

Politechnika Poznańska  
Wydział Elektryczny  
Instytut Automatyki i Inżynierii Informatycznej



Konrad Gabara  
Adam Halicki

**Sprawdzanie obecności w laboratorium z wykorzystaniem legitymacji  
studenckiej**

Poznań, 2017

# Spis treści

## 1. Wstęp

- 1.1 Opis tematu
- 1.2 Uzasadnienie

## 2. Podział i harmonogram pracy

## 3. Funkcjonalności oferowane przez aplikację

- 3.1 Wyświetlanie listy obecności
- 3.2 Wyświetlanie studentów
- 3.3 Wyświetlanie prowadzących
- 3.4 Wyświetlanie przedmiotów
- 3.5 Dodawanie nowych list obecności
- 3.6 Dodawanie nowych studentów
- 3.7 Dodawanie nowych prowadzących
- 3.8 Dodawanie nowych przedmiotów
- 3.9 Usuwanie list obecności
- 3.10 Usuwanie studentów
- 3.11 Usuwanie prowadzących
- 3.12 Usuwanie przedmiotów
- 3.13 Uzupełnianie listy obecności za pomocą czytnika kart

## 4. Wybór technologii informatycznych

- 4.1 System Github
- 4.2 Język programowania Java
- 4.3 Spring Framework
- 4.4 AngularJS Framework
- 4.5 IntelliJ IDEA
- 4.6 MySQL
- 4.7 Node.js
- 4.8 TypeScript

## 5. Architektura

- 5.1 Model View Controller
- 5.2 Dependency Injection
- 5.3 Data Access Object
- 5.4 Baza danych stworzona dzięki podejściu code first za pomocą frameworka Hibernate

- przedmiot
- student
- lista obecności
- prowadzący
- lista studentów

#### 5.5 Restowe Kontrollery

- kontroler studentów
- kontroler prowadzących
- kontroler listy obecności
- kontroler przedmiotów

#### 5.6 Komponenty

- komponent studentów
- komponent prowadzących
- komponent listy obecności
- komponent przedmiotów

#### 5.7 Serwisy

- serwis studentów
- serwis prowadzących
- serwis listy obecności
- serwis przedmiotów

#### 5.8 Single Page Application

#### 5.9 Routing

### 6. Problemy i rozwiązania

#### 6.1 Czytanie danych z legitymacji studenckiej

### 7. Instrukcja użytkowania aplikacji

## 1. Wstęp

### 1.1 Opis tematu

- Stworzenie aplikacji do scentralizowanego zarządzania obecnością na laboratorium.
- Wykorzystanie czytników kart do sczytywania danych z legitymacji studenckich
- importowanie/wczytywanie listy studentów

### 1.2 Uzasadnienie

Podjęcie tej problematyki wydało nam się ciekawym wyzwaniem. Chcieliśmy nauczyć się czegoś nowego, oraz rozszerzyć obecnie posiadaną przez nas wiedzę.

## 2. Podział i harmonogram pracy

W rozdziale przedstawiony został podział pracy między autorami projektu. Harmonogram pracy widoczny jest w Tabeli 1

Tabela 1: Harmonogram pracy

<i>Lp.</i>	<i>Opis zadań</i>
1.	Przygotowanie zdalnego repozytorium
2.	Przygotowanie środowiska programistycznego
3.	Przygotowanie środowiska Java, oraz założenie wstępnego projektu
4.	Stworzenie bazy danych w MySQL
5.	Stworzenie aplikacji sczytującej dane z legitymacji studenckich za pomocą czytnika kart
6.	Stworzenie aplikacji webowej z użyciem springa
7.	Stworzenie funkcjonalności w aplikacji webowej
8.	Stworzenie frontendu za pomocą angulara
9.	Połączenie aplikacji webowej z desktopową
10.	Stworzenie dokumentacji

### **3. Funkcjonalności oferowane przez aplikację**

W rozdziale przedstawione zostały funkcjonalności oferowane przez aplikację. W kolejnych podsekcjach zamieszczone zostały szczegółowe opisy funkcjonalności.

#### **3.1 Wyświetlanie listy obecności**

możliwość wyświetlenia listy obecności i jej szczegółów z poziomu aplikacji webowej

#### **3.2 Wyświetlanie studentów**

możliwość wyświetlenia wszystkich studentów z poziomu aplikacji webowej

#### **3.3 Wyświetlanie prowadzących**

możliwość wyświetlenia wszystkich prowadzących z poziomu aplikacji webowej

#### **3.4 Wyświetlanie przedmiotów**

możliwość wyświetlenia wszystkich prowadzonych przedmiotów z poziomu aplikacji webowej

#### **3.5 Dodawanie nowych list obecności**

możliwość tworzenia nowych list obecności z poziomu aplikacji webowej

#### **3.6 Dodawanie nowych studentów**

możliwość tworzenia nowych studentów z poziomu aplikacji webowej

#### **3.7 Dodawanie nowych prowadzących**

możliwość tworzenia nowych prowadzących z poziomu aplikacji webowej

#### **3.8 Dodawanie nowych przedmiotów**

możliwość tworzenia nowych prowadzonych przedmiotów z poziomu aplikacji webowej

### **3.9 Usuwanie list obecności**

możliwość usuwania list obecności z poziomu aplikacji webowej

### **3.10 Usuwanie studentów**

możliwość usuwania prowadzonych przedmiotów z poziomu aplikacji webowej

### **3.11 Usuwanie prowadzących**

możliwość usuwania prowadzących z poziomu aplikacji webowej

### **3.12 Usuwanie przedmiotów**

możliwość usuwania prowadzonych przedmiotów z poziomu aplikacji webowej

### **3.13 Uzupełnianie listy obecności za pomocą czytnika kart**

## **4. Wybór technologii informatycznych**

W tym rozdziale znajduje się opis wybranych technologii informatycznych. W kolejnych podsekcjach umieszczone zostały szczegółowe opisy technologii.

### **4.1 System Github**

Hostingowy serwis internetowy przeznaczony dla projektów programistycznych wykorzystujących system kontroli wersji Git.

### **4.2 Język programowania Java**

Język Java - język prosty w użyciu, bezpieczny i niezawodny.

### **4.3 Spring Framework**

Spring jest szkieletem tworzenia aplikacji w języku Java. Wybraliśmy springa ponieważ jest powszechnie używany i rozbudowany.

#### **4.4 AngularJS Framework**

AngularJS to otwarty framework oparty na języku JavaScript, wspierany i firmowany przez Google, wspomagający tworzenie i rozwój aplikacji internetowych na pojedynczej stronie.

#### **4.5 IntelliJ IDEA**

Zintegrowane środowisko programistyczne dla Javy.

#### **4.6 MySQL**

System zarządzania relacyjnymi bazami danych.

#### **4.7 Node.js**

Środowisko uruchomieniowe zaprojektowane do tworzenia wysoce skalowalnych aplikacji internetowych, szczególnie serwerów www napisanych w języku JavaScript.

#### **4.8 TypeScript**

Wolny i otwartoźródłowy język programowania stworzony przez firmę Microsoft jako nadzbiór języka JavaScript. Umożliwia on opcjonalne statyczne typowanie oraz programowanie zorientowane obiektowo oparte na klasach.

### **5. Architektura**

#### **5.1 Model View Controller**

Wzorzec projektowy służący do organizowania struktury aplikacji posiadających graficzne interfejsy użytkownika. Model-View-Controller zakłada podział aplikacji na trzy główne części:

- **Model** – jest pewną reprezentacją problemu bądź logiki aplikacji.
- **Widok** – opisuje, jak wyświetlić pewną część modelu w ramach interfejsu użytkownika. Może składać się z podwidoków odpowiedzialnych za mniejsze części interfejsu.
- **Kontroler** – przyjmuje dane wejściowe od użytkownika i reaguje na jego poczynania, zarządzając aktualizacje modelu oraz odświeżenie widoków.

## 5.2 Dependency Injection

Wzorzec projektowy i wzorzec architektury oprogramowania polegający na usuwaniu bezpośrednich zależności pomiędzy komponentami na rzecz architektury typu plug-in. Polega na przekazywaniu gotowych, utworzonych instancji obiektów udostępniających swoje metody i właściwości obiektom, które z nich korzystają (np. jako parametry konstruktora)

## 5.3 Data Access Object

Komponent dostarczający jednolity interfejs do komunikacji między aplikacją a źródłem danych (np. bazą danych czy plikiem). Jest często łączony z innymi wzorcami projektowymi. Dzięki DAO, aplikacja nie musi znać sposobu oraz ostatecznego miejsca składowania swoich danych, a ewentualne modyfikacje któregoś z czynników nie pociągają za sobą konieczności modyfikowania jej kodu źródłowego



## 5.4 Baza danych stworzona dzięki podejściu code first za pomocą frameworka Hibernate

### Lista obecności:

```
public AttendanceList(String name, String dateTime) {  
    this.name = name;  
    this.dateTime = dateTime;  
}  
  
public AttendanceList(String name, String dateTime, List<Student> studentList, Subject subject, Lecturer lecturer) {  
    this.name = name;  
    this.dateTime = dateTime;  
    this.studentList = studentList;  
    this.subject = subject;  
    this.lecturer = lecturer;  
}
```

Lista obecności składa się z:

- nazwy
- daty
- listy studentów
- przedmiotu
- prowadzącego.

### Prowadzący:

```
public Lecturer(String firstName, String lastName, String title) {  
    this.firstName = firstName;  
    this.lastName = lastName;  
    this.title = title;  
}
```

Klasa prowadzącego składa się z:

- imienia
- nazwiska
- tytułu

### Student:

```
public class Student {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "student_Id")  
    private Long id;  
    private String firstName;  
    private String lastName;  
    private String albumNumber;  
}
```

Klasa student składa się z:

- id
- imienia
- nazwiska
- numeru albumu

### Przedmiot:

```
public class Subject {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "subject_id")  
    private Long id;  
    @Enumerated(EnumType.STRING)  
    private SubjectType subjectType;  
    @Column(unique = true)  
    private String name;  
}
```

Klasa przedmiot składa się z:

- id,
- rodzaju przedmiot
- nazwy

### Lista studentów:

```
public class StudentList {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "studentlist_id")  
    private Long id;  
    private String name;  
    @OneToMany  
    private List<Student> studentList;  
}
```

Lista studentów składa się z:

- id
- nazwy
- listy studentów

## 5.5 Restowe Kontrollery:

### Kontroler studentów:

```
@RestController  
@RequestMapping("/students")  
public class StudentController {  
    private final StudentDAOImpl studentDAO;  
  
    @Autowired  
    public StudentController(StudentDAOImpl studentDAO) { this.studentDAO = studentDAO; }  
  
    @GetMapping("/{id}")  
    public Student findOne(@PathVariable long id) { return studentDAO.findOne(id); }  
  
    @GetMapping("")  
    public List<Student> getAll() { return studentDAO.findAll(); }  
  
    @RequestMapping("/create")  
    public void create(@RequestBody Student student) {  
        Student studentDummy = studentDAO.findByAlbumNumber(student.getAlbumNumber());  
        if (studentDummy == null) {  
            studentDAO.save(student);  
        }  
    }  
}
```

Kontroler listy studentów posiada mapowania funkcji na poszczególne adresy url. Mapowanie “create” pozwala na stworzenie nowego studenta, mapowanie “id” pozwala na wyświetlenie danych wybranego studenta. Mapowanie na “” pozwala na wyświetlenie listy wszystkich studentów w bazie danych. PathVariable to w adresie url w tym przypadku odpowiadająca wybranemu numerowi id obiektu, która w danej metodzie wyświetla wszystkie informacje na temat obiektu z danym numerem identyfikującym. Requestbody jest to obiekt przesyłany w formacie JSON.

### **kontroler przedmiotów:**

```
@RequestMapping("/subject")
public class SubjectController {
    private final SubjectDAOImpl subjectDAO;

    @Autowired
    public SubjectController(SubjectDAOImpl subjectDAO) { this.subjectDAO = subjectDAO; }

    @RequestMapping("/create")
    public void create(@RequestBody Subject subject) {
        Subject studentDummy = subjectDAO.findByNameAndSubjectType(subject.getName(), subject.getSubjectType());
        if (studentDummy == null) {
            subjectDAO.save(subject);
        }
    }

    @GetMapping("")
    public List<Subject> getAll() { return subjectDAO.findAll(); }
```

Kontroler przedmiotu posiada mapowania funkcji na poszczególne adresy url. Mapowanie “Create” pozwala na utworzenie nowego przedmiotu, natomiast mapowanie “” wyświetla listę wszystkich przedmiotów.

### **- kontroler listy obecności**

```
export class AttendanceComponent implements OnInit {
    attendances: Attendance[];
    studentList: Students[];
    constructor(private attendanceService: AttendanceService, private Router: Router) {
        this.attendances = [];
        this.attendanceService.getAttendanceList().subscribe(attendance => this.attendances = this.attendances.concat(attendance));
    }

    ngOnInit() {
    }

    showDetails(id) {
        console.log(id);
        this.Router.navigate(['attendance/', id]);
        console.log(this.Router.navigate(['attendance/', id]));
    }
}
```

## - kontroler prowadzącego

```
export class LecturerComponent implements OnInit {  
  
  lecturers: Lecturer[];  
  constructor(private lecturerService: LecturerService) {  
    this.lecturers = [];  
    this.lecturerService.getLecturer().subscribe(lecturer => this.lecturers = this.lecturers.concat(lecturer));  
  }  
  
  ngOnInit() {  
  }  
}
```

## 5.6 Komponenty

Komponent jest to element w aplikacji odpowiedzialny za dokładnie jedną funkcję. Element zawiera widok oraz logikę.

### - komponent studentów:

```
export class StudentsComponent implements OnInit {  
  
  @Input()  
  students: Students[];  
  constructor(private studentService: StudentService) {  
    this.students = [];  
    this.studentService.getStudents().subscribe(students => this.students = this.students.concat(students));  
  }  
  
  create(data: Students) {  
    if(data) {  
      this.studentService.addStudent(data).subscribe(() => this.getStudents());  
      data = null;  
    }  
  }  
  
  getStudents() {  
    this.students = [];  
    this.studentService.getStudents().subscribe(students => this.students = this.students.concat(students));  
  }  
  
  deleteStudent(id) {  
    this.studentService.deleteStudentById(id).subscribe(() => this.getStudents());  
  }  
}
```

## - komponent prowadzących

```
@Component({
  selector: 'app-lecturer',
  templateUrl: './lecturer.component.html',
  styleUrls: ['./lecturer.component.css'],
  providers: [LecturerService]
})
export class LecturerComponent implements OnInit {

  lecturers: Lecturer[];
  constructor(private lecturerService: LecturerService) {
    this.lecturers = [];
    this.lecturerService.getLecturer().subscribe(lecturer => this.lecturers = this.lecturers.concat(lecturer));
  }
}
```

## - komponent listy obecności

```
@Component({
  selector: 'app-attendance',
  templateUrl: './attendance.component.html',
  styleUrls: ['./attendance.component.css'],
  providers: [AttendanceService]
})
export class AttendanceComponent implements OnInit {
  attendances: Attendance[];
  studentList: Students[];
  constructor(private attendanceService: AttendanceService, private Router: Router) {
    this.attendances = [];
    this.attendanceService.getAttendanceList().subscribe(attendance => this.attendances = this.attendances.concat(attendance));
  }

  ngOnInit() {
  }

  showDetails(id) {
    console.log(id);
    this.Router.navigate(['attendance/', id]);
    console.log(this.Router.navigate(['attendance/', id]));
  }
}
```

## -komponent przedmiotów

```
@Input()
subjects: Subject[];
constructor(private subjectService: SubjectService) {
  this.subjects = [];
  this.subjectService.getSubject().subscribe(subject => this.subjects = this.subjects.concat(subject));
}

create(data: Subject) {
  if(data) {
    this.subjectService.addSubject(data).subscribe(() => this.getSubject());
    data = null;
  }
}

getSubject() {
  this.subjects = [];
  this.subjectService.getSubject().subscribe(subjects => this.subjects = this.subjects.concat(subjects));
}

deleteSubject(id) {
  this.subjectService.deleteSubjectById(id).subscribe(() => this.getSubject());
}
```



## -komponent szczegółowej listy obecności

```
export class AttendanceDetailsComponent implements OnInit {
  public attendanceList: Attendance;
  private eventID: number;

  constructor(private attendanceService: AttendanceService, private activatedRoute: ActivatedRoute) {
    this.activatedRoute.params.subscribe(param => {
      this.eventID = +param['id'];
      console.log(this.eventID);
    });
    console.log(this.attendanceList);
    this.attendanceService.getOne(this.eventID).subscribe(attendance => this.attendanceList = attendance);
    console.log(this.attendanceList);
  }

  getStudents() {
    this.activatedRoute.params.subscribe(param => {
      this.eventID = +param['id'];
      console.log(this.eventID);
    });
    console.log(this.attendanceList);
    this.attendanceService.getOne(this.eventID).subscribe(attendance => this.attendanceList = attendance);
    console.log(this.attendanceList);
  }
}
```

## 5.7 Serwisy

### - serwis studentów

```
private _studentsUrl = 'http://localhost:4200/students';
private _studentAddUrl = 'http://localhost:4200/students/create';
private _studentDeleteById = 'http://localhost:4200/students/delete/';
constructor(private _http: Http) {
}

getStudents(): Observable<Students[]> {
  return this._http.get(this._studentsUrl).map((res: Response) => res.json());
}

addStudent(data: any): Observable<Students> {
  let headers = new Headers({ 'Content-Type': 'application/json' });
  let options = new RequestOptions({ headers: headers });
  console.log(data);
  return this._http.post(this._studentAddUrl, JSON.stringify(data), options).map(this.extractData);
}

deleteStudentById(id): Observable<Students[]> {
  return this._http.delete(this._studentDeleteById + id).map(this.extractData);
}

private extractData(res: Response) {
  let body;

  if (res.text()) {
    body = res.json();
  }
}
```

- serwis prowadzących

```
export class SubjectService {

  private _subjectUrl = 'http://localhost:4200/subject'
  private _subjectAddUrl = 'http://localhost:4200/subject/create'
  private _subjectDeleteById = 'http://localhost:4200/subject/delete/';
  constructor(private _http: Http) {
  }

  getSubject(): Observable<Subject[]> {
    return this._http.get(this._subjectUrl).map((res: Response) => res.json());
  }

  addSubject(data: any): Observable<Subject> {
    let headers = new Headers({ 'Content-Type': 'application/json' });
    let options = new RequestOptions({ headers: headers });
    console.log(data);
    return this._http.post(this._subjectAddUrl, JSON.stringify(data), options).map(this.extractData);
  }

  deleteSubjectById(id): Observable<Subject[]> {
    return this._http.delete(this._subjectDeleteById + id).map(this.extractData);
  }

  private extractData(res: Response) {
    let body;

    if (res.text()) {
      body = res.json();
    }

    return body || {};
  }
}
```

- serwis listy obecności

```
getAttendanceList(): Observable<Attendance[]> {
  return this._http.get(this._attendanceUrl).map((res: Response) => res.json());
}

getOne(id): Observable<Attendance> {
  return this._http.get(this._getOneAttendanceUrl + id).map(this.extractData);
}

addStudentToList(data: any, id): Observable<Students> {
  let headers = new Headers({ 'Content-Type': 'application/json' });
  let options = new RequestOptions({ headers: headers });
  console.log(data);
  return this._http.post(this._addStudentToListUrl + id + '/students/add', JSON.stringify(data), options).map(this.extractData);
}

private extractData(res: Response) {
  return res.json() || {};
}
```



## - serwis przedmiotów

```
getSubject(): Observable<Subject[]> {
  return this._http.get(this._subjectUrl).map((res: Response) => res.json());
}

addSubject(data: any): Observable<Subject> {
  let headers = new Headers({ 'Content-Type': 'application/json' });
  let options = new RequestOptions({ headers: headers });
  console.log(data);
  return this._http.post(this._subjectAddUrl, JSON.stringify(data), options).map(this.extractData);
}

deleteSubjectById(id): Observable<Subject[]> {
  return this._http.delete(this._subjectDeleteById + id).map(this.extractData);
}

private extractData(res: Response) {
  let body;

  if (res.text()) {
    body = res.json();
  }

  return body || {};
}
```

## 5.8 Single Page Application

SPA (Single Page Application) to aplikacja lub strona internetowa, która w całości wczytuje się za jednym razem. Cały potrzebny do działania strony kod (HTML, CSS, JavaScript) przesyłany jest na początku lub dodawany dynamicznie w kawałkach, zwykle w odpowiedzi na interakcje generowane przez użytkownika.

## 5.9 Routing

Routing pozwala na łączenie kontrolerów z widokami i linkami url i mapuje asciejkę do istniejącej definicji trasy.

```
import {ModuleWithProviders} from '@angular/core';
import {RouterModule, Routes} from '@angular/router';
import {StudentsComponent} from './students/students.component';
import {LecturerComponent} from './lecturer/lecturer.component';
import {AttendanceComponent} from './attendance/attendance.component';
import {MainComponent} from './main/main.component';
import {SubjectComponent} from './subject/subject.component';
import {AttendanceDetailsComponent} from './attendance-details/attendance-details.component';

export const routes: Routes = [
  {path: '', component: MainComponent},
  {path: 'students', component: StudentsComponent},
  {path: 'lecturer', component: LecturerComponent},
  {path: 'attendance', component: AttendanceComponent},
  {path: 'subject', component: SubjectComponent},
  {path: 'attendance/:id', component: AttendanceDetailsComponent}
];
```

## 5.10 Moduły

Moduły są to kontenery dla różnych części aplikacji takich jak np. kontrolery, serwisy, filtry,

```
@NgModule({
  declarations: [
    AppComponent,
    StudentsComponent,
    AttendanceComponent,
    LecturerComponent,
    MainComponent,
    NavbarComponent,
    SubjectComponent,
    AttendanceDetailsComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    FormsModule,
    NgbModule.forRoot(),
    routing,
  ],
  providers: [StudentService, AttendanceService, LecturerService, SubjectService],
  bootstrap: [AppComponent]
})
```

Za pomocą głównego modułu łączymy wszystkie inne moduły ze sobą

## 6. Problemy i rozwiązania

### 6.1 Czytanie danych z legitymacji studenckiej

Ponieważ dokumentacja legitymacji studenckich była bardzo przestarzała, nie mogliśmy na początku wydobyć z legitymacji studenckich potrzebnych nam danych. Musieliśmy przechodzić po całej strukturze drzewa, aby trafić na potrzebne nam dane.

## 7. Instrukcja użytkowania aplikacji

- Uruchomienie serwera MySQL
- Uruchomienie serwera Node.js
- Uruchomienie programu
- Uruchomienie przeglądarki
- Przejście na adres <http://localhost:4200>
- Z menu wybrać gdzie chcemy się przekierować

Class Attendance Master
Home
Menu ▾

Attendance List
Students
Lecturers
Subjects

- W listach obecności można wybrać szczegóły danej listy

Attendance List

ID	Class Name	Date	Lecturer	Subject	Details
1	Lab PT	08/06/2017 12:06	mgr.inz Pierwszy Nauczyciel	PT cw Excercises	<a href="#">Details</a>
2	SI project	08/06/2017 13:06	dr Drugi Nauczyciel	SI Project	<a href="#">Details</a>

- W szczegółach list można dodawać nowych studentów

Attendance List

ID	Class Name	Date	Lecturer	Subject
1	Lab PT	08/06/2017 12:06	mgr.inz Pierwszy Nauczyciel	PT cw Excercises

ID	Name	Last Name	Album Number
1	Jan	Nowak	122011
2	Adrian	Nowak	122066
3	Adam	Halicki	116289
4	Jan	Kowal	21312

New Student:

First Name	Last Name	Album Number	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<a href="#">Add New</a>

- w zakładce Students wyświetlane są Imiona, Nazwiskawszyscy studenci zapisani w bazie danychmożemy dodawać i usuwać studentów

## Students

First Name	Last Name	Album Number	Options
Jan	Nowak	122011	<button>Delete</button>
Adrian	Nowak	122066	<button>Delete</button>
Adam	Halicki	116289	<button>Delete</button>
Jan	Kowal	21312	<button>Delete</button>
Piotr	Kowalski	155011	<button>Delete</button>
Szymon	Krzak	132326	<button>Delete</button>

New Student:

First Name	Last Name	Album Number	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<button>Add New</button>

- w zakładce Subjects możemy dodawać i usuwać przedmioty

## Subjects

Nazwa przedmiotu	Typ przedmiotu	ID	Options
PT cw	Excercises	1	<button>Delete</button>
SI	Project	2	<button>Delete</button>

New Subject:

Subject Name	Subject Type	ID	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<button>Add New</button>

- w zakładce Lecturers możemy dodawać i usuwać prowadzących

## Lecturers

Title	First Name	Last Name	Options
mgr.inz	Pierwszy	Nauczyciel	<button>Delete</button>
dr	Drugi	Nauczyciel	<button>Delete</button>

New Lecurer:

First Name	Last Name	Title	
<input type="text"/>	<input type="text"/>	<input type="text"/>	<button>Add New</button>

