**Software Requirements Specification (SRS)**

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to define the technical and functional requirements of a Django-based project comprising three key applications: **Course**, **Student**, and **Payment**. Each app operates with its own database and communicates using Django REST Framework (DRF). The system employs Stripe for secure payment handling, making it efficient for educational institutions.

**1.2 Scope**

The system serves as an online portal for course management, student registration, and payment processing. It ensures modularity by separating concerns into three distinct applications, ensuring maintainability, scalability, and robustness.

**2. Overall Description**

**2.1 System Perspective**

The system is modular:

1. **Course App** handles the creation and management of courses.
2. **Student App** oversees student information and enrollment.
3. **Payment App** integrates with Stripe for payment transactions.

All applications share data via a centralized API router.

**2.2 Product Features**

* **Course Management**: Add, view, update, and delete courses.
* **Student Management**: Register students, manage profiles, and enroll students in courses.
* **Payment Handling**: Process payments securely and notify users of payment status.

**3. Functional Requirements**

**3.1 Course App**

* Models:
  + **Course**: Fields for title, description, instructor, schedule, and price.
* Features:
  + CRUD operations on courses.
  + API Endpoints:
    - POST /api/courses/ - Add a new course.
    - GET /api/courses/ - Retrieve all courses.
    - GET /api/courses/{id}/ - Retrieve course by ID.
    - PUT /api/courses/{id}/ - Update a course.
    - DELETE /api/courses/{id}/ - Delete a course.

**3.2 Student App**

* Models:
  + **Student**: Fields for name, email, phone, and enrolled courses.
* Features:
  + Register new students.
  + Allow course enrollment.
  + API Endpoints:
    - POST /api/students/ - Register a new student.
    - GET /api/students/{id}/ - Retrieve student details.
    - POST /api/students/enroll/ - Enroll in a course.

**3.3 Payment App**

* Models:
  + **Payment**: Fields for student ID, course ID, amount, status, and payment date.
* Features:
  + Integrate Stripe for payments.
  + Track payment status.
  + API Endpoints:
    - POST /api/payments/ - Initiate payment.
    - GET /api/payments/{id}/ - Retrieve payment details.
    - Webhook: Listen to Stripe events for payment success/failure.

**4. Non-Functional Requirements**

**4.1 Performance**

* **APIs** should handle up to 500 requests/second.

**4.2 Security**

* Use **Stripe's official API** for payment transactions.
* Store sensitive keys in .env files and load them with django-environ.

**4.3 Scalability**

* Use **Django's database routing** to support seamless communication between multiple databases.

**4.4 Usability**

* Provide a user-friendly interface for admin tasks using Django Admin.
* Ensure APIs are well-documented with Swagger or DRF-YASG.

**5. System Models**

**5.1 Database Schema**

* **Course Database**:
  + Table: Course  
    Fields: id, title, description, price, schedule.
* **Student Database**:
  + Table: Student  
    Fields: id, name, email, phone, enrolled\_courses.
* **Payment Database**:
  + Table: Payment  
    Fields: id, student\_id, course\_id, amount, status, payment\_date.

**5.2 API Communication**

* Use a **central API router**:
  + /api/courses/
  + /api/students/
  + /api/payments/

**5.3 Stripe Integration**

* Stripe Python SDK is used for:
  + **Creating payments**: stripe.PaymentIntent.create()
  + **Webhook**: Listen for payment\_intent.succeeded events.
  + Save transaction data in the Payment model.

**6. Implementation Plan**

**6.1 Project Structure**

mysite/

├── course/

├── student/

├── payment/

├── api/

├── templates/

├── static/

├── settings.py

├── urls.py

└── wsgi.py

**6.2 Third-Party Libraries**

* **Stripe**: pip install stripe
* **DRF**: pip install djangorestframework
* **Environment Variables**: pip install django-environ

**6.3 API Documentation**

* Use Swagger for API documentation:
* pip install drf-yasg

**7. Deployment Plan**

**7.1 Development**

* Use SQLite for local databases.
* Test APIs with Postman.

**7.2 Production**

* Migrate databases to MySQL or PostgreSQL.
* Deploy using Docker or platforms like Heroku or AWS.