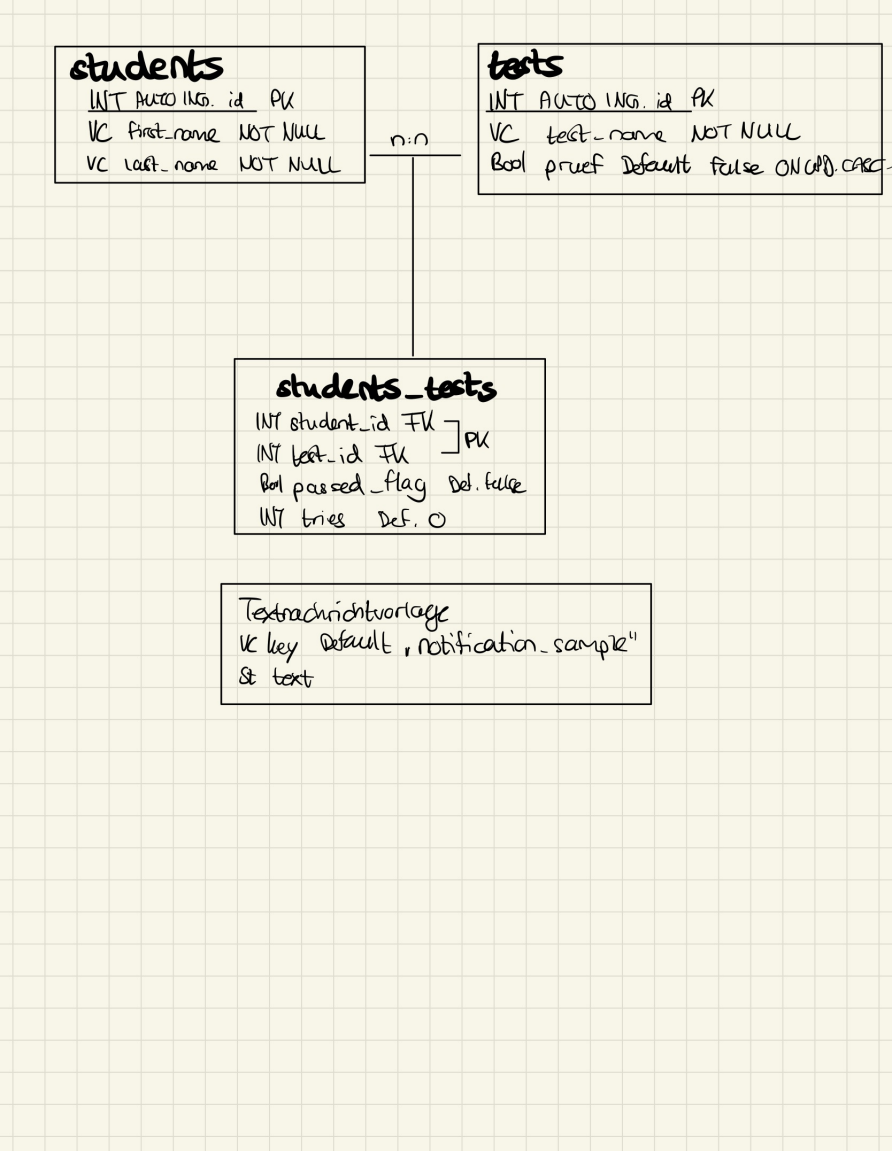
Job Application Documentation

Projekt erstellen mit Spring Initializer

https://start.spring.io/

- Maven
- Language Java
- Spring Version 3.1.4.
- Java 21
- Dependencies:
    - Spring Boot DevTools
    - Spring Web (REST)
    - Spring Data JPA (SQL)
    - MySQL Driver

Step 1 - Datenmodell vorbereiten



Key Relation: Student n: - Students_Tests - :n Test

Schema.sql

```
CREATE TABLE notification_sample (
    notification_name VARCHAR(300) PRIMARY KEY DEFAULT "notification_sample",
    text_sample TEXT
);

CREATE TABLE students (
    id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(300) NOT NULL CHECK (LENGTH(first_name) > 0),
    last_name VARCHAR(300) NOT NULL CHECK (LENGTH(last_name) > 0)
);

CREATE TABLE tests (
    id INT PRIMARY KEY AUTO_INCREMENT,
    test_name VARCHAR(300) NOT NULL CHECK (LENGTH(test_name) > 0),
    pruef BOOLEAN DEFAULT false
);

CREATE TABLE students_tests (
```

```
    student_id INT REFERENCES students (id) ON DELETE CASCADE,
    test_id INT REFERENCES tests (id) ON DELETE CASCADE,
    passed_flag BOOLEAN DEFAULT false,
    tries INT CHECK (tries < 4) DEFAULT 0,
    PRIMARY KEY (student_id, test_id)
);
```

Data.sql

```
INSERT INTO
    students (first_name, last_name)
VALUES
    ("Jörn", "Bachmeier"),
    ("Nikolai", "Killer"),
    ("Stefan", "Sieber"),
    ("Eli", "Else"),
    ("Henrik", "Henrikson"),
    ("Karl", "Alles"),
    ("Tobias", "Abel");

INSERT INTO
    tests (test_name)
VALUES
    ("Big Data"),
    ("Software Engineering"),
    ("Logik"),
    ("NLP"),
    ("Gestalte, Genese und Info"),
    ("Physik"),
    ("Chemie"),
    ("Biologie");

INSERT INTO
    notification_sample (text_sample)
VALUES
    ("Lieber $vorname $nachname,\n Sie haben den Test $testname $ergebnis.\n Liebe Grüße ihr TypeFlowTeam.");

INSERT INTO
    students_tests (student_id, test_id, passed_flag, tries)
VALUES
    (1, 3, True, 1), (1, 4, False, 2), (1, 5, True, 1), (1, 7, False, 3), (1, 8, False, 3),
    (2, 1, False, 1), (2,6, True, 2), (2, 7, False, 1), (2, 8, True, 1),
    (3, 2, False, 1), (3, 4, True, 1), (3, 6, True, 1), (3, 7, False, 3),
    (4, 1, True, 1), (4, 3, True, 1), (4, 4, False, 3),
    (5, 1, False, 2), (5, 2, False, 1), (5, 3, False, 3), (5, 6, False, 2), (5, 7, True, 2), (5, 8, True, 1),
    (7, 2, True, 1), (7, 3, True, 1), (7, 5, False, 3), (7, 6, True, 1);

INSERT INTO
    students_tests (student_id, test_id)
VALUES
    (1, 1), (1, 2), (2, 2), (2, 3), (3, 1), (4, 8), (6, 1), (6, 3), (6, 5), (7, 7), (7, 8);
```

All Entities

```
@Entity
@Table(name = "notification_sample")
public class MessageSample {

    @Id
    private String notification_name;
    private String text_sample;
}

@Entity
@Table(name = "students")
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String first_name;
```

```
    private String last_name;

    @JsonIgnore
    @OneToMany(
        mappedBy = "student",
        cascade = CascadeType.ALL,
        orphanRemoval = true
    )
    private List<StudentTest> tests = new ArrayList<>();
}

@Entity
@Table(name = "tests")
public class Test {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String test_name;
    private Boolean pruef;

    @JsonIgnore
    @OneToMany(
        mappedBy = "test",
        cascade = CascadeType.ALL,
        orphanRemoval = true
    )
    private List<StudentTest> students = new ArrayList<>();
}

@Embeddable
public class StudentTestId implements Serializable{

    @Column(nullable = false, name = "student_id")
    private Integer student_id;
    @Column(nullable = false, name = "test_id")
    private Integer test_id;
}

@Entity
@Table(name = "students_tests")
public class StudentTest{

    @EmbeddedId
    private StudentTestId studentTestId;

    @JsonIgnore
    @ManyToOne(fetch = FetchType.LAZY)
    @MapsId("student_id")
    private Student student;

    @JsonIgnore
    @ManyToOne(fetch = FetchType.LAZY)
    @MapsId("test_id")
    private Test test;

    private Boolean passed_flag;
    private Integer tries;
}
```

Step 2 - REST API erstellen

Für die Rest API werden zunächst die Prozesse in Layer aufgeteilt. Von Modell zu Controller (API Layer) und dann über die Service Class (Business Layer) zum Data Access Layer. In Spring bekommt jede Klasse dafür passende Annotationen.

Für jede Klasse soll es möglich sein Entitäten anzulegen, bearbeiten, löschen und anzuzeigen.

Aufgabe Studenten path = “/api/v1/student”

Student Repository

```
@Query("SELECT s FROM Student s WHERE s.first_name = ?1 and s.last_name = ?2")
public Optional<Student> findStudentByName(String first_name, String last_name);

@Query(value = "SELECT s.first_name, s.last_name, t.test_name, st.passed_flag, st.tries FROM students as s\n" + //
            "INNER JOIN students_tests AS st ON s.id = st.student_id\n" + //
            "INNER JOIN tests AS t ON st.test_id = t.id\n" + //
            "WHERE st.tries = 3", nativeQuery = true)
public List<Object[]> getAllFailedStudents();
```

GET All Students - http://localhost:8080/api/v1/student/all

```
@GetMapping(path = "/all")
public List<Student> fetchAllStudents() { return studentService.getAllStudents(); }

public List<Student> getAllStudents() { return studentRepository.findAll(); }
```

GET Student By Id - http://localhost:8080/api/v1/student/id?student_id=3

```
@GetMapping(path = "/id")
public Optional<Student> getStudentById(@RequestParam Integer student_id) {
    return studentService.getStudentById(student_id);
}

public Optional<Student> getStudentById(Integer student_id) { return studentRepository.findById(student_id); }
```

DELETE Student By Id - http://localhost:8080/api/v1/student/3

```
@DeleteMapping(path = "{student_id}")
public void deleteStudentById(@PathVariable("student_id") Integer student_id) {
    studentService.deleteStudentById(student_id);
}

public void deleteStudentById(Integer student_id) {
    if (!studentRepository.existsById(student_id)) { throw new IllegalStateException("Student does not exist!"); }
        studentRepository.deleteById(student_id);
}
```

POST Add New Student - http://localhost:8080/api/v1/student

```
@PostMapping
public void addNewStudent(@RequestBody Student student) {
    studentService.addNewStudent(student);
}

public void addNewStudent(Student student) {
    Optional <Student> studentOptional = studentRepository.findStudentByName(student.getFirst_name(), student.getLast_name());

    if (studentOptional.isPresent()) { throw new IllegalStateException("Name taken"); }
        studentRepository.save(student);
}
```

```
{
    "first_name": "Simon",
    "last_name": "Seifüßl"
}
```

PUT Update Student Name - http://localhost:8080/api/v1/student/3?first_name=Maria

```
@PutMapping(path = "{student_id}")
public void updateStudent(@PathVariable("student_id") Integer student_id, @RequestParam(required = false) String first_name, @RequestParam(
    studentService.updateStudent(student_id, first_name, last_name);
```

```
}

@Transactional
public void updateStudent(Integer student_id, String first_name, String last_name) {
    Student student = studentRepository.findById(student_id).orElseThrow(() -> new IllegalStateException("Student does not exist"));

    if ((first_name != null && first_name.length() > 0
            && !Objects.equals(student.getFirst_name(), first_name))
            || (last_name != null && last_name.length() > 0
            && !Objects.equals(student.getLast_name(), last_name)))
        {
        Optional<Student> studentOptional = studentRepository.findStudentByName(first_name, last_name);
        if (studentOptional.isPresent()) { throw new IllegalStateException("Name already taken"); }
        if (first_name != null) { student.setFirst_name(first_name); }
        if (last_name != null) { student.setLast_name(last_name); }
    }
}
```

GET All Failed Students

```
@GetMapping(path = "/failed")
public List<Object[]> getAllFailedStudents() {
    return studentService.getAllFailedStudents();
}

public List<Object[]> getAllFailedStudents() {
    return studentRepository.getAllFailedStudents();
}
```

Aufgabe Tests path = “/api/v1/test”

Test Repository

```
@Query("SELECT t FROM Test as t WHERE t.test_name = ?1")
public Optional<Test> findTestByName(String test_name);
```

GET All Tests - http://localhost:8080/api/v1/test/al

```
@GetMapping(path = "/all")
public List<Test> fetchAllTests() { return testService.getAllTests(); }

public List<Test> getAllTests() { return testRepository.findAll(); }
```

GET Test By Id - http://localhost:8080/api/v1/test/id?test_id=2

```
@GetMapping(path = "/id")
public Optional<Test> getTestById(@RequestParam Integer test_id) {
    return testService.getStudentById(test_id);
}

public Optional<Test> getStudentById(Integer test_id) { return testRepository.findById(test_id); }
```

DELETE Test By Id - http://localhost:8080/api/v1/test/9

```
@DeleteMapping(path = "{test_id}")
public void deleteTestById(@PathVariable("test_id") Integer test_id) {
    testService.deleteTestById(test_id);
}

public void deleteTestById(Integer test_id) {
    if (!testRepository.existsById(test_id)) { throw new IllegalStateException("Test does not exist!"); }
        testRepository.deleteById(test_id);
}
```

POST Add New Test - http://localhost:8080/api/v1/test

```
@PostMapping
public void addNewTest(@RequestBody Test test) { testService.addNewTest(test); }

public void addNewTest(Test test) {
    Optional <Test> testOptional = testRepository.findTestByName(test.getTest_name());

    if (testOptional.isPresent()) { throw new IllegalStateException("Name taken"); }
        testRepository.save(test);
}
```

```
{
    "test_name": "IT"
}
```

PUT Update Test By Name - http://localhost:8080/api/v1/test/3?test_name=Maria

```
@PutMapping(path = "/pruef")
public void updatePruef(@RequestParam Integer test_id) {
    testService.updatePruef(test_id);
}

@Transactional
public void updateTest(Integer test_id, String test_name) {
    Test test = testRepository.findById(test_id).orElseThrow(() -> new IllegalStateException("test does not exist"));

    if (test_name != null && test_name.length() > 0
            && !Objects.equals(test.getTest_name(), test_name)) {
        Optional<Test> testOptional = testRepository.findTestByName(test_name);
        if (testOptional.isPresent()) { throw new IllegalStateException("Name already taken"); }
        test.setTest_name(test_name);
    }
}
```

Aufgabe Student hat Test bestanden

Student Test Repository

```
@Query(value = "SELECT st.passed_flag FROM students_tests as st\n" + //
            "INNER JOIN tests AS t ON st.test_id = t.id\n" + //
            "INNER JOIN students AS s ON st.student_id = s.id\n" + //
            "WHERE t.id = ?2 and s.id = ?1 and st.passed_flag = true", nativeQuery = true)
public Boolean hasPassed(Integer student_id, Integer test_id);
```

GET Student Passed Test - http://localhost:8080/api/v1/st_combination/hasPassed?student_id=1&test_id=3

```
@GetMapping(path = "/hasPassed")
public Boolean hasPassed(@RequestParam Integer student_id, @RequestParam Integer test_id) {
    return studentTestService.hasPassed(student_id, test_id);
}

public Boolean hasPassed(Integer student_id, Integer test_id) {
    if (studentTestRepository.hasPassed(student_id, test_id) == null) { return false; }
    return true;
}
```

Step 3 - Textnachrichtvorlage

Eintrag in der Datenbank anlegen

Wird in der Data.sql Datei gemacht…

“Lieber **$vorname $nachname**,
Sie haben den Test **$testname $ergebnis**.
Liebe Grüße ihr TypeFlowTeam.”

Sende Benachrichtigung mit Prüfungsergebnis

☐ Wenn Button “bestanden / nicht bestanden” gedrückt wird, dann soll dieser Text mit den nötigen Informationen abgeschickt werden (E-Mail)

- Es Darf nur eine Benachrichtigung ankommen pro bestanden / nicht bestanden
- Es muss geprüft werden, ob die Nachricht angekommen ist (Response checken)

Step 4 - Nachrichtenvorlage anpassen path = “/api/v1/messagetemplate”

Die Entity MessageSample hat keinen Integer Index als Key, sondern einen String. Dadurch funktioniert sie wie eine Collection aus Key-Text Paaren. Solang in der Anwendung nur der “notification_sample” Key steht, kann der Text beliebig oft ausgetauscht werden, ohne dass etwas in der Anwendung geändert werden soll.

MessageSample Repository

```
@Query(value = "SELECT n.text_sample FROM notification_sample n WHERE n.notification_name=?1", nativeQuery = true)
public String getTemplateByKey(String key);

@Query(value = "SELECT * FROM notification_sample n WHERE n.notification_name=?1", nativeQuery = true)
public MessageSample getMessageSampleByKey(String key);
```

PUT Update NotificationSample & GET Notification Text - http://localhost:8080/api/v1/messagetemplate/notification_sample

```
@GetMapping(path = "{key}")
public String getTemplateByString(@PathVariable("key") String key) {
    return messageSampleService.getMessageTemplate(key);
}

@PutMapping(path = "{key}")
public void updateTemplate(@PathVariable("key") String key, @RequestBody String text) {
    messageSampleService.updateTemplate(text, key);
}

public String getMessageTemplate(String key) {
    return messageSampleRepository.getTemplateByKey(key);
}

public void updateTemplate(String text, String key) {
    MessageSample ms = messageSampleRepository.getMessageSampleByKey(key);
    ms.setText_sample(text);
}
```

```
{
    "notification_name": "notification_sample",
    "text_sample": "Lieber $vor $nach,\n Sie haben den Test $testname $ergebnis.\n Liebe Grüße ihr TypeFlowTeam."
}
```

Step 5 - Probetests

Durch die Limitierung der Versuche eines Studentens auf 3 und der Möglichkeiten Probeklausuren zu schreiben, ergeben sich folgende Bedingungen…

Aufgabe Test ändern zur Prüfung

Mit dieser Methode wird der Test nur zu einer Prüfung eingeschaltet (Pruef = true) oder wieder zurück. Funktioniert wie ein Schalter.

PUT Update Test Pruef - http://localhost:8080/api/v1/test/pruef?test_id=1

```
@PutMapping(path = "/pruef")
public void updatePruef(@RequestParam Integer test_id) {
    testService.updatePruef(test_id);
}

@Transactional
public void updatePruef(Integer test_id) {
    Test test = testRepository.findById(test_id).orElseThrow(() -> new IllegalStateException("test does not exist"));
    test.setPruef(!test.getPruef());
}
```

Aufgabe Tries eines Studenten hochzählen

Die Tries eines Studenten zu einer Prüfung kann nur hochgezählt werden, wenn die Tries nicht schon höher als 3 sind, die Prüfung eine richtige Klausur ist und der Stundent nicht bereits bestanden hat (Passed_flag = false). Wenn sich ein Student für die Prüfung anmeldet, dann werden die Tries erst hochgezählt und nicht erst nach Bestnaden/Nicht-Bestanden. Das hat damit zu tun, dass die Tries auf 0 sind, sollte der Student das Fach nicht gewählt haben.

PUT Count Tries +1 - http://localhost:8080/api/v1/st_combination/addTry?student_id=1&test_id=1

```
@PutMapping(path = "/addTry")
public void addTries(@RequestParam Integer student_id, @RequestParam Integer test_id) {
    studentTestService.addTries(student_id, test_id);
}

@Transactional
public void addTries(Integer student_id, Integer test_id) {
    StudentTest st = studentTestRepository.getCombination(student_id, test_id);
    Test test = testRepository.findById(test_id).orElseThrow(() -> new IllegalStateException("Test does not exist"));

    if (st != null) {
        if (st.getTries() < 4 && test.getPruef() == true) {
            st.setTries(st.getTries() + 1);
        } else { new IllegalStateException("Test is not an exam or you already tried this test 3 times"); }
    }
}
```

Aufgabe Student besteht einen Test

Sollte der Student angemeldet sein (Tries > 0), noch nicht zu viele Versuche gebraucht haben (Tries < 4) und die Klausur eine Prüfung gewesen sein (Pruef = true), dann hat der Student bestanden. Es kann ihm auch nicht mehr weggenommen werden.

PUT Update Student hat Bestanden - http://localhost:8080/api/v1/st_combination/passing?student_id=1&test_id=2

```
@PutMapping(path = "/passing")
public void passing(@RequestParam Integer student_id, @RequestParam Integer test_id) {
    studentTestService.passing(student_id, test_id);
}

@Transactional
public void passing(Integer student_id, Integer test_id) {
    StudentTest st = studentTestRepository.getCombination(student_id, test_id);
    Test test = testRepository.findById(test_id).orElseThrow(() -> new IllegalStateException("Test does not exist"));

    if (st != null) {
        if (st.getTries() < 4 && st.getTries() > 0 && test.getPruef() == true) {
            st.setPassed_flag(true);
        } else { new IllegalStateException("You can only pass the exam if your enrolled and your have tried the test less then 3 times"); }
    }
}
```

Step 6 - Daten erhalten

Alle Ergebnisse eines Tests inklusive der Namen der Schüler

GET All Students By Test - http://localhost:8080/api/v1/st_combination/students?t_id=3

```
@Query(value = "SELECT s.first_name, s.last_name, t.test_name, st.passed_flag, st.tries FROM tests as t\n" + //
            "INNER JOIN students_tests AS st ON t.id = st.test_id\n" + //
            "INNER JOIN students AS s ON st.student_id = s.id\n" + //
            "WHERE t.id = ?1", nativeQuery = true)
public List<Object[]> findAllByTest_id(Integer test_id);
```

```
@GetMapping(path = "/students")
public List<Object[]> fatchOneStudentTestCombination(@RequestParam(required = true) Integer t_id) {
    return studentTestService.findAllByTest_id(t_id);
}

public List<Object[]> findAllByTest_id(Integer t_id) {
    return studentTestRepository.findAllByTest_id(t_id);
}
```

Alle Ergebnisse der Tests eines Studenten

GET All Tests By Student - http://localhost:8080/api/v1/st_combination/tests?s_id=3

```
@Query(value = "SELECT s.first_name, s.last_name, t.test_name, st.passed_flag, st.tries FROM students as s\n" + //
            "INNER JOIN students_tests AS st ON s.id = st.student_id\n" + //
            "INNER JOIN tests AS t ON st.test_id = t.id\n" + //
            "WHERE s.id = ?1", nativeQuery = true)
public List<Object[]> findAllByStudent_id(Integer student_id);
```

```
@GetMapping(path = "/tests")
public List<Object[]> fatchOneByTestId(@RequestParam(required = true) Integer s_id) {
    return studentTestService.findAllByStudent_id(s_id);
}

public List<Object[]> findAllByStudent_id(Integer s_id) {
    return studentTestRepository.findAllByStudent_id(s_id);
}
```

Alle Schüler mit allen Prüfungstests, den sie beim dritt versuch nicht bestanden haben

GET All Failed Students - http://localhost:8080/api/v1/st_combination/failed

```
@Query(value = "SELECT s.first_name, s.last_name, t.test_name, st.passed_flag, st.tries FROM students as s\n" + //
            "INNER JOIN students_tests AS st ON s.id = st.student_id\n" + //
            "INNER JOIN tests AS t ON st.test_id = t.id\n" + //
            "WHERE st.tries = 3", nativeQuery = true)
public List<Object[]> getAllFailedStudents();
```

```
@GetMapping(path = "/failed")
public List<Object[]> getAllFailedStudents() {
    return studentTestService.getAllFailedStudents();
}

public List<Object[]> getAllFailedStudents() {
    return studentTestRepository.getAllFailedStudents();
}
```