Product Requirement Document (PRD) - Student API

1. Overview

The **Student API** provides CRUD (Create, Read, Update, Delete) operations for managing student records in a database. The API ensures **data validation** for fields like age, grade, and email while maintaining data integrity and consistency.

---

2. Objectives & Features

Core Features

- **Create Student**: Add a new student with name, age, grade, and email.
- **Retrieve Students**: Fetch all students or a specific student by ID.
- **Update Student**: Modify existing student details.
- **Delete Student**: Remove a student record.
- **Field Validations**: Prevent invalid data entry (negative age, incorrect grade, duplicate emails).
- **Profile URL Validation**: Ensure valid URL format for profile images.

---

3. Functional Requirements

ID	Requirement	Description
FR-01	Create Student	API should accept `name`, `age`, `grade`, and `email` to create a student.
FR-02	Fetch All Students	API should return a list of all students.
FR-03	Fetch Student by ID	API should return a specific student’s details based on the provided ID.
FR-04	Update Student	API should allow updating student details with PUT/PATCH requests.
FR-05	Delete Student	API should allow deletion of a student by ID.
FR-06	Validate Age	Age must be **between 0 and 120**.
FR-07	Validate Grade	Only **A, A+, B, B+, C, C+, D, D+** are allowed.
FR-08	Validate Email	Email must be **unique** and follow a **valid email format**.
FR-09	Validate Profile URL	URL must be in **valid format** (if provided).

---

4. Non-Functional Requirements

ID	Requirement	Description
NFR-01	Performance	API should return responses within **500ms**.
NFR-02	Scalability	API should support **thousands of concurrent users**.
NFR-03	Security	**Email & Profile URL** should be properly validated.
NFR-04	Error Handling	Proper **400/404/500** errors should be returned for bad requests.

---

5. API Endpoints & Methods

Endpoint	Method	Description
`/api/students/create/`	**POST**	Create a new student
`/api/students/`	**GET**	Fetch all students
`/api/students/{id}/`	**GET**	Fetch a student by ID
`/api/students/update/{id}/`	**PUT**	Update student details
`/api/students/patch/{id}/`	**PATCH**	Partially update a student
`/api/students/delete/{id}/`	**DELETE**	Delete a student

---

6. Error Handling

Error Code	Scenario	Example Response
`400 Bad Request`	Invalid input data	`{ "error": "Age cannot be negative." }`
`404 Not Found`	Student ID not found	`{ "error": "Student not found." }`
`409 Conflict`	Duplicate email	`{ "error": "Email already exists." }`
`500 Internal Server Error`	Unknown server issue	`{ "error": "Something went wrong." }`

High-Level Design (HLD) - Student API

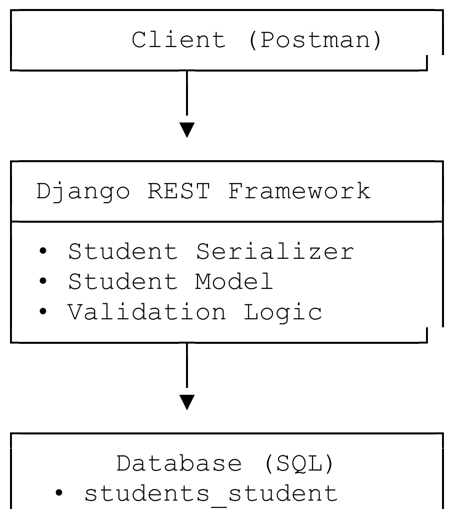
1.System Architecture

Tech Stack

- **Backend:** Python (Django, Django REST Framework)
- **Database:** SQLite / PostgreSQL
- **Validation:** Django Validators & Custom Functions
- **Testing:** Postman / Pytest / Unittest

---

2. Component Diagram



3. Data Model

Student Model

```
class Student(models.Model):
    name = models.CharField(max_length=255)
    age = models.IntegerField(
        validators=[
            MinValueValidator(0, message="Age cannot be negative."),
            MaxValueValidator(120, message="Age cannot be more than 120.")
        ]
    )
    grade = models.CharField(
        max_length=2,
        validators=[validate_grade]
    )
    email = models.EmailField(unique=True)
```

4. API Workflow

Student Creation Flow

1. **Client sends** `POST /api/students/create/`
2. **Backend validates input:**
   - Check **age range** (0-120)
   - Validate **grade values**
   - Ensure **unique email**
3. **If valid**, save to **database**
4. **Return success response** (`201 Created`)
5. **If invalid**, return `400 Bad Request`

5. Database Schema

Column Name	Type	Constraints
id	Integer	Auto-increment (Primary Key)
name	Varchar(255)	Not Null
age	Integer	Between 0 and 120
grade	Varchar(2)	Allowed: A, A+, B, B+, C, C+, D, D+
email	Varchar(255)	Unique

6. Deployment Considerations

□ Performance Optimization

- **Indexing** on `email` for fast lookup.
- **Validation at the database level** to prevent redundant checks.

□ Security

- **Input Sanitization** to prevent SQL injection.
- **Rate Limiting** to avoid API abuse.

□ Conclusion

This **PRD & HLD** provides a clear roadmap for the **Student API**, covering requirements, architecture, database design, API flow, and security considerations. The API ensures **data integrity**, **validation**, and **scalability** for future expansion.

User Story - Student API

□ Title: Manage Student Records Efficiently

□ Overview:

As an **Admin**, I want to **create, view, update, and delete student records** so that I can efficiently manage student data with proper validation and security.

□ User Stories

1. User Story: Create a Student

As an Admin,
I want to add a new student with a valid name, age, grade, and email,
So that the student can be registered in the system properly.

Acceptance Criteria:

- The API should **validate** all input fields.
- Age must be **between 0 and 120**.
- Grade should only be **A, A+, B, B+, C, C+, D, D+**.
- Email must be **unique** and properly formatted.
- If any validation fails, the API should return **400 Bad Request**.
- On successful creation, return **201 Created** with student details.

□ **Scenario:**
□ Given that I have valid student details, when I submit them, the student is created successfully.
□ If I provide an invalid grade (e.g., **"AB"**), the system should return **400 Bad Request**.

---

2. User Story: Fetch All Students

As an Admin,
I want to fetch a list of all students,
So that I can view their details.

□ **Acceptance Criteria:**

- If students exist, return a **200 OK** response with a list of students.
- If no students exist, return an **empty list `[]`**.
- Response time should be **under 500ms**.

□ **Scenario:**
□ If students exist, return a **list of student objects**.
□ If no students exist, return `[]`.

---

3. User Story: Fetch a Specific Student by ID

As an Admin,
I want to fetch a student by ID,
So that I can view their details.

□ **Acceptance Criteria:**

- If the student exists, return **200 OK** with student details.
- If the ID does not exist, return **404 Not Found**.

□ **Scenario:**
□ If I provide a valid student ID, I should get the student's details.
□ If I enter an invalid ID, I should receive **"Student not found" (404)**.

---

4. User Story: Update a Student

As an Admin,
I want to update a student's details,
So that I can correct or modify their records.

□ **Acceptance Criteria:**

- If the student exists, allow **full (PUT) or partial (PATCH) updates**.
- If an invalid age or grade is provided, return **400 Bad Request**.
- If the student does not exist, return **404 Not Found**.

□ **Scenario:**
□ If I provide valid update data, the student's details should be updated successfully.
□ If I provide an invalid grade, the API should return **400 Bad Request**.

---

5. User Story: Delete a Student

As an Admin,
I want to delete a student,
So that I can remove records that are no longer needed.

□ **Acceptance Criteria:**

- If the student exists, return **200 OK** with a success message.
- If the student does not exist, return **404 Not Found**.

□ **Scenario:**
□ If I delete a valid student, they should be removed successfully.
□ If I try to delete a non-existent student, the API should return **404 Not Found**.

---

6. User Story: Handle Edge Cases

As an API User,
I want the system to properly handle edge cases,
So that invalid data does not corrupt the database.

□ **Acceptance Criteria:**

- If age is **-9**, return **400 Bad Request** with `"Age cannot be negative"` error.
- If age is **200**, return **400 Bad Request** with `"Age cannot be more than 120"` error.
- If grade is **"AB"**, return **400 Bad Request** with `"Invalid grade"` error.
- If email is already in use, return **409 Conflict**.
- If profile URL is invalid, return **400 Bad Request**.

□ **Scenario:**
□ Given that I enter valid data, the system should accept it.
□ If I enter invalid data (negative age, incorrect grade, duplicate email), the system should reject it.

□ Security & Validations

Validation	Error Message

Validation	Error Message
Negative Age (-9)	`"Age cannot be negative."`
Too High Age (200)	`"Age cannot be more than 120."`
Invalid Grade (AB)	`"Invalid grade. Allowed values: A, A+, B, B+, C, C+, D, D+"`
Duplicate Email	`"Email already exists."`
Invalid Profile URL	`"Invalid URL format."`

Summary

This **Student API User Story** ensures proper **data management, validation, and error handling**. It guarantees that **admins can efficiently create, update, fetch, and delete students** while preventing invalid inputs.