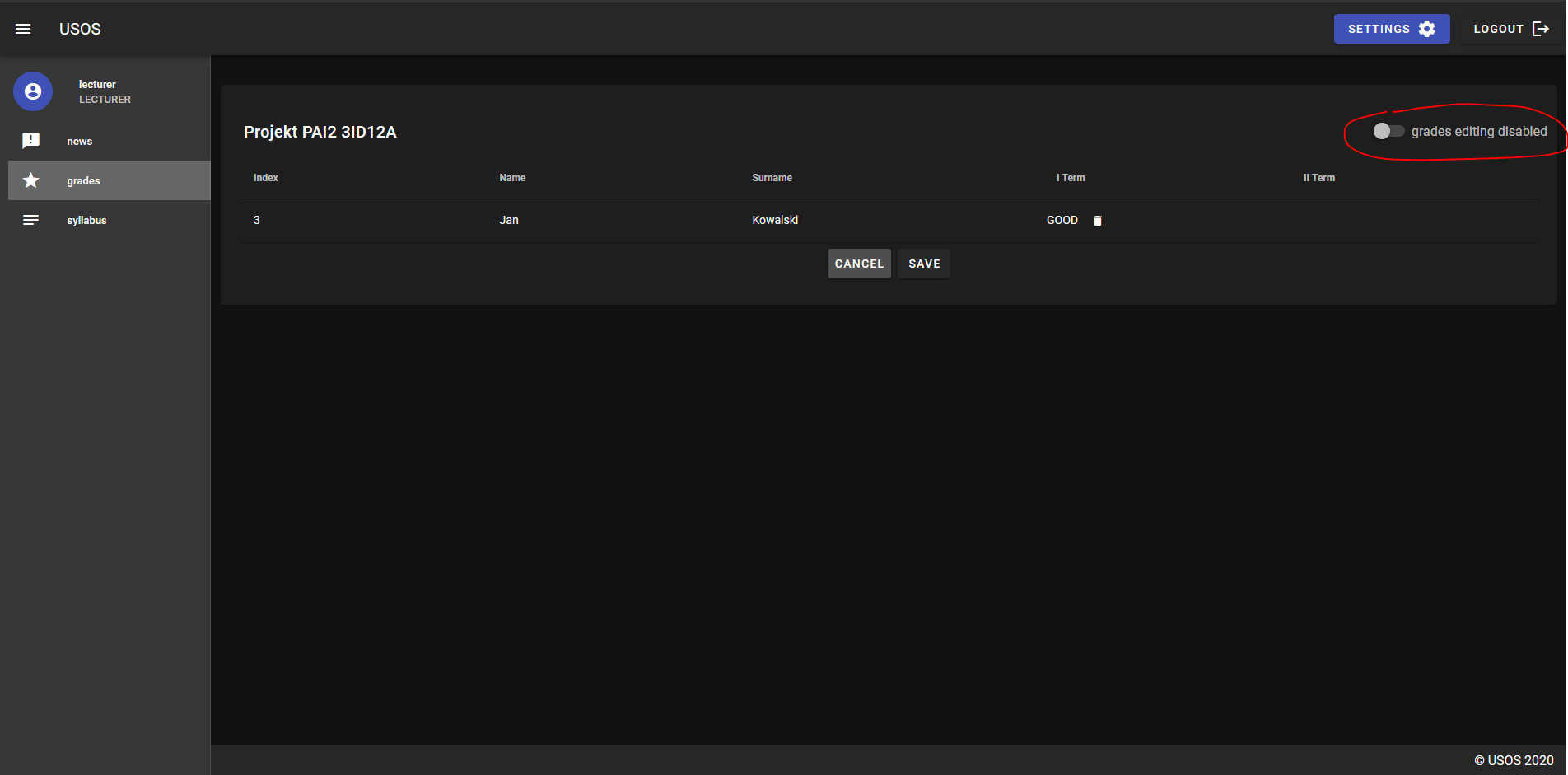


Uzupełnione pola do wyboru ocen dla danej grupy:

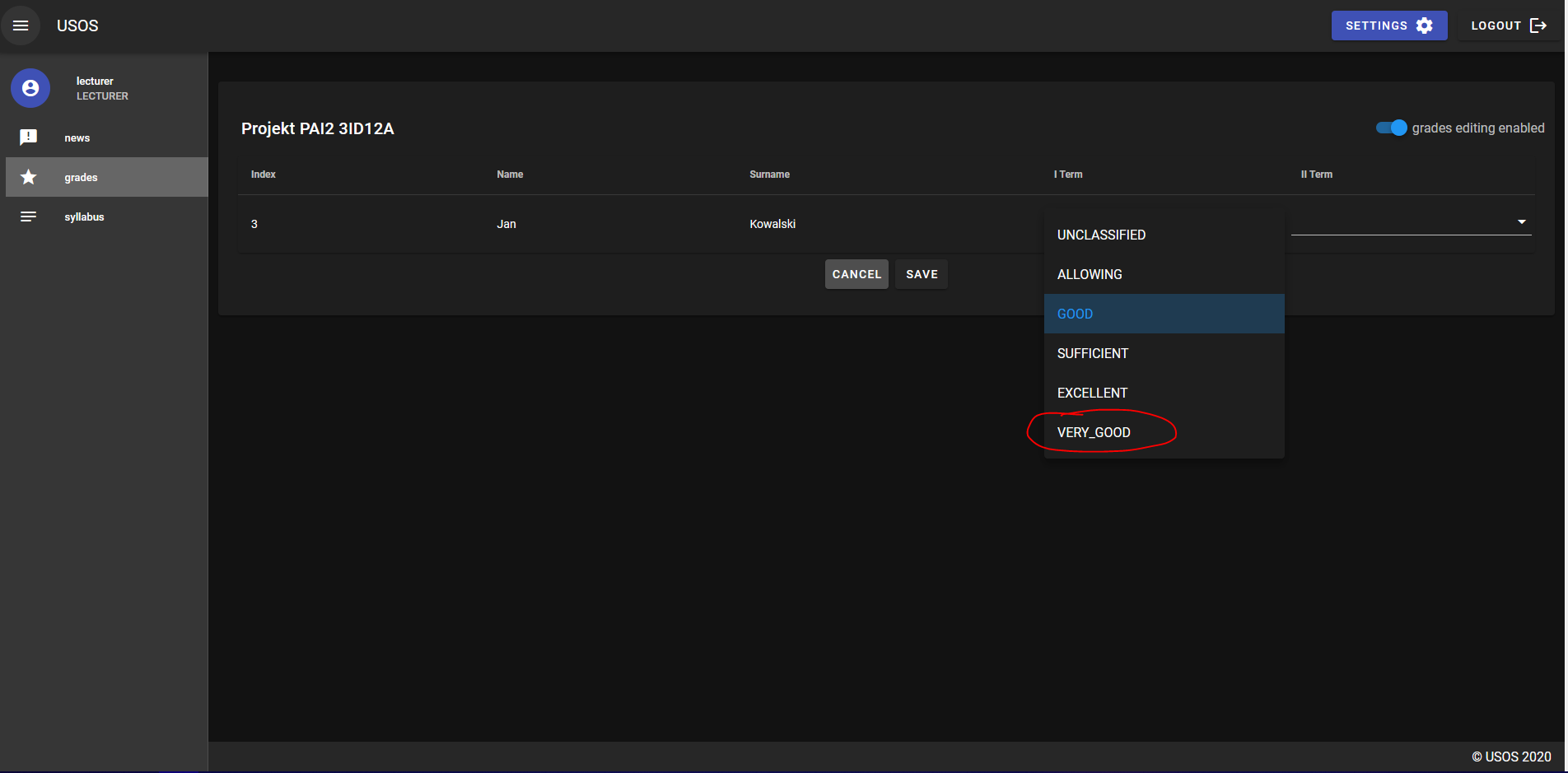


Następnie należy kliknąć przycisk „Show grades table”

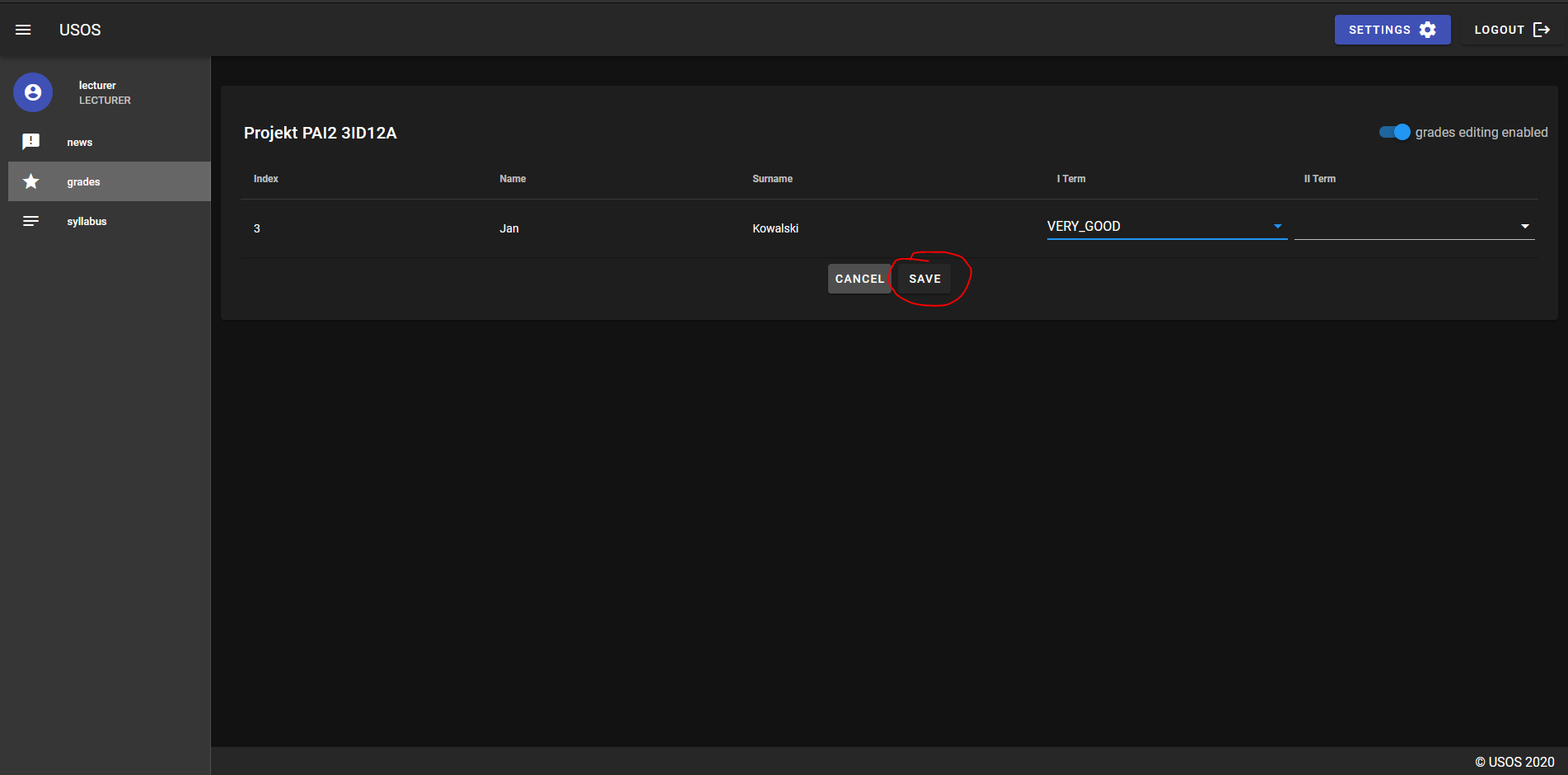
W tym momencie jest wyłącznie widoczny podgląd ocen, aby przystąpić do edycji ocen należy przesunąć suwak z pozycji „grades editing disabled” na pozycję „grades editing enabled”



Następnie można przystąpić do zmiany oceny – w tym przypadku z „good” na „very good” wybierając opcję z dropdown menu odpowiednio przy terminie – w tym przypadku pierwszym terminie.



Po wybraniu odpowiedniej oceny należy zapisać zmiany za pomocą przycisku „save” Zmienione pola zmienią swój kolor z białego na niebieski ( w przypadku czarnego tła aplikacji) do momentu gdy nie zostaną zapisane zmiany.



Po zapisaniu, zmieniona ocena powinna być widoczna oraz suwak powinien zmienić swoją pozycję na „grades editing disabled”. O fakcie pomyślnego zapisania zmian użytkownik będzie również poinformowany zielonym popupem w górnej części interfejsu z informacją „grades successfully saved”

