

# פרויקט סמינר הישן



# הסמינר הישן

**שם המוסד החינוכי: מכון בית יעקב למורות-מודיעין עילית**

**שם הסטודנט: שירה כרמי**

**שם החברה המארחת: diversitech: טכנולוגיה**

**שם הלקוח הסופי: הסמינר הישן**

**שם הפרויקט: הסמינר הישן**

**תיאור הפרויקט: אתר לניהול האגף האדמיניסטרטיבי של המכון**

**שם המנחה האישי: תהילה אשלג**

**שם המנחה האקדמי: תהילה אשלג**

**תאריך הגשה :**

## תוכן העניינים:

1. מבוא
2. תיאור חברת Diversitek
3. תיאור הלקוח הסופי – הסמינר הישן
4. תיאור הפרויקט
5. מטרות הפרויקט
6. תהליך העבודה:
  - שלב א: הכרת החברה והפרויקט.
  - שלב ב: אפיון ודרישות.
  - שלב ג: תכנון.
  - שלב ד: ביצוע ופיתוח.
  - שלב ה: בדיקות ואבטחת איכות.
  - שלב ו: הטמעה ותמיכה.
7. סיכום ומסקנות.
8. נספחים.

## מבוא:

**תיאור כללי של הפרויקט :** אתר לניהול האגף האדמיניסטרטיבי של המכון.

כיום בית העסק מתנהל באמצעות קבצי excel. לניהול האדמיניסטרטיבי של המכון. בפרויקט זה מתוכנן לבצע המרה של צורת העבודה לדיגיטלית. הפרויקט נוגע בחלקים מסוימים של שירותי המשרד והם : קורסים, תלמידים ודווח נוכחות .

## מטרות המערכת:

1. תכלול תשתית אימות והרשאות לזיהוי המשתמש וסמכויותיו המערכת מאפשרת למזכירות לגשת למידע ופעולות באתר לפי הרשאות שקיבלו ממנהל האתר.
2. תכלול תשתית DB (Postgres SQL DB) - לשמירת המידע במרוכז.
3. תכלול תשתית ענן -המערכת מתוכננת להיות פרוסה בענן.
4. תהיה ידידותית למשתמש, אינפורמטיבית ואינטראקטיבית למכללה
5. תהיה חכמה מהירה וחדשנית -תפותח בארכיטקטורת MS , בשפות: TS , Java , Angular , Java spring boot ובטכנולוגיות:

# חברת Diversitek

**שם החברה :** דייברסיטק טכנולוגיה

**תחום עיסוק:**

דייברסיטק טכנולוגיה הינו בית תוכנה המספק פתרונות טכנולוגיים מגוונים ללקוחות קצה, ביניהם עסקים קטנים ובינוניים.

**מוצרים ושירותים:**

פיתוח תוכנות לניהול לידים ולקוחות ,  
בניית אתרי תדמית משוכללים ,  
פתרונות מותאמים אישית לפי דרישות הלקוח ,  
שירותי תמיכה טכנית והטמעה

**לקוחות החברה:**

עסקים קטנים ובינוניים ממגוון תחומים, חברות טכנולוגיה, מוסדות חינוך

**מבנה ארגוני:**

מנכ"ל: אביגיל מיכלסון

CTO: שוקי גור

PMO: חנה ברגמן

צוותי פיתוח ותמיכה

**תיאור תפקיד הסטודנטית בחברה:**

הסטודנטיות בפרקטיקום משתלבות בצוותי הפיתוח של דייברסיטק טכנולוגיה, ועובדות על פרויקטים טכנולוגיים אמיתיים עבור לקוחות החברה. במסגרת הפרקטיקום , הסטודנטיות לוקחות חלק בכל שלבי הפיתוח, החל מהאיפיון והתכנון, דרך הפיתוח והבדיקות, ועד להטמעה ותמיכה טכנית .

# תיאור לקוח הקצה - הסמינר הישן ירושלים

תיאור המכון באופן כללי:

מכון ההכשרה והשתלמויות של סמינר הישן בירושלים מציע לבוגרות הסמינר לימודי המשך, הכוללים קורסים פרונטליים ומקוונים להרחבת אופקים, וכן השתלמויות המוכרות לגמולים, ל"אופק חדש" ול"עוז לתמורה". [26] במסגרת המכון ניתן ללמוד לקראת תואר שווה ערך: "אקוויולנט לתואר בוגר" ("דרגה מס' 1") ו"אקוויולנט לתואר מוסמך" ("דרגה מס' 2"). [27]

בנוסף, מתקיימות במכון תוכניות לנשות חינוך ותיקות וגמלאיות. [26] (מתוך ויקיפדיה)

לפרטים נוספים על המכון: תיאור סמינר הישן - ויקיפדיה חרדית

האגף העיקרי שמולו הפרויקט מתנהל הוא:

האגף האדמיניסטרטיבי של המכון.

אנשי קשר:

חני לוין [chlevin@mbj.org.il](mailto:chlevin@mbj.org.il)

חני פוליקמן [ch-f@mbj.org.il](mailto:ch-f@mbj.org.il)

שולמית ברלין [shulamitberlin@gmail.com](mailto:shulamitberlin@gmail.com)

# תיאור הפרויקט

**תיאור כללי:** הקמת מערכת full-stack כמפורט בשקופית מספר 4.

**תפקיד הסטודנט:** מפתחת תשתיות full-stack

**מטרות ויעדים:** לספק פלטפורמה נגישה לתפעול האגף באופן דיגיטלי.

**מוצרים צפויים:**

1. DB מרכזי לשמירת כל נתוני התלמידים והקורסים.
2. מסכים ליצירה, עדכון ומחיקה של תלמידים וקורסים.
- 2.1 מסך לצפיית מערכת שעות של קורס בתצוגת לוח שנה + אפשרות להוסיף שיעור למערכת השעות הקיימת. וכן אופציות עריכת פרטי שיעור קיימים.
- 2.2 מסך לצפייה בהיסטוריית נוכחות של תלמיד בתצוגת לוח שנה.
- 2.3 מסך לצפייה בפרטי סטודנטים.
3. מסך לעדכון ודווח נוכחות
4. רכיב זיהוי פנים
5. מנגנון הפעלת פעולות אוטומטיות במערכת לפי הגדרת זמנים ותדירות של הצוות.
6. מסך הגדרת הגדרות מערכת
7. מסך הגדרת הרשאות

## מטרות הפרויקט:

### מטרות עיקריות:

שיפור נגישות המידע, הגברת מעורבות המשתמשים וייעול המשימות הניהוליות.

### תרומה לחברת Diversitech:

לספק פרויקט מוצלח ורווחי, להציג את יכולת החברה בפיתוח אתרים ולשפר את תיק העבודות של החברה.

### תרומה לסמינר הישן:

מעבר מניהול ידני לדיגטלי של כל נושאי התלמידות והקורסים. חיסכון באנשי צוות לניהול האגף. הרחבת פעולות המכון באמצעות המערכת שתמהר ותקצר תהליכים.



# תהליך העבודה:

## שלב 1: היכרות עם החברה והפרויקט

היכרות עם טכנולוגיית Diversitek תחום הפרויקט.  
מוצר נדרש: מסמך סקירת הפרויקט.

## שלב 2: אפיון ודרישות

כתיבת מסמך דרישות ותכנון ראשוני.  
מוצר נדרש: מסמך אפיון דרישות.

## שלב 3: תכנון

הכנת תוכניות עבודה וארכיטקטורת המערכת.  
מוצר נדרש: מסמך תכנון המערכת.

## שלב 4: ביצוע ופיתוח

כתיבת קוד, שימוש בטכנולוגיות המוגדרות  
(SQL DB, Java Spring Boot, jQuery, Bootstrap, CSS).  
מוצר נדרש: אבטיפוס ראשוני ומאגר קוד מקור.

## שלב 5: בדיקות ואבטחת איכות

ביצוע בדיקות ידניות ואוטומטיות.  
מוצר נדרש: תוכניות בדיקה, מקרי בדיקה ודוחות בדיקה.

## שלב 6: הטמעה ותמיכה

תיאור תהליך ההטמעה והתמיכה הטכנית.  
מוצר נדרש: מדריכי פריסה ומסמכי תמיכה.

## שלב א': הכרת החברה והפרויקט

ראשית הגעתינו לחברה התקיים כנס הסברה להצגת הלקוחות והפרויקטים בחברה.  
בכנס זה חולקנו לצוותות לפי פרויקטים ולאחר הכרות עם ראש הצוות התחילה העבודה  
במתודולוגיית אדג'ייל (Adgile).

# שלב ב': אפיון ודרישות

לאחר שיחות מול הלקוח, הופק אפיון עבור הפרויקט ע"י ה-CTO של Diversitech.
אנו קיבלנו את האפיון במסמכי word ב – google drive

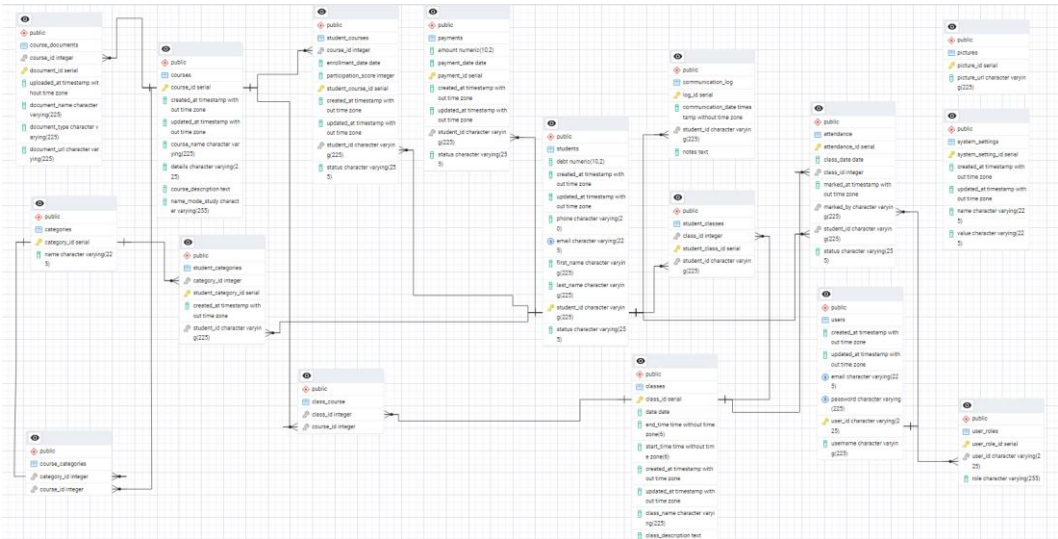
1. בתור מייסדת תשתית DB ישבתי על האפיון שקיבלנו וביצעתי ניתוח מעמיק המרתי לפורמט של טבלאות כך שכל שורה בטבלה מתארת את המשימות הנדרשות עבור פיתוח תשתית הDB עבור מסך ספציפי

אלו מסמכי האפיון עם הדגשות על ה-API'S הנדרשים:

המסך	URL	תיאור	הרשאה	נעשה?
Student Details Page	(get:by courseId and userId) getAllPaymentsByUserId getAllAttendanceByUserId getAllComLogByUserId	הצג וערוך את פרטי התלמידים, כולל מידע אישי, היסטוריית תשלומים, סטטוס הרשמה, רשומות השתתפות ויומני תקשורת.	מזכירה? ומנהל?	שירה
Student Search Screen	get:by ObjSearch {fName , lName, id }	חפש תלמידים או תלמידים פוטנציאליים לפי תעודת זהות או שם ורשום תוצאות רלוונטיות לגישה נוחה.	בהתאם למסך הבא,	שירה

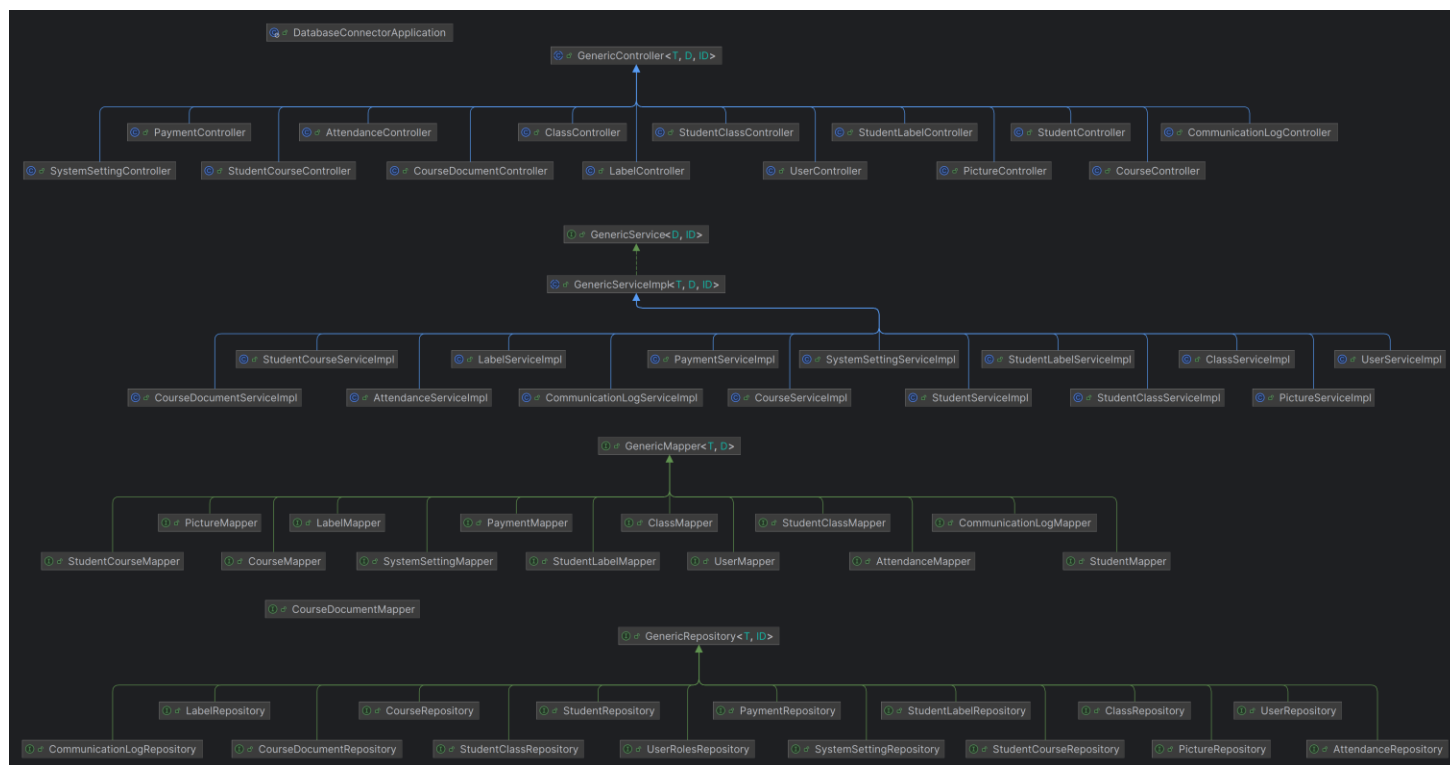
שלב ב': אפיון ודרישות-החלק שלי

2. בשל התנהלות הצוות במטולוגיית אדג'יל , לאחר שלב דמו עלו דרישות חדשות מצד הלקוח- אותן אפיינו מאפס בכל שכבות המערכת (מסך, סרוויסים | db-connector) כמו כן השתתפנו בשיחות אפיון עם ה CTO של החברה לצורך כך.

[illegible]

## שלב ג': תכנון-תשתית ה-DB

לפני הקמת תשתית ה DB (Db-connector MS) ישבנו לתכנן את מבנה הפרויקט :  
שכבות (packages) היררכית מחלקות (oop)



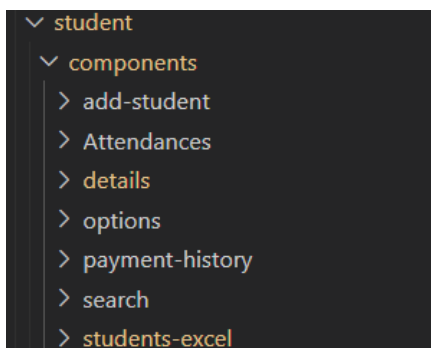
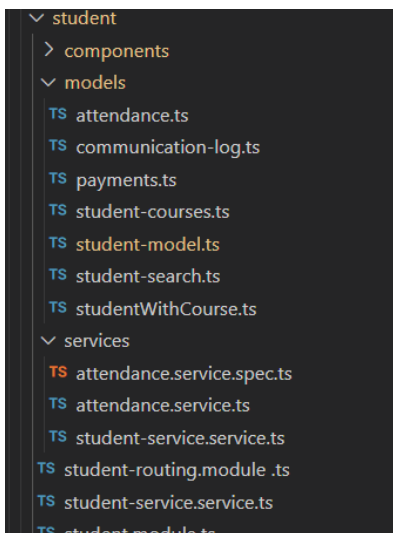
## שלב ג': תכנון-צד קליינט

בתור התחלה ישבתי עם מפתחות נוספות לתכנן את המבנה הארכיטקטי של הפרויקט בצד קליינט.

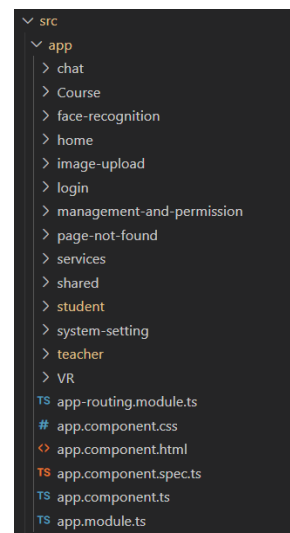
אציין שהשתמשנו בטכנולוגיית אנגולר לכתובת צד קליינט ועל כן התכנון כלל:

- הבנת דרישות הלקוח מתוך מסמכי האיפיון לעומק.
- חלוקת מסכי האפליקציה למודולים לפי המסכים שנדרשו, כשהמטרה שכל מפתחת תהיה אחראית על feature אחד.
- איפיונו לעומק את המודול סטודנט חילקנו את הדרישות לקומפוננטות, ואת העבודה בין הבנות.
- אני פיתחתי את קומפוננטת search ו-routing של המודול (יפורט בהמשך)

פירוט החלוקה של המודול סטודנט:



פירוט החלוקה למודולים:



הראש צוות תכנן את ארכיטקטורת צד הסרבר באמצעות סרוויסים שונים.

## שלב ד': ביצוע ופיתוח

במהלך הפיתוח פיתחתי באזורים שונים בכל רחבי האפליקציה, בשל כך הטכנולוגיות והביצועים שאפרט להלן הינן מגוונות הן מבחינת סוגיהן והן מבחינת רמתן:

### תאור כללי של כל המשימות שלי בתשתית ה-DB-CONNECTOR :

#	החלק הסופי במוצר:	הטכנולוגיה	השפה	תיאור המשימה
*	תשתית לניהול מסד הנתונים של האתר . התשתית חושפת API'S ל- MS השונים במערכת שבאמצעותם נשמרים ומתעדכנים נתונים במסד	<p><b>Spring Boot</b>: לשימוש בבניית יישומים מבוססי Spring Framework.</p> <p><b>Maven</b>: לניהול התלויות בפרויקט ובנייתו.</p> <p><b>H2 Database</b>: מסד נתונים מובנה שמאפשר פיתוח קל וקליל, במיוחד לצורכי פיתוח ובדיקה.</p> <p><b>Java</b>: השפה העיקרית לכתיבת הקוד.</p>	<p>השפה המרכזית שתשמש לפיתוח הפרויקט היא Java, יחד עם שימוש ב- SQL לניהול מסד הנתונים.</p>	<p><b>1. היכרות עם מבנה הנתונים ומסמכי האפיון:</b> מעבר על מסמכי האפיון הקיימים. הבנה של מבנה הטבלאות הקיימים. עיבוי ויצירת השכבות המטפלות בפעולות על הטבלאות תוך כדי שימוש בספריית JPA והקפדה על:</p> <p>1. עקרונות ה-OOP , עקרונות הגנריות, עקרונות ה-spring. BOOT</p> <p>2. מיצוי שימוש ביכולות הספרייה JPA הקיימות</p> <p>3. חשיפת API'S של התשתית והטמעתו בצוות</p>

# שלב ד': ביצוע ופיתוח

## תיאור פרטני של משימה 1:

#	החלק הסופי במוצר:	הטכנולוגיה	השפה	תיאור המשימה
1.	תשתית גנרית לניהול מסד הנתונים של האתר . הכתובה והפועלת על פי עקרונות ה oop . התשתית שומרת ומעדכנת נתונים במסד	<b>Spring Boot</b> : לשימוש בבניית יישומים מבוססי Spring Framework. <b>Java</b> : השפה העיקרית לכתיבת הקוד.	Java	עיבוי ויצירת השכבות המטפלת בפעולות על הטבלאות תוך כדי שימוש בספריית JPA והקפדה על עקרונות ה-OOP : עבור כל טבלה בDB יצרנו את האובייקטים (מחלקות) הבאים:  1. Controller 2. Dto 3. Service 4. Mapper 5. Repository 6. Entity  הקפדנו ליצור תשתית גנרית כך שכל האובייקטים הנ"ל מבוססים ויורשים ממחלקה גנרית ככל האפשר. בגרסאות הבאות תחזוקת תשתית זו תהיה זולה וקלה למפתחים הבאים היות ובשל המחלקה הגנרית תוספות של טבלאות חדשות יתבצעו בפשטות . (טבלה חדשה דורשת יצירה של כל האובייקטים הנ"ל כל אובייקט יירש מהמחלקה הבסיסית והגנרית שיצרנו היות שה-Spring.boot מנהל לי את כל מחזור החיים של האובייקטים ולכן הוא צריך נתונים לפני ההרצה כדי לבנות אותם אצלו ואי אפשר להתשמש בגנרי בזמן ריצה ולכן נצרך יצירה חדשה של כל השכבות לכל טבלה ועדיין הוספת טבלה חדשה היא תהליך פשוט למדי בזכות הגנריות) את התשתית הגנרית יצרנו על פי עקרונות ה-OOP במסכים הבאים אפרט את היררכית ההמחלקות של האובייקטים הנ"ל.



# פרויקט גנרי להטמעת CRUD (OOP)

## 1. הגדרת entities והכרת JPA

יצירת מחלקות entities והגדרת שדות וטבלאות.

שימוש באנוטציות כמו :

@Entity, @Id, @GeneratedValue.

## 2. יצירת repositories גנריים וייחודיים:

יצירת repositories שהם יורשים מ- repository גנרי שמורכב מ- JpaRepository כך שכל repository מכיל פונקציות בסיסיות של JpaRepository + פונקציות ייחודיות.

שימוש באנוטציות: @Repository, @NoRepositoryBean.

## 3. בניית services והזרקת תלויות:

יצירת Generic Service שמכיל את הפעולות הבסיסיות של CRUD והוא, abstract, הזרקת תלויות באמצעות @Autowired.

לאחר מכן, יצירת שירותים ייחודיים לכל ישות Entity-specific Services שמרחיבים את השירותים הגנריים ומוסיפים פונקציות ייחודיות לכל ישות.

## 4. יצירת Controllers גנריים וייחודיים:

יצירת Controllers גנריים שמקבלים את השירותים הגנריים, וכל ה- Controllers יורשים מהם.

יצירת מחלקות controller עם אנוטציות: @RestController, @RequestMapping.

יצירת endpoints לכל פעולת CRUD.

## 5. הוספת Mappers לחילוץ והמרת נתונים ל- DTO:

השתמשי בספריית - MapStruct.

יצרתי Mapper גנרי שישמש למיפוי הנתונים בכל השירותים הגנריים שלנו, וממנו ירשו כל Mappers הספציפיים ליישומים השונים בפרויקט.

# דוגמא של קוד ודיאגרמה של ישות –Student :

Controller:

Service:

Mapper:

Entity, DTO:

Repository:

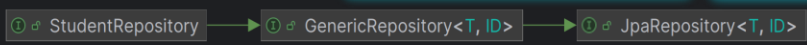
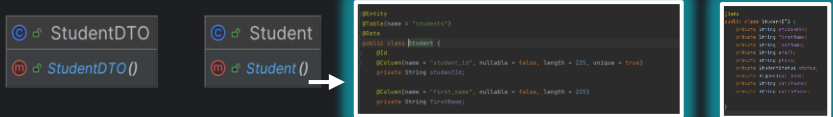
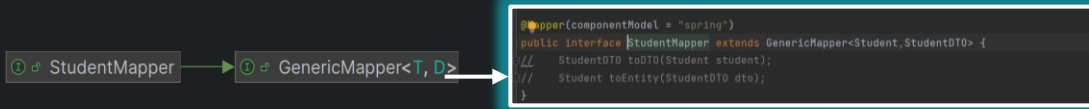
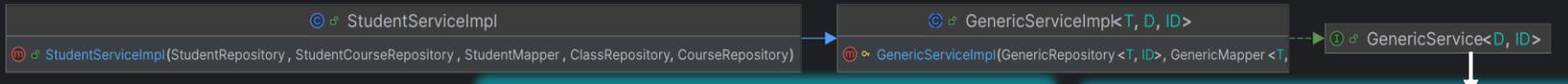
```
@RestController
@RequestMapping("/api/db/students")
public class StudentController extends GenericController<Student, StudentDTO, String> {

    private StudentServiceImpl studentService;

    @Autowired
    public StudentController(StudentServiceImpl studentService) {
        super(studentService);
        this.studentService = studentService;
    }

    @GetMapping("/{byName}")
    @ResponseBody
    public List<StudentDTO> getStudentBySearch(@RequestParam(name="firstName") String firstName, @RequestParam(name="lastName")
        System.out.println("in controller" + lastName + "=>" + firstName);
        return studentService.getSearchStudent(firstName, lastName);
    }

    @GetMapping("/{getClassId/{classId}")
    public ResponseEntity<List<Student>> getAllStudentByClassId(@PathVariable Integer classId) {
        List<Student> documents = studentService.findAllStudentsByClassId(classId);
        return ResponseEntity.ok(documents);
    }
}
```



```
@Repository
public interface StudentRepository extends GenericRepository<Student, String> {

    List<Student> findByFirstNameAndLastName(String firstName, String lastName);

    List<Student> findByLastName(String lastName);

    List<Student> findByFirstName(String firstName);
}
```

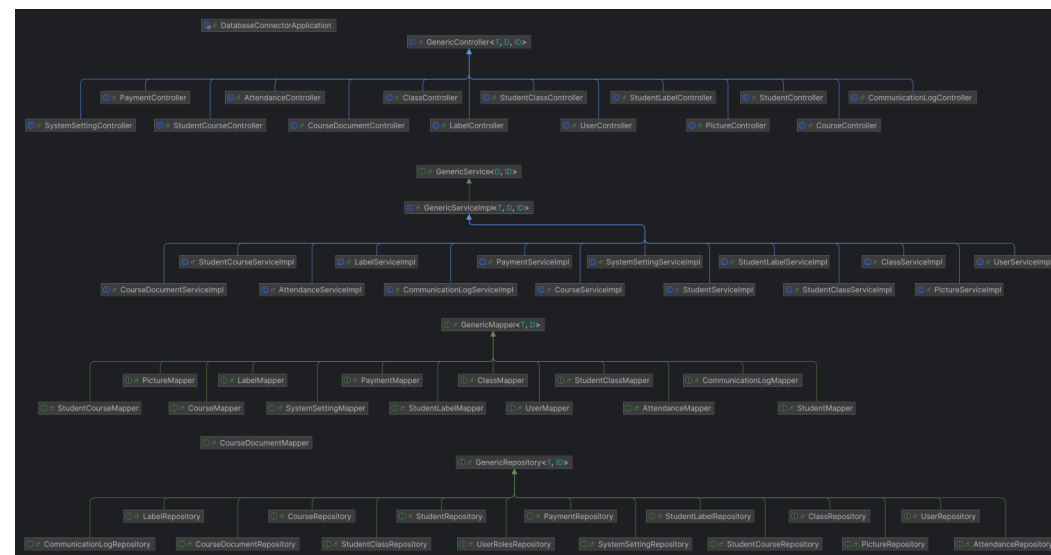
```
@Service
public class StudentServiceImpl extends GenericServiceImpl<Student, StudentDTO, String> {
    private final StudentRepository studentRepository;
    private final StudentCourseRepository studentCourseRepository;
    private final StudentMapper studentMapper;
    private final ClassRepository classRepository;
    private final CourseRepository courseRepository;

    @Autowired
    public StudentServiceImpl(StudentRepository studentRepository, StudentCourseRepository studentCourseRepository, StudentMapper studentMapper, ClassRepository classRepository, CourseRepository courseRepository) {
        super(studentRepository, studentMapper);
        this.studentRepository = studentRepository;
        this.studentCourseRepository = studentCourseRepository;
        this.studentMapper = studentMapper;
        this.classRepository = classRepository;
        this.courseRepository = courseRepository;
    }

    public List<StudentDTO> getStudentsByCourseId(Integer courseId) {
        List<StudentCourse> studentCourses = studentCourseRepository.findAllByCourse_CourseId(courseId);
        return studentCourses.stream()
            .map(studentCourse -> studentCourse.getStudent())
            .map(studentCourse -> studentCourse.getStudent())
            .map(studentCourse -> studentCourse.getStudent())
            .collect(Collectors.toList());
    }
}
```

# דיאגרמה כללית לכל ה-DB-Connector:

ההזרקות הן:  
 controller-ל service  
 service-ל repository ו mapper



# שלב ד': ביצוע ופיתוח

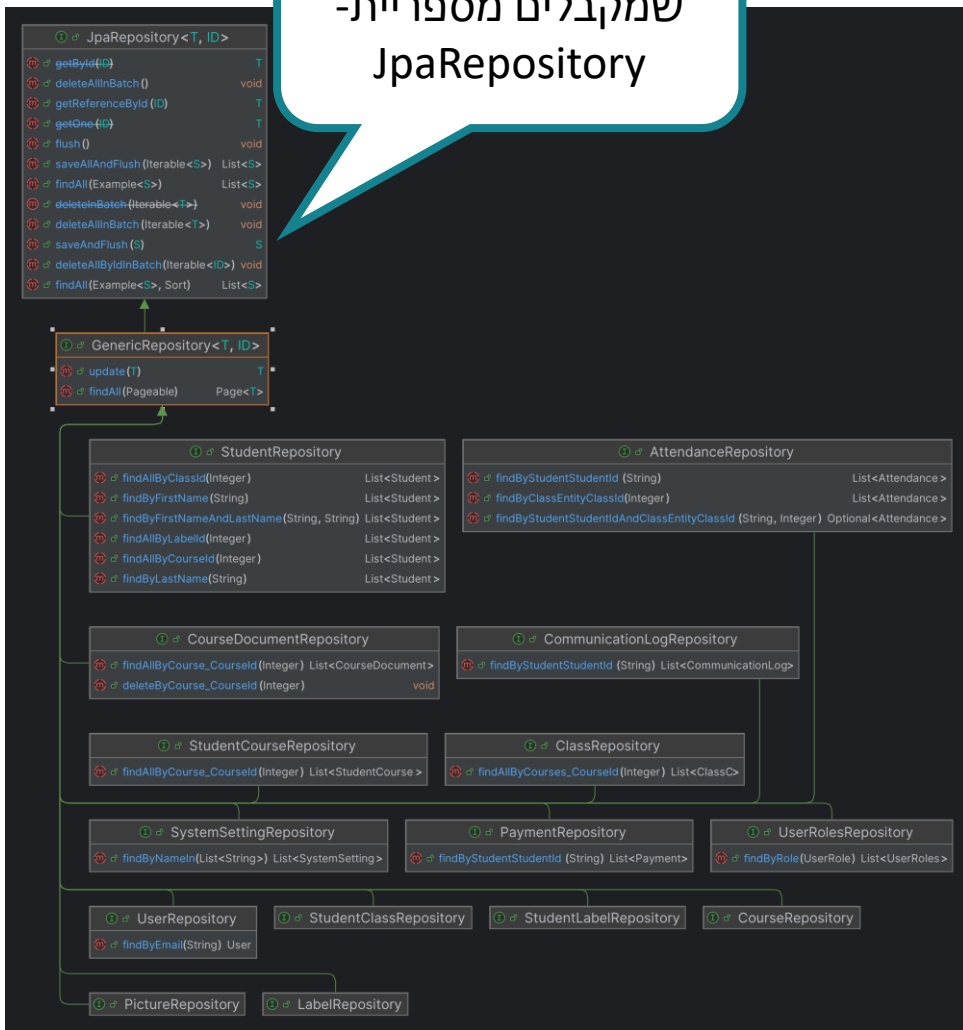
## תיאור פרטני של משימה 2:

#	החלק הסופי במוצר:	הטכנולוגיה	השפה	תיאור המשימה
2.	תשתית גנרית לניהול מסד הנתונים של האתר . הכתובה וממצה שימוש ביכולות הקיימות בספריית JPA התשתית שומרת ומעדכנת נתונים במסד	<b>Spring Boot</b> : לשימוש בבניית יישומים מבוססי Spring.Framework <b>Java</b> : השפה העיקרית לכתיבת הקוד.	Java	<p>1. ראשית חקרנו על ספריית JPA והיכולות האוטומטיות שלה.</p> <p>2. יצרנו אובייקט Repository עבור כל טבלה שיורש מהאובייקט בסיס ומקבל ע"י כך את כל היכולות האוטומטיות של JpaRepository היכולות הם בעצם הפונקציונליות של הספרייה שעושה פעולות על הנתונים מ-DB (יצירת אובייקט עבור כל טבלה היא עצה של מפתח מנוסה על סמך ניסיונו מהתעשייה לשם פשטות התחזוקה )</p> <p>3. היכולות האוטומטיות של JPA מבצעות פעולות על טבלאות ע"י פיענוח של חתימות פונקציות הקיימות ב- &lt;class&gt;Repository</p> <p>תפקידנו היה להבין את חוקיות החתימות ולהוסיף חתימות מתאימות לקלאסים השונים לפי דרישות המערכת (client-side)</p> <p>במסכים הבאים נציג צילומי מסך של אובייקט רפוסטורי של טבלה ספציפית שהיא מדגם לטבלאות רבות שעשיתי</p>

# שלב ד': ביצוע ופיתוח

## צילומי מסך של JPA:

פונקציות השליפה  
שמקבלים מספריית-  
JpaRepository



Code of GenericRepository extend from JpaRepository :

```
@NoRepositoryBean
public interface GenericRepository <T, ID extends Serializable> extends JpaRepository<T, ID> {
    T update(T entity);
}
```

דוגמא לירושה מהגנרי של הישות Attendance - AttendanceRepository :

```
@Repository
public interface AttendanceRepository extends GenericRepository<Attendance,Integer> {
    List<Attendance> findByStudentStudentId(String studentId);
    List<Attendance> findByClassEntityClassId(Integer classId);
}
```

```
@Repository
public interface PaymentRepository extends GenericRepository<Payment, Integer> {
    List<Payment> findByStudentStudentId(String studentId);
}
```

```
@Repository
public interface ClassRepository extends GenericRepository<ClassC, Integer> {
    List<ClassC> findAllByCourses_CourseId(Integer courseId);
}
```

## שלב ד': ביצוע ופיתוח

### תיאור פרטני של משימה 3:

#	החלק הסופי במוצר:	הטכנולוגיה	השפה	תיאור המשימה
3.	אוסף API'S שמבצעים פעולות על ה-DB	<b>Spring Boot</b> : לשימוש בבניית יישומים מבוססי Spring Framework. <b>Java</b> : השפה העיקרית לכתיבת הקוד.	Java	<p>עבור כל טבלה אני חושפת אוסף של API'S שדרכם ניתן לבצע פעולות על טבלאות .</p> <p>במסכים הבאים אראה את התהליך עבור טבלה ספציפית שהיא מדגם לטבלאות רבות שעשיתי :</p> <p>1. אציג את מחלקת הקונטרולר שחושפת את הApi's המתאימים לטבלה זו .</p> <p>2. אציג קריאה לAPI הנ"ל דרך , Postman</p> <p>אציג שהקריאה בוצעה בהצלחה והתקבל response-200 כמו כן אראה את השינויים שבוצעו בהצלחה ב- DB (הראתי את השליפות דרך הדפדפן chrom)</p>

# שלב ד': ביצוע ופיתוח

## צילומי מסך למשימה 3:

לאחר הקמת התשתית ה-DB-Connector חשפנו API'S CRUD (גנרי) והוספנו ה-API'S לפי האפיון הראשוני (פורט בהתחלה באפיון)

getAllPaymentsByStudentId

פונקציה שליפה נוספת בנוסף לגנרי:

```
public abstract class GenericController<T, D, ID extends Serializable> {
    protected final GenericServiceImpl<T, D, ID> crudService;
    public GenericController(GenericServiceImpl<T, D, ID> crudService) { this.crudService = crudService; }
    @GetMapping("/all/{pageable}")
    public ResponseEntity<List<D>> getAll(@PathVariable int pageable) {...}
    @GetMapping("/all")
    public ResponseEntity<List<D>> getAll() {...}
    @GetMapping("/{id}")
    public ResponseEntity<D> getById(@PathVariable ID id) {...}
    @PostMapping()
    public ResponseEntity<?> create(@RequestBody D entity) {...}
    @PostMapping(value = "/addList")
    public ResponseEntity<?> create(@RequestBody List<D> entityies) {...}
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> delete(@PathVariable ID id) {...}
    @Transactional
    @PutMapping("/{id}")
    public ResponseEntity<D> update(@RequestBody D entity) {...}
}
```

```
@RestController
@RequestMapping("/api/db/payments")
public class PaymentController extends GenericController<Payment, PaymentDTO, Integer> {

    private PaymentServiceImpl paymentService;
    @Autowired
    public PaymentController(PaymentServiceImpl paymentService) {...}

    @GetMapping("/byStudentId/{studentId}")
    public ResponseEntity<List<PaymentDTO>> getByStudentId(@PathVariable String studentId) {
        List<PaymentDTO> payments = this.paymentService.getAllPaymentByStudentId(studentId);
        return ResponseEntity.ok(payments);
    }
}
```

פונקציות שליפה נוספות ב CourseDocument

```
@RestController
@RequestMapping("/api/db/courseDocuments")
public class CourseDocumentController extends GenericController<CourseDocument, CourseDocumentDTO, Integer> {

    private CourseDocumentServiceImpl courseDocumentService;
    @Autowired
    public CourseDocumentController(CourseDocumentServiceImpl courseDocumentService) {...}

    @GetMapping("/getAllByCourseId/{courseId}")
    public ResponseEntity<List<CourseDocumentDTO>> getAllDocByCourseId(@PathVariable Integer courseId) {...}

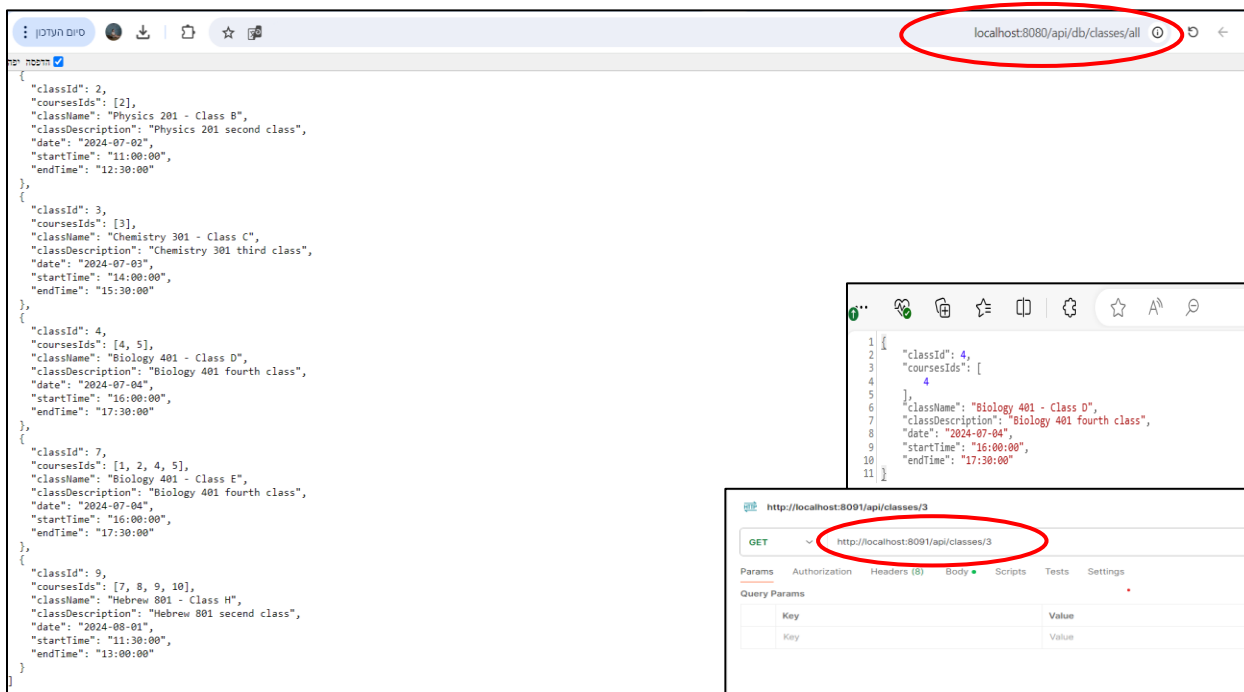
    @DeleteMapping("/deleteByCourseId/{courseId}")
    public ResponseEntity<String> deleteDocumentsByCourseId(@PathVariable Integer courseId) {...}
}
```

# שלב ד': ביצוע ופיתוח

## צילומי מסך למשימה 3:

שליפות שבוצעו בהצלחה:

דוגמא לשליפת גרנית של הישות Classes:  
<http://localhost:8080/api/db/classes/all>



```

[
  {
    "classId": 2,
    "coursesIds": [2],
    "className": "Physics 201 - Class B",
    "classDescription": "Physics 201 second class",
    "date": "2024-07-02",
    "startTime": "11:00:00",
    "endTime": "12:30:00"
  },
  {
    "classId": 3,
    "coursesIds": [3],
    "className": "Chemistry 301 - Class C",
    "classDescription": "Chemistry 301 third class",
    "date": "2024-07-03",
    "startTime": "14:00:00",
    "endTime": "15:30:00"
  },
  {
    "classId": 4,
    "coursesIds": [4, 5],
    "className": "Biology 401 - Class D",
    "classDescription": "Biology 401 fourth class",
    "date": "2024-07-04",
    "startTime": "16:00:00",
    "endTime": "17:30:00"
  },
  {
    "classId": 7,
    "coursesIds": [1, 2, 4, 5],
    "className": "Biology 401 - Class E",
    "classDescription": "Biology 401 fourth class",
    "date": "2024-07-04",
    "startTime": "16:00:00",
    "endTime": "17:30:00"
  },
  {
    "classId": 9,
    "coursesIds": [7, 8, 9, 10],
    "className": "Hebrew 801 - Class H",
    "classDescription": "Hebrew 801 second class",
    "date": "2024-08-01",
    "startTime": "11:30:00",
    "endTime": "13:00:00"
  }
]
    
```

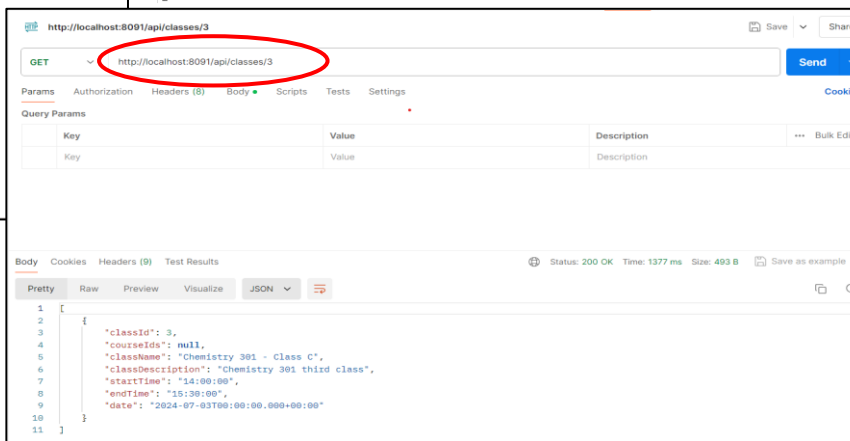
דוגמא לשליפת גרנית של הישות Classes:  
<http://localhost:8080/api/db/classes/4> - **getById**



```

{
  "classId": 4,
  "coursesIds": [
    4
  ],
  "className": "Biology 401 - Class D",
  "classDescription": "Biology 401 fourth class",
  "date": "2024-07-04",
  "startTime": "16:00:00",
  "endTime": "17:30:00"
}
    
```

דוגמא לשליפת מ-classes-service  
 ששולף נתונים מ-DB-Connector  
<http://localhost:8091/api/classes/3> - **getById**



```

{
  "classId": 3,
  "coursesIds": null,
  "className": "Chemistry 301 - Class C",
  "classDescription": "Chemistry 301 third class",
  "date": "2024-07-03",
  "startTime": "14:00:00",
  "endTime": "15:30:00",
  "date": "2024-07-03T00:00:00+00:00"
}
    
```



# שלב ד': ביצוע ופיתוח

## צילומי מסך למשימה 3:

במהלך הטמעת ה– DB-connector לכל צוות הפיתוח, התקבלו פניות חדשות ליצירת שליפות נתונים מסוימות.

עבדתי על מנת לפתח את השליפות המדויקות בהתאם לדרישות. תיאמתי את הדרישות עם הצוות , ווידאתי שהשליפות מסופקות בצורה נכונה ויעילה, ותיקנתי בעיות שהתעוררו במהלך התהליך.

במהלך תהליך זה נוצרה דרישה לא להעמיס על הקליינט בשליפת get ולכן עשיתי את ה-Paging (יפורט בשקופית 25)

להלן צילומי מסך של האקסל שתחזקנו ע"מ לקבל את הדרישות מהקליינט וליישם אותם:

B	C	D	E	F	G	H	I
	do?	return	url	name of function - כרטיס - משימה	Post/Put/Get/Delete	מקבל גישה מודול API-ה	לשים את אחד מה שיש כאן וזה בעצם גודל הוא ישמש לנו את המספרים אותו דבר כל פעם על attendance-מספר במקום ה
		ResponseEntity<String>	https://old-seminar-2-back-4hid.onrender.com/api/attendance/all	getAll	Get		classes, communicationLog, course, courseDocuments, payment, students, studentCourse, users
		List<P>	https://old-seminar-2-back-4hid.onrender.com/api/attendance/by/id/{id}	getById	Get	url- id:ID	classes, communicationLog, course, courseDocuments, payment, students, studentCourse, users
		D	https://old-seminar-2-back-4hid.onrender.com/api/attendance/update	put	Put	body- entity:D	classes, communicationLog, course, courseDocuments, payment, students, studentCourse, users
		Void	https://old-seminar-2-back-4hid.onrender.com/api/attendance/delete/{id}	delete	Delete	url- id:ID	classes, communicationLog, course, courseDocuments, payment, students, studentCourse, users
		D	https://old-seminar-2-back-4hid.onrender.com/api/attendance	post	post	body- entity:D	classes, communicationLog, course, courseDocuments, payment, students, studentCourse, users
		<List<AttendanceDTO>	https://old-seminar-2-back-4hid.onrender.com/api/attendance/getAllByStudentId/{studentId}	getAttendanceByStudentId	Get	url- studentId:String	
		List<StudentDTO>	https://old-seminar-2-back-4hid.onrender.com/api/attendance/getStudents(classId)	getStudentsByClass	Get	url- classId:Integer	
		<List<ClassDTO>	https://old-seminar-2-back-4hid.onrender.com/api/classes/getAllByCourseId/{courseId}	getAllClassesByCourseId	Get	url- courseId:Integer	
		List<CommunicationLogDTO>	https://old-seminar-2-back-4hid.onrender.com/api/communicationLog/getAllByStudentId/{studentId}	getCommunicationLogByStudentId	Get	url- studentId:String	
		List<CourseDocumentDTO>	https://old-seminar-2-back-4hid.onrender.com/api/courseDocuments/deleteByCourseId/{courseId}	deleteDocumentsByCourseId	Delete	url- courseId:Integer	
		String	https://old-seminar-2-back-4hid.onrender.com/api/paymentByStudents/{studentId}	getPaymentByStudentId	Get	url- studentId:String	
		List<PaymentDTO>	https://old-seminar-2-back-4hid.onrender.com/api/students/students-by-course/{courseId}	getStudentsByCourse	Get	url- courseId:Integer	
		List<StudentDTO>	https://old-seminar-2-back-4hid.onrender.com/api/users/getUserByEmail	findByEmail	post	url- email:String	
		UserDTO	https://old-seminar-2-back-4hid.onrender.com/api/users/deleteUser(email)	deleteUser	Delete	url- email:String	
		Void					
			להוסיף:				
		Void	localhost:8080/api/classes/deleteClass/{id}	deleteClass	Delete	url - classId: Integer	
		Void	localhost:8080/api/classes/addClass/	addClass	Post	url - class:Classes	
		Void	localhost:8080/api/classes/updateClass/{id class}	updateClass	Put		
		List<UserEntity>	localhost:8080/api/users/getUsersByRole/teacher	getUsersByRole	get		
		UserPermission	localhost:8080/api/users/addUser	addUser	post		
		Void	localhost:8080/api/users/changeUserPermission	changeUserPermission	post		
		Void	localhost:8080/api/users/updateUser	updateUser	post		
		AttendanceDTO	localhost:8080/api/attendance/createAttendances	createAttendances	post	body- listAttendances:<List<Attendance>	
		AttendanceDTO	localhost:8080/api/attendance/updateAttendanceStatus/{studentId=1&classId=2&status=current	updateAttendanceStatus	put	@RequestParam String studentId, @RequestParam String studentId, @RequestParam String studentId,	

דוגמא לפונקציה שמימשתי:

```

@PutMapping("/updateAttendanceStatus")
public ResponseEntity<String> updateAttendanceStatus(
    @RequestParam String studentId,
    @RequestParam Integer classId,
    @RequestParam String status) {
    AttendanceStatus attendanceStatus;
    try {
        attendanceStatus = AttendanceStatus.valueOf(status.toUpperCase());
    } catch (IllegalArgumentException e) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Invalid AttendanceStatus: " + status);
    }
    boolean updated = attendanceService.updateAttendanceStatus(studentId, classId, attendanceStatus);

    if (updated) {
        return ResponseEntity.ok("Attendance status updated successfully.");
    } else {
        return ResponseEntity.status(404).body("Attendance record not found.");
    }
}

```

# שלב ד': ביצוע ופיתוח

## : Paging

### יישום Paging לשליפה חכמה ומדויקת של נתונים

- **רקע והצורך:**  
בהתאם לדרישות העסקיות, שליפות נתונים מסיביות מהמסד נתונים עלולות להכביד על הלקוח ולגרום להורדת ביצועים ולצריכת משאבים מיותרת. תהליך שליפה שלא נבדק מספיק יכול להוביל למצבים שבהם הלקוח מקבל כמויות נתונים שהוא אינו זקוק להם, דבר שמבזבז זמן ומשאבים.
- **פתרון יישום:**  
לצורך פתרון הבעיה, יישמתי **Paging** שתהליך בו הנתונים מחולקים לדפים קטנים יותר ומועברים ללקוח לפי הצורך. יישום זה לא רק שמפחית את העומס על השרת והלקוח, אלא גם מספק את הנתונים בצורה מדויקת וממוקדת, מבלי להעמיס על המערכת בנפחי נתונים מיותרים.
- בנוסף, התאמתי את פתרון ה-**Paging** כך שיהיה גנרי וקל להתאמה, מה שמפחית את נפח הקוד ומשפר את הקריאות והתחזוקה של הקוד. השיטה הכללית מאפשרת חידוש בקלות עבור סוגים שונים של ישויות ושליפות, ומפשטת את ההתמודדות עם דרישות משתנות בעתיד.

```

@GetMapping("/all/{pageable}")
public ResponseEntity<List<D>> getAll(@PathVariable int pageable) {
    Pageable firstPageWithTwoElements = PageRequest.of(pageable, pageSize: 3);
    List<D> dtos = crudService.findAll(firstPageWithTwoElements);
    return ResponseEntity.ok(dtos);
}

```

```

@NoRepositoryBean
public interface GenericRepository <T, ID extends Serializable> extends JpaRepository<T, ID> {
    T update(T entity);
    Page<T> findAll(Pageable pageable);
}

```

# Git-Github

במהלך הסטאג' העברתי סשן בנושא Git להלן לינק למצגת בדרייב שהצגתי בסשן : [למצגת על ה-Git](#)  
 בנוסף ניהלתי את כל העבודה ב-Git ליוויתי את המפתחות בצוות בנושא והוא היה תחת אחריותי במסגרת  
 זה הגדרתי את ה-Branch-ים השונים ואת מטרותיהם.  
 מצורף צילום מסך של מסמך שכתבתי לצוות לגבי עבודה  
 עם ה-Branch-ים ב-Angular.

4. מיזוג הקומפוננטה ל branch של המודול:

- לאחר שהקומפוננטה מוכנה ובדקה, בצעו pull request למיזוג ה branch של הקומפוננטה ל branch של המודול.
- ודאו שכל הבדיקות עוברות לפני המיזוג ושאינן קונפליקטים.

5. מיזוג המודול ל main-בפרויקט הכללי:

- כאשר המודול כולו מוכן, בצעו pull request למיזוג ה branch של המודול ל branch הראשי (main) של ה repository הכללי.
- ודאו שכל הבדיקות עוברות ושאינן קונפליקטים לפני המיזוג הסופי ל main של הפרויקט.

6. תחזוקה ועדכונים:

- הקפידו לעדכן את ה branch של המודול בכל עדכון חשוב מה ה branch הראשי (main) כדי להימנע מקונפליקטים בעת המיזוג הסופי.**
- במידה ויש צורך בעדכונים או תיקונים במודול, חזרו על שלבים 2-5 בהתאם לצורך.

טיפים למניעת קונפליקטים:

- תקשורת שוטפת:** הקפידו על תקשורת טובה בין חברי הצוות ובין הקבוצות השונות כדי למנוע כפילויות בעבודה ושינויים מתנגשים.
- עדכונים תכופים:** משכו עדכונים מה main-אל ה branch של המודול באופן תדיר כדי לוודא שאתם עובדים על גרסה מעודכנת של הקוד.
- מיזוגים קטנים ותכופים:** בצעו מיזוגים קטנים ותכופים כדי להקל על פתרון קונפליקטים לפני שהם מצטברים והופכים לקשים יותר לפתרון.

הנחיות עבודה עם Git בפרויקט Angular1 עבודה בקבוצות על מודולים:

כל קבוצה שאחראית על מודול מסוים (למשל: Student) תעבוד על branch נפרד העבודה על כל מודול תתבצע באופן הבא:

- פתחו branch נפרד לכל מודול ב repository הכללי של הפרויקט עם שם המודול.
- לדוגמה, אם אתם עובדים על מודול "Student", פתחו branch בשם module/Student.

2. יצירת branch לכל קומפוננטה:

- בכל פעם שאתם מתחילים עבודה על קומפוננטה חדשה בתוך המודול, פתחו branch חדש מתוך ה branch של המודול עם שם הקומפוננטה.
- לדוגמה, אם אתם עובדים על קומפוננטה בשם "StudentList", פתחו branch בשם module/Student/feature/StudentList.

source

3. עבודה על הקומפוננטה:

- בצעו את כל השינויים והפיתוחים הנדרשים בתוך ה branch של הקומפוננטה.
- ודאו שאתם מבצעים commit ים תכופים עם הודעות ברורות ומפורשות שמתארות את השינויים שביצעתם.
- לשים לב להתעדכן מידי פעם מה- branch של המודול כמה פעמים ביום!

# שלב ד': ביצוע ופיתוח

## משימות ב-Angular :

#	החלק הסופי במוצר:	הטכנולוגיה	השפה	תיאור המשימה
4.	אינטגרציה של <b>WebSocket</b> ב- <b>Angular</b> לניהול מסך שיעורים בזמן אמת באמצעות הודעות מ- <b>Kafka</b>	Angular	Ts ,Angular	<p>במסגרת הפרויקט לקחתי על עצמי את האחראיות לפיתוח מערכת לניהול שיעורים בזמן אמת באמצעות <b>WebSocket</b> ב-<b>Angular</b> .</p> <p>המערכת נועדה לספק למשתמשים (המורות) עדכונים מיידיים אודות שינויים בשיעורים על פי הודעות המתקבלות מהשרת, אשר מאזין לאירועים בתור <b>Kafka</b> .</p> <p>המסך קיים כדי:</p> <p>-לאפשר למורות להיכנס דרכו לשיעור שהם מנהלות עכשיו ולסמן בו נוכחות.</p> <p>-לאפשר למזכירות לדעת איזה שיעורים עתידיים למחר ולהכין את הנצרך לאותם שיעורים...</p>

## פירוט משימה 4:

- **רקע והצורך:**  
בפרויקט זה נדרש פתרון שמאפשר עדכונים בזמן אמת ללקוחות על שינויים בשיעורים או בפעילויות שקשורות בהם. הפתרון היה חייב להיות אינטראקטיבי ויעיל, כדי למנוע צורך בטעינה מחדש של המידע ולהגביר את חוויית המשתמש.
- **פתרון יישום:**  
  - \* **פיתוח WebSocket ב-Angular :**  
יישמתי ב-Angular רכיב **WebSocket** המתקשר עם השרת בזמן אמת. הרכיב מאזין להודעות מהשרת ומעדכן את המשתמשים במידע חדש אודות השיעורים בצורה מיידית וללא צורך בריענון הדף. השימוש ב-**WebSocket** מאפשר תקשורת דו-כיוונית בין הלקוח לשרת, ומבטיח שהמשתמשים יקבלו עדכונים בהקדם האפשרי על שינויים או פעילויות בשיעורים.
  - \* **האזנה להודעות מהשרת באמצעות BehaviorSubject :**  
ב-Angular השתמשתי ב-**BehaviorSubject** כדי לנהל את ההאזנה להודעות מ-**WebSocket. BehaviorSubject** הוא סוג של **Observable** שמחזיק את הערך האחרון ומפיץ אותו למנויים חדשים מיד כשהם נרשמים. יצרתי מנגנון המנפיק **BehaviorSubject** המחובר ל-**WebSocket**, כך שהמשתמשים מקבלים עדכונים בזמן אמת. כאשר מתקבלת הודעה חדשה מהשרת, ה-**BehaviorSubject** מעדכן את כל המנויים שלו ומבצע את הפעולות הדרושות, כגון עדכון תצוגת הקורסים או הצגת התראה למשתמש. השימוש ב-**BehaviorSubject** מאפשר ניהול חלק ויעיל של עדכונים בזמן אמת ומספק פתרון פשוט לתקשורת בין רכיבי ה-UI לבין המידע המעודכן.
  - \* **אינטגרציה עם Kafka בצד השרת:**  
השרת, שנכתב ב-Java Spring Boot, מאזין לאירועים בתור **Kafka**. כאשר מתקבלת הודעה חדשה בקשר לשיעורים, השרת שולח את המידע המעודכן ללקוחות המחוברים דרך **WebSocket**. תהליך זה מבטיח שהעדכונים הנדרשים יועברו בצורה מהירה ויעילה לכל המשתמשים, מה שמפחית את הצורך בתקשורת מוגברת וטעינה חוזרת של המידע.

Complexity is 4 Everything is cool!

```
ngOnInit(): void {  
  
    this.websocketService.getMessageStart()?.subscribe(message => {  
        const data = JSON.parse(message.body);  
  
        const lesson: Classes = this.createClassFromData(data);  
        this.teacherservice.addClass(lesson);  
    });  
    this.websocketService.getMessageFinish()?.subscribe(message => {  
        const data = JSON.parse(message.body);  
        this.teacherservice.removeClass(data.classId);  
    });  
    this.websocketService.getMessageFutureClass()?.subscribe(message => {  
        const data = JSON.parse(message.body);  
        const futureClass: Classes = this.createClassFromData(data);  
        this.teacherservice.addFutureClass(futureClass);  
    });  
}
```

קומפוננטת app היא הקומפוננטה הראשית בפרויקט, והיא עולה ראשונה בעת הרצת הפרויקט. הפונקציה ngOnInit של קומפוננטה זו מתבצעת מיד עם עלייתה, מאזינה להודעות WebSocket על תחילת, סיום ושיעורים עתידיים, יוצרת ומוסיפה שיעורים חדשים על פי הודעות תחילת ושיעורים עתידיים, ומסירה שיעורים לפי הודעות סיום.

Complexity is 5 Everything is cool!

```
createClassFromData(data: any): Classes {  
    const { classId, courseId, className, classDescription, date, startTime, endTime } = data;  
  
    const dateObj = new Date(date[0], date[1] - 1, date[2]);  
    const startTimeObj = new Date(date[0], date[1] - 1, date[2], startTime[0], startTime[1], startTime[2], startTime[3] / 1000000);  
    const endTimeObj = new Date(date[0], date[1] - 1, date[2], endTime[0], endTime[1], endTime[2], endTime[3] / 1000000);  
  
    return new Classes(  
        classId.toString(),  
        courseId.toString(),  
        className,  
        classDescription,  
        dateObj,  
        startTimeObj,  
        endTimeObj  
    );  
}
```

המתודה **createClassFromData** ממירה נתוני JSON לאובייקטים מסוג Classes

# WebSocketService

המחלקה אחראית על ניהול חיבור WebSocket לשרת STOMP, שמאפשר תקשורת בזמן אמת בין הלקוח לשרת. היא מגדירה את החיבור לשרת WebSocket ומבצעת מנויים לשלושה נושאים שונים, כך שהיא יכולה לקבל ולהפיץ הודעות בהתאם. כאשר מתקבלת הודעה, היא מפיצה אותה למנויים המתאימים דרך Subject. מתרחשת שגיאה במהלך החיבור, היא מציגה את פרטי השגיאה ומבצעת ניסיון להתחבר מחדש לאחר זמן קצר.

```
import { Injectable } from '@angular/core';
import { Observable, Subject } from 'rxjs';
import { Client, Message } from '@stomp/stompjs';

@Injectable({
  providedIn: 'root'
})
export class WebSocketService {
  private client: Client;
  private messageSubject: Subject<Message> = new Subject<Message>();
  private messageEndClass: Subject<Message> = new Subject<Message>();
  private messageFutureClass: Subject<Message> = new Subject<Message>();

  constructor() {
    this.client = new Client({
      brokerURL: 'ws://localhost:8081/ws',
      connectHeaders: {},
      debug: function (str) {
        console.log(str);
      },
      reconnectDelay: 5000,
      heartbeatIncoming: 10000,
      heartbeatOutgoing: 10000,
```

```
      this.client.onConnect = (frame) => {
        console.log('Connected: ' + frame);
        this.client.subscribe('/topic/socket_lesson_start', (message) => {
          this.messageSubject?.next(message);
          console.log('Received message: ', message.body);
        });
        this.client.subscribe('/topic/future_lessons', (message) => {
          this.messageFutureClass?.next(message);
          console.log('Received message: ', message.body);
        });
        this.client.subscribe('/topic/socket_lesson_end', (message) => {
          this.messageEndClass?.next(message);
          console.log('Received message: ', message.body);
        });
      };

      this.client.onStompError = (frame) => {
        console.error('Broker reported error: ' + frame.headers['message']);
        console.error('Additional details: ' + frame.body);
        // Schedule reconnection
        setTimeout(() => {
          0

```

```
        this.client.activate();
      }, 5000);
    };

    this.client.activate();
  }

  public getMessagesStart(): Observable<Message> {
    return this.messageSubject.asObservable();
  }

  public getMessagesFinish(): Observable<Message> {
    return this.messageEndClass.asObservable();
  }

  public getMessagesFutureClass(): Observable<Message> {
    return this.messageFutureClass.asObservable();
  }
}
```



```

export class TeacherService {
  private classesSubject = new BehaviorSubject<Classes[]>(this.loadClassesFromLocalStorage());
  classes$ = this.classesSubject.asObservable();

  private futureClassesSubject = new BehaviorSubject<Classes[]>(this.loadFutureClassesFromLocalStorage());
  futureClasses$ = this.futureClassesSubject.asObservable();

  constructor() { }
  Complexity is 3 Everything is cool!
  private loadFutureClassesFromLocalStorage(): Classes[] {
    const storedFutureClasses = localStorage.getItem('futureClasses');
    return storedFutureClasses ? JSON.parse(storedFutureClasses) : [];
  }

  private saveFutureClassesToLocalStorage(classes: Classes[]) {
    localStorage.setItem('futureClasses', JSON.stringify(classes));
  }
}

```

```

addFutureClass(newClass: Classes) {
  const futureClasses = this.futureClassesSubject.value;
  if (!futureClasses.some(cls => cls.classId === newClass.classId)) {
    const updatedFutureClasses = [...futureClasses, newClass];
    this.futureClassesSubject.next(updatedFutureClasses);
    this.saveFutureClassesToLocalStorage(updatedFutureClasses);
  }
}

```

BehaviorSubject הוא סוג של Subject ב-RxJS המאפשר לשמור ערך אחרון ולשלוח אותו לכל מנוי חדש מיד לאחר הצטרפותו. זה שימושי כאשר יש צורך לשמור ולשלוח מצב נוכחי או ערך מעודכן למנויים חדשים. ב-TeacherService, BehaviorSubject משמשים לניהול ולעדכון של רשימות שיעורים נוכחיים ושיעורי עתיד. הם שומרים את הערכים הנוכחיים ומוודאים שמנויים חדשים יקבלו את המידע המעודכן מיד לאחר הצטרפותם.



# התהליך הכולל (סיכום קודים שקופיות קודמות)

קבלת הודעות: WebSocket: כאשר הודעה מתקבלת מהשרת דרך WebSocket, WebSocketService שולח את ההודעה המתאימה ל-Subject הרלוונטי.

עדכון: TeacherService אם ישנם שינויים בשיעורים הנוכחיים או העתידיים שמגיעים מהודעות WebSocket, TeacherService מעדכן את ה-BehaviorSubject המתאים. השיעורים מעודכנים באחסון המקומי של הדפדפן.

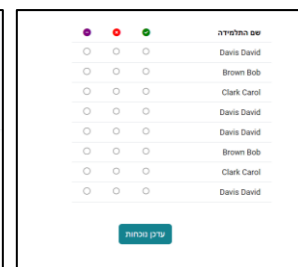
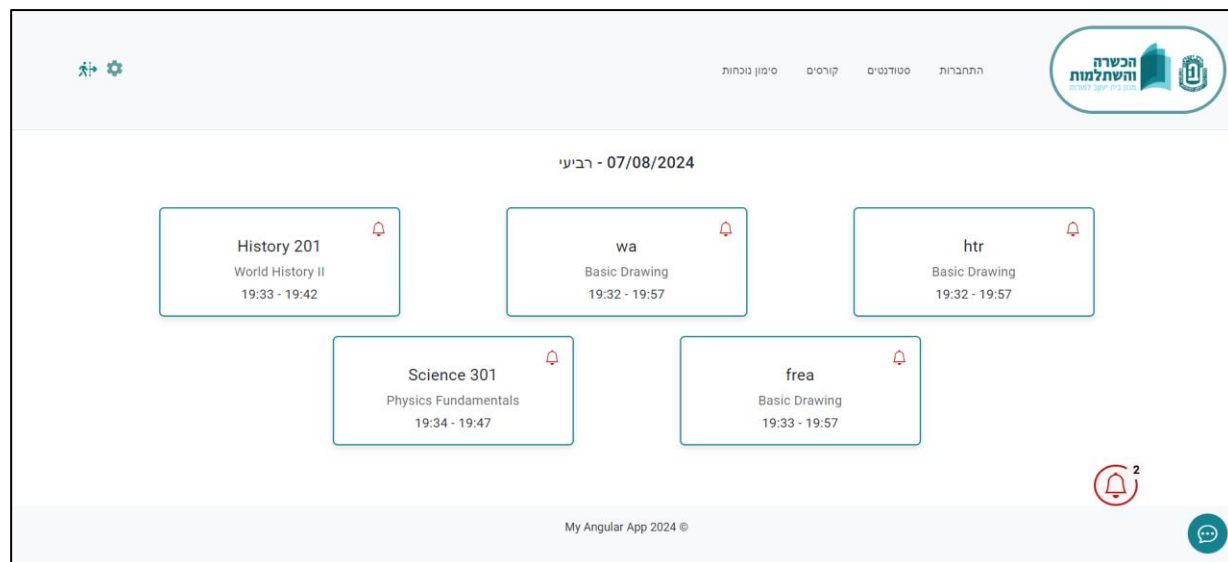
פלט למרכיבים: המרכיבים השונים באפליקציה מאזינים ל-Observable של TeacherService ומגיבים לשינויים בשיעורים הנוכחיים והעתידיים, מה שמאפשר להם להציג את המידע המעדכן בזמן אמת למשתמש.

# צילומי מסך של משימת סגירת קורסים ב-Angular

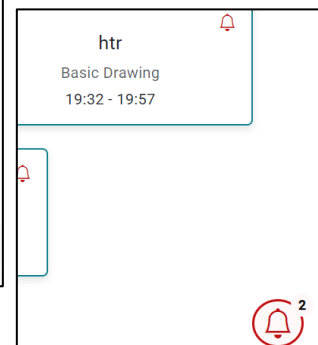
## • תוצאות:

היישום של **WebSocket** ב-Angular, בשילוב עם אינטגרציה עם **Kafka** בצד השרת, הביא לשיפור משמעותי בחוויית המשתמש. המשתמשים קיבלו עדכונים מידיים ושינויים בקורסים בלא צורך בריענון הדף, מה שתרם לשיפור ביצועים ולתגובה מהירה לכל שינוי. השימוש ב-**Observable** אפשר ניהול יעיל של הודעות בזמן אמת והגביר את גמישות המערכת. הפתרון גם הפחית את העומס על השרת, מכיוון שעדכונים נשלחים בצורה פרואקטיבית ולא לפי בקשה, מה שהפך את המערכת ליעילה ומוזרנית יותר.

(מסך שיעורים בזמן אמת פעמון אדום מראה שעדיין לא סימנו נוכחות לשיעור זה)



(כשלוחצים על שיעור מסוים הוא מביא אותך לדף נוכחות של כל התלמידות של אותו שיעור)




(פעמון זה מופיע אם קיים שיעור שנגמר ולא סימנו לו לנכחות העיגול עם המספר מראה כמה שיעורים כאלה יש כשלוחצים עליו רואים את ההודעות.)

# שלב ד': ביצוע ופיתוח

## משימות ב-Angular :

#	החלק הסופי במוצר:	הטכנולוגיה	השפה	תיאור המשימה
5.	סידור ה-מודול Student ב-Angular ופיתוח קומפוננטת- search	Angular	Ts ,Angular	אני הייתי אחראית על המסך-מודול – Student על ה-Routes שלו וחילוקו לקומפוננטות . בנוסף פיתחתי את קומפוננטת search לחיפוש תלמידים הקומפוננטה הראשית למודול זה .

## משימות ב-Angular : משימה 5 :



הכשרה והשתלמות

מנהל רשות מקומית

התחברות סטודנטים קורסים סימון נוכחות

העלאת קובץ אקסל להוספת תלמידות

שם פרטי:

שם משפחה:

תעודת זהות:

שירה

חיפוש

נמצאו 2 סטודנטים :

שם פרטי	שם משפחה	ת.ז.
שירה	שלום	324565778
שירה	כרמי	324565778

My Angular App 2024 ©

שם פרטי:

שם משפחה:

תעודת זהות:

שירה

כרמי

חיפוש

נמצאו 1 סטודנטים :

שם פרטי	שם משפחה	ת.ז.	פרטים נוספים
שירה	כרמי	324565778	

# שלב ד': ביצוע ופיתוח

## משימות ב-Angular 5 : משימה 5 :

```

</button>
<div class="container mt-4">
  <form #searchForm="ngForm" class="w-75 mx-auto">
    <div class="row g-3">
      <div class="col-md-4">
        <label for="first_name" class="form-label">שם פרטי:</label>
        <input type="text" class="form-control" id="first_name" name="first_name"
          [(ngModel)]="studentSelector.firstName" #firstName="ngModel" minlength="2" required>
        <ng-container *ngIf="firstName.invalid && (firstName.value!=null)">
          <div class="text-danger">Please enter a valid first name.</div>
        </ng-container>
      </div>
      <div class="col-md-4">
        <label for="last_name" class="form-label">שם משפחה:</label>
        <input type="text" class="form-control" id="last_name" name="last_name"
          [(ngModel)]="studentSelector.lastName" #lastName="ngModel" minlength="2" required>
        <ng-container *ngIf="lastName.invalid && (lastName.value!=null)">
          <div class="text-danger">Please enter a valid last name.</div>
        </ng-container>
      </div>
      <div class="col-md-4">
        <label for="student_id" class="form-label">מספר זהות:</label>
        <input type="text" class="form-control" id="student_id" name="student_id"
          [(ngModel)]="studentSelector.studentId" #studentId="ngModel" pattern="[0-9]{9}">
        <ng-container *ngIf="studentId.invalid && (studentId.value!=null)">
          <div class="text-danger">Please enter a valid 9-digit student ID.</div>
        </ng-container>
      </div>
    </div>
  </div>
</div>

```

(הקומפוננטה option מצטיירת באשר studentsSelector אצלי מלא הוא מכניס ל-  
 סלקטור student בקומפוננטת option את הנתונים ואז הקומפוננטה מוצגת עם כל  
 תוצאות החיפוש שקיימות).

```

<button type="submit" class="btn btn-primary mt-3"
  (click)="search(studentId.value, lastName.value, firstName.value)"
  [disabled]="firstName.invalid&&lastName.invalid&&studentId.invalid">חיפוש</button>
</form>
</div>
<app-options [students]="studentSelector"></app-options>

```

```

export class SearchComponent implements OnInit {
  constructor(private studentService: StudentService, private router: Router) {
    const type: string = "gggg-r0003";
  }
  studentsSelector: Student[] = [];
  student: Student | undefined;
  studentSelector: Student = {
    "firstName": undefined, "lastName": undefined,
    studentId: undefined
  };
  ngOnInit(): void {
  }
  uploadFile() {
    this.router.navigate(['students/studentsExcel'])
  }

  isValidIsraeliID(id: string): boolean { ...
  }

  search(id: string, lastName: String, firstName: String) {
    const obj: StudentSearch = {"studentId": Id, "firstName": firstName, "lastName": lastName};
    // alert("insert function on search")
    this.studentsSelector = this.studentService.getStudentBySearch(obj);

    this.studentService.getStudentById(Id).subscribe(
      (data) => { ...
      },
      (error) => { ...
      });
  }
}

```

## שלב ה': בדיקות אבטחה ואיכות

כאשר סיימתי לפתח, הרצתי את האתר בסביבות שונות:

1. לוקאלית

2. על הענן render

בדקתי שהמסכים עובדים כראוי, במידה והמשימה הייתה על צד קליינט, ובמידה והמשימה הייתה על צד סרבר, בדקתי שה apis מגיבים כראוי באמצעות postman.

## שלב ו': הטמעה ותמיכה

בסיום כל ספרינט נערך דמו ללקוח, בסביבת ענן.  
אנשי הקשר של הלקוח קבלו את הלינק לאתר להכרות עם המערכת ונתינת פידבק לשינויים נדרשים בספרינטים הבאים.  
בשל המוגבלות הפיננסית של החברה הוחלט להעלות את האפליקציה ל  
render-cloud  
בעתיד הצוותים הבאים יפעילו את המערכת בענן חזק יותר ויטמיעו את  
המערכת באגף לשימוש יומיומי.

## סיכום ומסקנות:

### סיכום כללי:

הפרויקט נחל הצלחה בזכות עבודה משותפת של הצוות, שימוש בטכנולוגיות מתקדמות וארכיטקטורת מיקרו-סרוויסים מבוצרת. המערכת מאפשרת ניהול קורסים, שיעורים ומסמכים בצורה יעילה ונוחה, תוך שמירה על חוויית משתמש גבוהה. הפרויקט הושק בהצלחה לענן וממשיך להיות מתוחזק ומשודרג באופן שוטף.

### מסקנות אישיות:

כמפורט בפרק שמונה- צברתי ידע רב וחדש בטכנולוגיות חדשניות וחכמות, למדתי נהלי עבודה במתודולוגיית אדג'ייל (Agile), התמודדתי עם אתגרים בפיתוח ופתרתי באגים שונים במהלך הפיתוח.

### תרומה לפרויקט ולחברה:

הייתי אחראית לתשתית ה-DB לכל אורך הפרויקט בנוסף למשימות פיתוח שונות בכל רחבי האפליקציה, אפיונים נוספים שנדרשו בעקבות בקשות חדשות של הלקוח. ובנוסף, הכוונתי את הצוות להתנהלות נכונה עם ה- Github



## נספחים

•

אני החלטתי שיהיה db-connector אחד כי :

רציתי לחסוך בהוצאות

התקציב שלי מוגבל ורציתי לצמצם את העלויות של תחזוקת מסדי נתונים מרובים, שימוש במסד נתונים אחד הוא פתרון חסכוני יותר. זה מפשט את ניהול התשתית ומפחית את הצורך בניהול מספר מאגרי נתונים.

אבל עדיין נשארתי בארכיטקטורת מיקרו סרויסים לשאר הסרויסים בשביל להנות מיתרונות כמו: סקלרוביליות, אפשרות לכתוב את צד הסרבר בכמה שפות - ( פייטון, גאווה כל סרויס מה שמתאים לו בינה מלאכותי ת רצה טוב עם פייטון וכו )