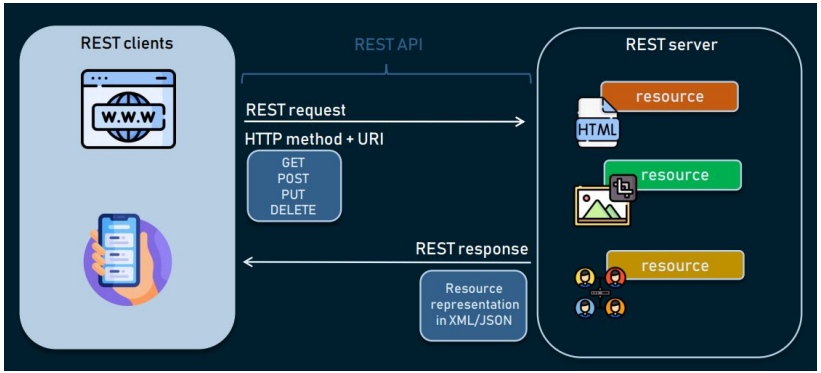
Student Behavior Report- RESTFul API

1. Introduction- The purpose of the project:
2. Overview of the Problem
3. Goals and Requirements: user stories and goals
4. Key Features of the API
5. System Design
6. Key Implementation Details:
   - MySQL Installation and Database creation
   - Developing the API endpoints
   - Demonstration of project
7. Testing
8. Challenges and Tradeoffs
9. Deliverables
10. Conclusion
11. Questions



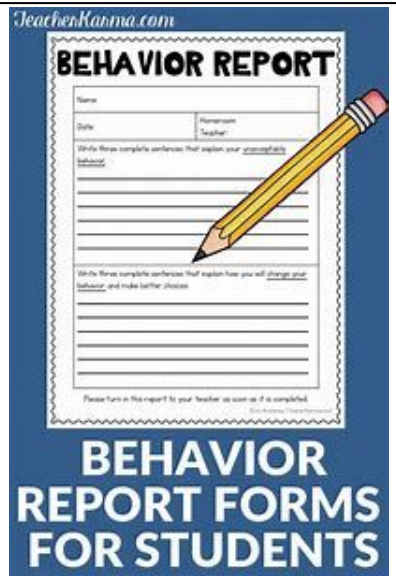
1/11- Introduction

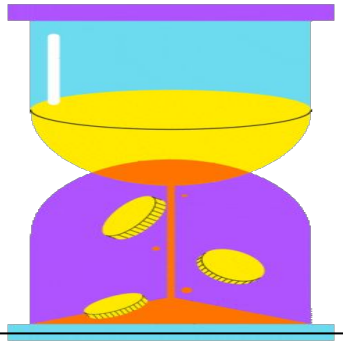
The **purpose** of the project:

- "This project, the Student Behavior Report API, aims to streamline **and** digitise student behavior reporting **between** teachers **and** parents, **focusing on** alternative provision environments."

2/11. Overview of the Problem

- The **problems** identified in the current **manual** system:
  - Time-consuming **and** prone to conflicts.
  - Physical **loss** of data.
  - Challenges in reporting historical data **and** ensuring parents have received updates.





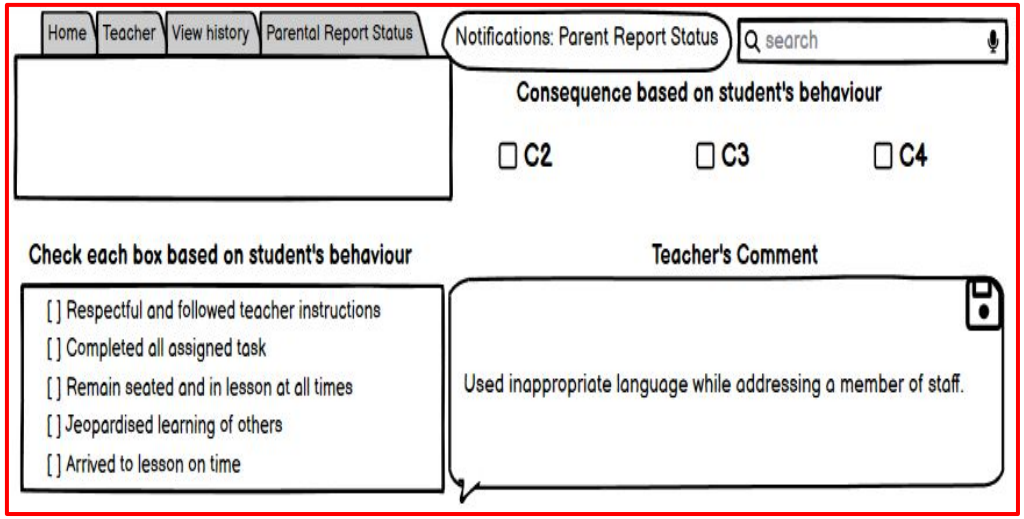
3/11. Goals and Requirements

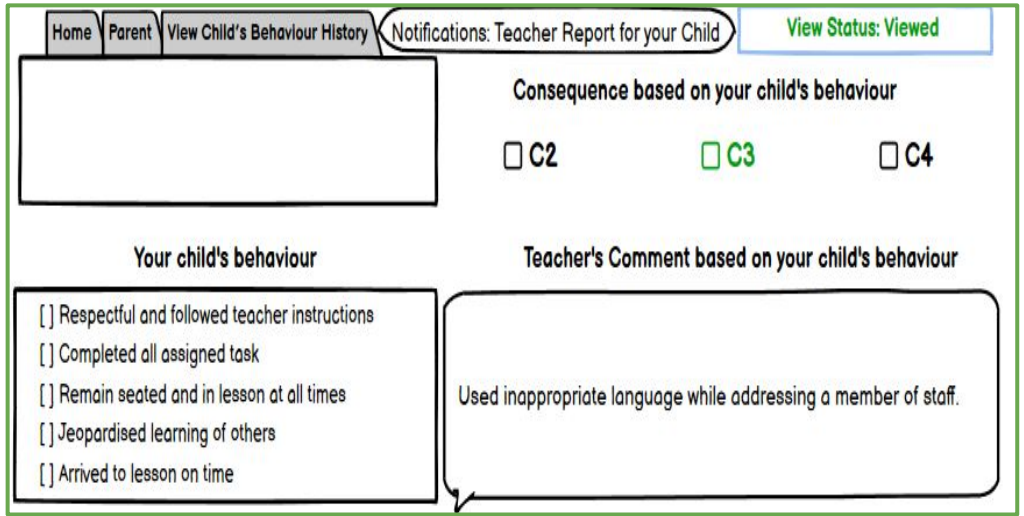
User stories **and** goals:

- Teachers **need to** log student behavior quickly **and** access historical data.
- Parents **require access to** behavior reports **and** notifications for updates.

Non-goals to show clear project scope:

- Include additional student data types **like** parental contact numbers, SEND needs, **as** the main database
- To **handle** academic grading, test scores, **or** other performance metrics, attendance
- The tool will **not** attempt **to** address school-wide data management.





4a/11. Key Features of the API

- CRUD operations for student records.
- Filter students by teacher comments.

Create (Add a New Student)
Method in StudentService.java:

```java
// Method to add a new student - CREATE
public Student addStudent(Student student) {
    return studentRepository.save(student);
}
```

Endpoint in StudentController.java:

```java
@PostMapping("/students")
public Student addStudent(@RequestBody Student student) {
    return studentService.addStudent(student);
}
```

4b/11. Key Features of the API

- CRUD operations for student records.
- Filter students by teacher comments.

Read (Get All Students and Get a Student by ID) Methods in `StudentService.java`:

```java
// Method to find a student by their ID - RETRIEVE
public Student findStudentById(String id) {
    return studentRepository.findById(id).orElse(null);
}
```

```java
// Method to get all students - RETRIEVE
public List<Student> getAllStudents() {
    return studentRepository.findAll();
}
```

Endpoints in `StudentController.java`:

```java
@GetMapping("/students")
public List<Student> getAllStudents() {
    return studentService.getAllStudents();
}
```

```java
@GetMapping("/students/{id}")
public Student getStudentById(@PathVariable String id) {
    return studentService.findStudentById(id);
}
```

5a/11. System Design

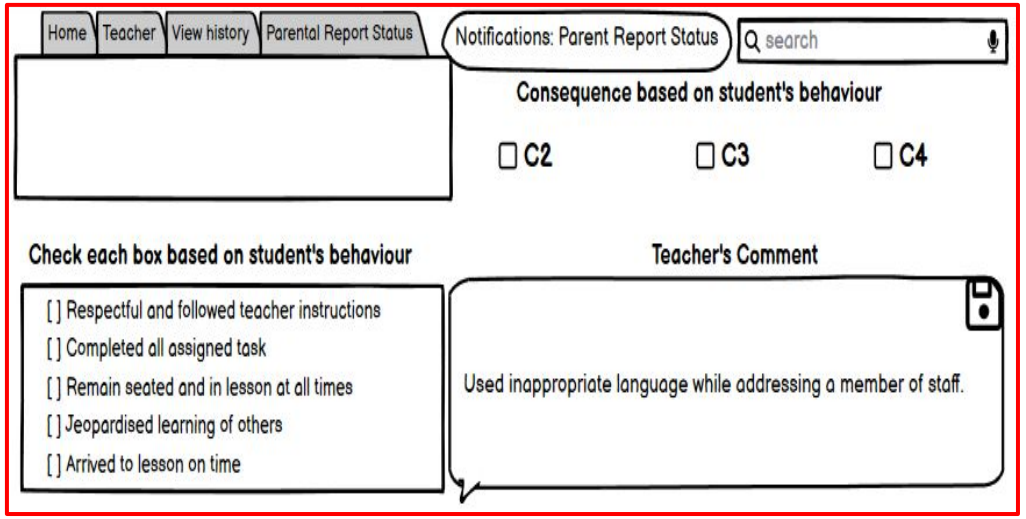
Relationship between Database entities:

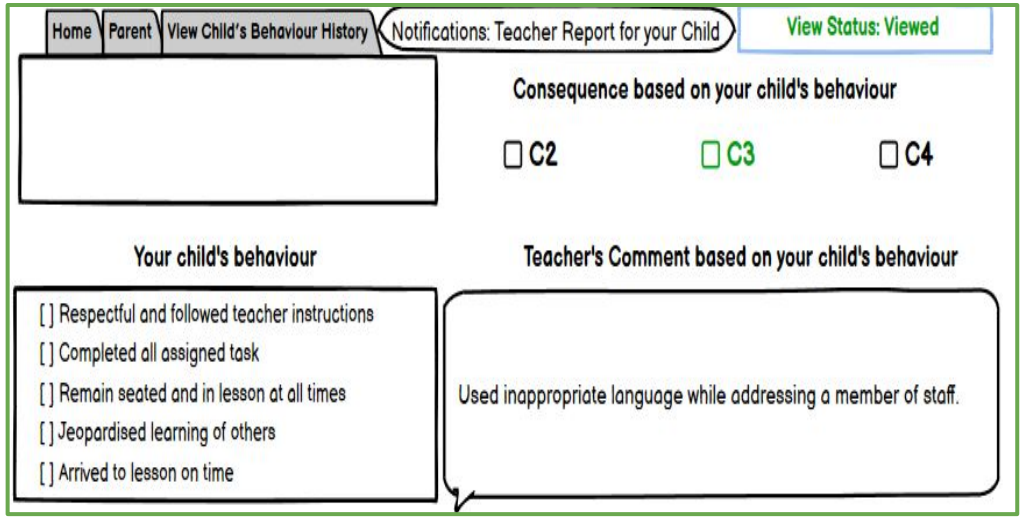
- **One-to-Many Relationship:**
  - A single `Teacher` can **have** many `Students`.

    Example: `Teacher` entity with `@OneToMany` annotation and Student entity with `@ManyToOne` annotation.

  - The `Parent` entity has a one-to-many relationship with the `Student` entity.
  - A Student can have many comments, from the same teacher or from various teachers
- **Many-to-Many Relationship:**
  - A `Teacher` can have many `Students`, and a Student can have many `Teachers`.
  - Example: Both `Teacher` and Student entities with `@ManyToMany` annotation and a join table.

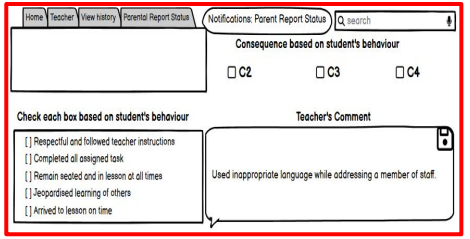
These examples demonstrate how to model the entity relationships between `Teacher`, Student and Parent in a Spring Boot application using JPA annotations. The choice of relationship depends on the specific requirements of this application.





5b/11. System Design

- How the backend and frontend components interact.
- Reference the parent and teacher interface wireframes.



Interaction Flow

1. **User Action:**
   - The user (Teacher/ Data Administrator) performs an action on the frontend (e.g., clicking a button to add a new student).
2. **HTTP Request:**
   - The frontend sends an HTTP request to the backend API endpoint (e.g., `POST /api/v1/students`).
3. **Controller Handling:**
   - The backend controller receives the request and calls the appropriate service method.
4. **Service Processing:**
   - The service method **processes** the request, **performs** business logic, **and** interacts with the repository to save data to the database.
5. **Repository Operation:**
   - The repository **performs** the database operation (e.g., **saving** the new student record).
6. **HTTP Response:**
   - The backend **sends an** HTTP response back to the frontend with the result of the operation (e.g., the newly created student object).
7. **UI Update:**
   - The frontend **receives** the response **and** updates the UI to reflect the changes (e.g., displaying the new student in a list).

6a/11. Key Implementation Details

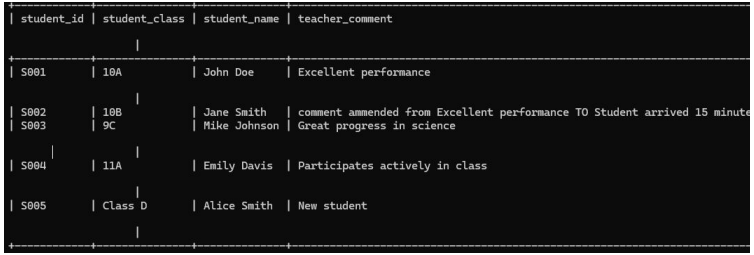
Setting up the **database**:

a. **Install MySQL:**
   - Download and install MySQL from the official website.
   - Start the MySQL server.

b. **Create a Database:**
   - Open MySQL Workbench **or** use the command line to create a new database.
   - Example command:
   - CREATE DATABASE: studentbehaviorreportdatabase;

Creating the Student class **and** related entities.

a. **Define** the Student **class** with appropriate fields **and** JPA annotations.



student_id	student_class	student_name	teacher_comment
S001	10A	John Doe	Excellent performance
S002	10B	Jane Smith	comment ammended from Excellent performance TO Student arrived 15 minute
S003	9C	Mike Johnson	Great progress in science
S004	11A	Emily Davis	Participates actively in class
S005	Class D	Alice Smith	New student

```java
// Add entity annotation to mark the Student class as a JPA
@Entity
public class Student {

    // Mark studentId as primary key
    @Id
    @Column(name = "student_id", nullable = false)
    private String studentId;

    @Column(name = "student_name", nullable = false)
    private String studentName;

    @Column(name = "student_class", nullable = true)
    private String studentClass;
```

```java
// Add constructors to initialize fields
public Student(String studentId, String s
    this.studentId = studentId;
    this.studentName = studentName;
    this.studentClass = studentClass;
    this.teacherComment = teacherComment;
}
```

6b/11. Key Implementation Details

Developing the API **endpoints** (GET, POST, PUT, DELETE).

a. **Create the StudentController Class:**
   - Define the REST API endpoints for CRUD operations.

```java
@RestController
@RequestMapping("/api/v1/students") // Base URL for all endpoints in this controller
public class StudentController {

    private final StudentRepository studentRepository;

    public StudentController(StudentRepository studentRepository) {
        this.studentRepository = studentRepository;
    }
```

- **`@RestController`:** Indicates that this class is a RESTful web service controller.
- **`@RequestMapping("/api/v1/students")`:** Sets the base URL for all endpoints in this controller to `/api/v1/students`.
- public class StudentController**:** Declares the StudentController class.
- private final StudentRepository studentRepository**:** Declares a private final field for the `StudentRepository` to interact with the database.
- public StudentController(StudentRepository studentRepository)**:** Constructor that initializes the studentRepository field through dependency injection.

```java
// GET a specific student's teacher comment by ID
@GetMapping("/{id}/getTeacherComment")
public ResponseEntity<String> getTeacherComment(@PathVariable String id) {
    Student student = studentRepository.findById(id)
            .orElseThrow(() -> new RuntimeException("Student not found"));
    return ResponseEntity.ok(student.getTeacherComment());
```

- **Annotation:** `@GetMapping("/{id}/getTeacherComment")` This annotation **defines** the endpoint **for** the GET request. It **specifies that this method will** handle HTTP GET requests sent to the URL pattern `/api/v1/students/{id}/getTeacherComment`.
- `{id}` is a path variable that will be replaced with the actual student ID when the request is made.

6c/11. key Implementation Details

- **Brief demo of project:**
  - creating, retrieving, updating, and deleting (POST, GET, UPDATE, DELETE) a student record
  - show the teacher's (and parent's) perspective using Postman.

Navigate to Project Directory:
cd "C:\Users\Clayton\CBFGitRepos\SpringInitilizrDependancies"

Build the Project:

mvn clean install

Run the Application:

mvn spring-boot:run

```
[INFO] ------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------
[INFO] Total time:  36.871 s
[INFO] Finished at: 2025-01-02T10:57:53Z
[INFO] ------------------------------------------
```

7/11. Testing

- Test cases implemented (CRUD operations)
- Tools used for testing (Postman, JUnit).

By **using** JUnit **and** Postman together, developers can ensure that **both** the individual components of their application **and** the overall API functionality **are** thoroughly tested **and** validated.

JUnit:

- **Purpose**: Automated unit **testing of individual** components **or** units of code in Java applications.
- **Use Case**: Writing **and** running test cases to **verify** the correctness of methods or classes.
- **Example**: Testing a method in the service layer to ensure it behaves as expected.

Postman:

- **Purpose**: API development and testing tool for sending HTTP requests and verifying API responses.
- **Use Case**: Testing RESTful API endpoints to ensure they return the expected responses.
- **Example**: Sending a POST request to add a new student and verifying the response.

8/11. Challenges and Tradeoffs

- Challenges faced during development and how I overcame them.
    - Example: Choosing pre-configured reports over customisable ones for simplicity.

9/11. Deliverables

- Database export, codebase, and the README file as deliverables.
- Everything is documented and accessible via the GitHub repository.

10/11. Conclusion

- The project's value:
  - "The Student Behavior Report API bridges the gap **between** teachers and parents, ensuring accountability **and** transparency **in** alternative provision settings."

- Next steps **or** potential improvements:
  1. extend the project by **adding** additional classes, **such as** a Teacher class, **to** enhance functionality **and** better represent the system's structure.
  2. integrating grading **or** attendance systems

11/11. Questions