**Product Requirements Document (PRD) - Global Curriculum Engine**

**Version:** 4.0 (Detailed for Development by an AI Agent) **Last Updated:** August 11, 2025 **Product Owner:** Mamdouh Ragab

**1. Guiding Strategic Principle and Overview**

**1.1. Global Vision:** To build a global educational ecosystem capable of ingesting any curriculum from any country and transforming it into a hyper-personalized learning experience for each individual student. We are not just building an app; we are establishing the **technological asset** that will drive the future of personalized education.

**1.2. Guiding Strategic Principle:** Build a **"Curriculum-Agnostic Engine"**. Every functional requirement serves the ultimate goal of building a flexible and scalable system where the "mind" of the system (the curriculum) is decoupled from its "body" (the application). This ensures horizontal scaling (new countries) and vertical scaling (new grade levels) with maximum speed and efficiency.

**1.3. Proposed Solution:** "Smart School" is a mobile-first application that offers a dynamic, AI-powered curriculum engine. Instead of serving static content, the application **generates multimedia lessons, explanations, and exercises in real-time on demand**, covering the entire curriculum.

**1.4. Core Philosophy (Hyper-Personalization and Adaptation):** The platform is an **Adaptation Engine**. Every decision the system makes is aimed at answering one question: "What is the best next step for this specific student to achieve maximum understanding?"

**Guiding Strategic Principle (with Technical Details)**

* **1.1 Global Vision:**
  + **Architecture:** Microservices (Kubernetes) + GraphDB (Neo4j)
  + **Data Flow:** JSON Curriculum → PostgreSQL (Structure) → Neo4j (Concept Relationships)
* **1.2 Technical Principle:**
  + **"Decoupling Curriculum from Application" via:**
    - **API:** Curriculum Ingestion API (REST/JSON)
    - **Storage Standard:** Curriculum Schema v1.2 (See Appendix A)

**2. Product Goals and Success Metrics**

**2.1. Business Goals:**

* **(BG-01):** Prove the technical feasibility of the "Dynamic Curriculum Engine" model.
* **(BG-02):** Achieve market validation by securing 200 prepaid subscriptions within 120 days.
* **(BG-03):** Build the foundation for a competitive data advantage by collecting initial interaction data.

**2.2. User Goals:**

* **(UG-01):** Enable students to study any lesson, anytime, anywhere.
* **(UG-02):** Provide students with clear and personalized explanations for difficult concepts.
* **(UG-03):** Help students solve their homework instantly and effectively.
* **(UG-04 - The Core Goal):** Make the student feel that the platform truly "understands" them.

**2.3. Key Success Metrics:**

* **(SM-01):** Reach 200 prepaid subscriptions.
* **(SM-02):** Active users spend an average of 60 minutes per day on the app.
* **(SM-03):** Achieve an average rating of > 4.0/5.0 on generated lessons.
* **(SM-04):** Achieve a lesson generation time of < 15 seconds.

**3. Target Audience**

**3.1. Primary Persona:** "Karim," a first-year secondary school student, striving for excellence and a deep understanding of scientific subjects, who is bored with traditional teaching methods.

**3.2. Secondary Persona:** "Nour's" parents, looking for an effective, reliable, and affordable alternative to private tutoring.

**4. Detailed Functional Requirements**

**Epic 1: Curriculum Ingestion & Student Profiling**

**FR-01: Curriculum Map Ingestion**

* **User Story:** As a system administrator, I want to upload a JSON file representing the complete structure of the curriculum, so that the system can understand and display all available subjects, units, and lessons.
* **Technical Details:** The system must accept a curriculum.json file with the following hierarchical structure (country -> grade -> subject -> units -> chapters -> lessons):
* {
* "country": "Egypt",
* "grade": "Secondary One",
* "subjects": [
* {
* "subjectId": "PHY-01",
* "name": "Physics",
* "units": [
* {
* "unitId": "PHY-01-U1",
* "name": "Physical Quantities and Units of Measurement",
* "chapters": [
* {
* "chapterId": "PHY-01-U1-C1",
* "name": "Chapter One: Introduction to Measurement",
* "lessons": [
* {
* "lessonId": "PHY-01-U1-C1-L1",
* "name": "Physical Measurement",
* "objectives": ["Understand the importance of measurement", "Distinguish between fundamental and derived quantities"],
* "dependencies": [],
* "keywords": ["Measurement", "International System", "Fundamental Quantities"]
* }
* ]
* }
* ]
* }
* ]
* }
* ]
* }
* **Acceptance Criteria:**
  + The system can parse the JSON file without errors.
  + Subjects, units, and lessons are displayed correctly in the user interface.

**FR-02 & FR-03: Create and Detail the "Student Skill Profile"**

* **User Story:** As a student, upon registration, I want the system to create a personal profile that tracks my progress and preferences, so it can provide me with a personalized learning experience.
* **Technical Details:** When a new user registers, a document is created in the database (Firestore) with the following structure:
* {
* "userId": "UNIQUE\_USER\_ID",
* "skillProfile": {
* "PHY-01-U1-L1": {
* "masteryScore": 0.0, // (0.0 to 1.0)
* "confidence": "low", // (low, medium, high)
* "lastAttempt": "timestamp",
* "interactionHistory": []
* }
* },
* "learningPreferences": {
* "style": "simplified", // (academic, simplified, humorous)
* "tutorPersona": {
* "name": "Professor Khalid",
* "gender": "male"
* }
* }
* }
* **Acceptance Criteria:**
  + The profile is created successfully for every new user.
  + masteryScore and learningPreferences values can be updated via the user interface.

**FR-04 & FR-05: Customize Learning Preferences**

* **User Story:** As a student, I want to choose the explanation style and tutor persona that suits me, so I can feel more comfortable while learning.
* **Technical Details:** The settings screen must provide clear options to update the learningPreferences fields in the student's profile.
* **Acceptance Criteria:** Changes are saved immediately and reflected in subsequently generated lessons.

**FR-06 & FR-07: Browse Curriculum and Diagnostic Test**

* **User Story:** As a student, I want to browse the curriculum easily, and when entering a new unit, I want to take a short test to determine my current level.
* **Technical Details:**
  + The UI displays the curriculum based on curriculum.json.
  + When a unit is clicked for the first time, the "Assessment Engine" is called to generate 3-5 questions covering the unit's objectives.
  + The skillProfile is updated based on the test results.
* **Acceptance Criteria:**
  + Browsing is smooth and logical.
  + The test is presented and the skill profile is updated successfully.

**Functional Requirements (with Implementation Specifications)**

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Implementation Specifications** | **Data Model** |
| **FR-01** | - Endpoint: POST /api/curriculums<br>- Validation: JSON Schema v7<br>- Storage: curriculums table (PostgreSQL) | json { "id": "uuid", "country": "EG", "subject": "Physics", "units": [...] } |
| **FR-02** | - Creation Trigger: On POST /api/register<br>- Structure: JSONB (PostgreSQL) with skill\_profile field | sql CREATE TABLE student\_profiles ( skill\_profile JSONB, preferences JSONB ) |

**Epic 2: Adaptive Dynamic Curriculum Engine**

**FR-08 & FR-09: Personalized Lesson Generation**

* **User Story:** As a student, when I request a lesson, I want it to be explained in a way that suits my level and preferred style.
* **Technical Details:** When a lesson (lessonId) is requested, the system assembles a detailed prompt and sends it to the Large Language Model (LLM).
  + **Prompt Structure:**
    - **Role:** You are an expert physics teacher named [tutorPersona.name].
    - **Persona:** Explain in a [learningPreferences.style] style.
    - **Context:** Student [userId] has a mastery level of [skillProfile.masteryScore] in this lesson.
    - **Task:** Explain the lesson "[lesson.name]" which aims to achieve: [lesson.objectives]. Focus on concepts that students typically find difficult.
    - **Output Format:** Provide the explanation in Markdown format.
* **Acceptance Criteria:**
  + The lesson text is generated in less than 10 seconds.
  + The generated content clearly reflects the style and persona of the chosen tutor.

**FR-10: Multimedia Synthesis Engine**

* **User Story:** As a student, I prefer watching a short video instead of reading a long text, and I want the tutor's voice to match the persona I chose.
* **Technical Details:**
  + **Step 1 (Text-to-Speech):** The text generated from (FR-09) is sent to a TTS service. The voice is selected based on tutorPersona.gender.
  + **Step 2 (Video Synthesis):** A simple video is created that displays the text synchronously with the audio, highlighting the lesson.keywords.
* **Acceptance Criteria:**
  + Video and audio are generated in less than 5 seconds (after text generation).
  + Audio quality is clear and video synchronization is accurate.

**Epic 2: Dynamic Curriculum Engine (with Implementation Specifications)**

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Prompt Engineering** | **Performance Parameters** |
| **FR-09** | Dynamic Template: "Explain {concept} to {student\_name}. Focus on {weakness}. Use {style} style. Speak like {persona}." | - Response Time: <5s<br>- Cost/Request: <$0.002 |
| **FR-10** | Service Integration:<br>1. Text → Amazon Polly (Audio)<br>2. Audio + Avatar → DeepMotion (Video) | - Video Quality: 720p<br>- File Size: <3MB/minute |

**FR-11 & FR-12: Interactive Questions and Skill Updates**

* **User Story:** After each lesson, I want to solve some questions to ensure I understood, and I want the system to remember my performance for the future.
* **Technical Details:**
  + The LLM generates three questions (easy, medium, hard) based on the lesson objectives.
  + After each answer, the updateSkillProfile(userId, lessonId, isCorrect) function is called.
  + The function modifies masteryScore and confidence based on a simple algorithm (e.g., +0.1 for a correct answer, -0.05 for incorrect).
* **Acceptance Criteria:**
  + The skillProfile in the database is updated instantly (< 1 second).

**Epic 3: Adaptive Smart Assistant**

**FR-13 & FR-14: Solution Mode**

* **User Story:** As a student, when I face a difficult homework question, I want to take a picture of it and get a detailed solution in the style of my smart tutor.
* **Technical Details:**
  + The app uses the device's camera to capture an image.
  + The image is sent to a Multimodal LLM with a prompt containing the learningPreferences.
  + The solution is displayed in the same interface as the lessons.
* **Acceptance Criteria:**
  + Text recognition accuracy in the image is > 90%.
  + The provided solution is accurate and follows the specified style.

**FR-15 & FR-16: Tutor Mode**

* **User Story:** While reading an explanation, I want to click on any term I don't understand to get an instant, simplified explanation.
* **Technical Details:**
  + The generated text is parsed, and lesson.keywords are identified as clickable elements.
  + On click, a lightweight language model (see FR-18) is called directly on the device (Edge) to provide a quick definition.
* **Acceptance Criteria:**
  + The pop-up window with the explanation appears in less than 0.5 seconds.

**Epic 4: Smart Cost & Performance Optimization**

**FR-17: Caching & Proactive Generation System**

* **Technical Details:**
  + **Caching:** When a lesson is generated, the result (text and video) is stored in a cache (e.g., Redis) with a key based on lessonId and learningPreferences.style.
  + **Proactive Generation:** A nightly Cron Job runs, analyzing the most requested lessons and pre-generating them for common styles.
* **Acceptance Criteria:**
  + Lesson load time from cache is < 2 seconds.
  + Pre-generation is not applied to lessons requiring recent student data.

**FR-18: Edge Computing for Static Tasks**

* **Technical Details:**
  + A small language model (e.g., a distilled version of a larger model, <3MB) is embedded within the mobile application.
  + This model is used exclusively for the "explain terms" task (FR-16) to ensure an instant, offline response.
* **Acceptance Criteria:**
  + Model size does not exceed 3MB.
  + On-device processing time is < 0.5 seconds.

**Smart Caching System (Technical Specs):**

def get\_lesson(student\_id, concept\_id):

# Generate a cache key based on student group and concept hash

cache\_key = f"{student\_group}:{concept\_hash}"

# If the key exists in cache and the student's profile hasn't been updated, return the cached version (FR-17)

if cache.exists(cache\_key) and not profile\_updated(student\_id):

return cache.get(cache\_key)

# Dynamic Generation (FR-08)

prompt = build\_prompt(student\_profile, concept)

content = generate\_content(prompt)

# Store in cache with video compression

compressed = compress\_video(content, quality=student\_connection\_quality) # NFR-06

cache.set(cache\_key, compressed, ttl=48h)

**FR-19: Dynamic Model Routing**

* **Technical Details:**
  + A gateway is created for requests to the language models.
  + **Logic:**
    - If request is "explain term" -> route to local (Edge) model.
    - If request is "generate questions" or "solve simple homework" -> route to a fast, cheap model (e.g., Gemini Flash).
    - If request is "generate full lesson" or "solve complex homework" -> route to the main model (e.g., Claude 3.5 Sonnet).
* **Acceptance Criteria:**
  + 100% of requests are routed correctly based on the task type.

**FR-20: Smart Quota System**

* **Technical Details:**
  + "Resource units" are assigned to each task (e.g., generate lesson = 10 units, solve homework = 5 units).
  + Each user gets 1000 free units per month.
  + When the balance runs out, the user can upgrade or continue using low-cost tasks (like explaining terms).
* **Acceptance Criteria:**
  + Units are accurately deducted from the user's balance.
  + A clear message is displayed when attempting a task with insufficient units.

**5. Non-Functional Requirements (NFRs)**

|  |  |  |
| --- | --- | --- |
| **ID** | **Description** | **Acceptance Criteria** |
| **NFR-01** | **Performance** | - End-to-End new lesson generation time: < 15 seconds.<br>- Cached lesson load time: < 2 seconds. |
| **NFR-02** | **Usability** | Simple and intuitive interface that requires no prior training. |
| **NFR-03** | **Reliability** | Achieve 99.5% Uptime. |
| **NFR-04** | **Security** | Encrypt all user data and passwords (at-rest and in-transit). |
| **NFR-05** | **Scalability** | Architecture capable of handling 10,000 concurrent users. |
| **NFR-06** | **Edge Computing Efficiency** | - Battery consumption: < 1% per executed task.<br>- Processing time: < 0.5 seconds. |
| **NFR-07** | **Local Data Security** | Encrypt content stored on the user's device using AES-256 standard. |
| **NFR-08** | **Proactive Generation Efficiency** | Used storage space must not exceed 10% of total storage capacity. |
| **NFR-09** | **Model Quality Assurance** | - Local model accuracy must exceed 95% for its tasks.<br>- An automatic failover mechanism to the main model if the local model fails. |

**6. Smart Developer's Guide (Technical Appendices)**

**Appendix A: Local Model Learning Curve (FR-18)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Training Accuracy** | **Model Size** | **Permissible Error Rate** |
| Explain Terms | 97.2% | 2.8MB | <4% |
| Simple Questions | 92.1% | 4.3MB | <8% |

**Automated Development Guidelines**

1. **Development Methodology:**
   * Model: Test-Driven Development (TDD)
   * Continuous Integration: GitHub Actions (with SonarCloud)
2. **Code Templates:**
   * Lesson Generation Service: /templates/lesson\_service.py
   * Camera Interface: /templates/camera\_widget.dart
3. **Calibration Tests:**
   * Performance Test: load\_test.py -users 1000
   * Accuracy Test: accuracy\_test.py -model edge\_v2

**7. External Integrations Map**

|  |  |  |
| --- | --- | --- |
| **Service** | **Documentation** | **Integration Point** |
| Claude API | Anthropic Docs | POST /v1/messages |
| DeepMotion | API Reference | POST /render-video |
| Redis Cloud | Redis Docs | CacheService.connect() |

**Instant Execution Version for the Smart Developer:**

# Quick Start Steps

git clone https://github.com/AiSchool/blueprint.git

cd blueprint

python setup.py --env=prod --ai\_key=$ANTHROPIC\_KEY

flutter build apk --release --target=lib/main\_prod.dart

* Social features (public profiles, forums).
* Advanced curricula (e.g., IGCSE, SAT).
* Parent reports and dashboard.
* Web application.

**8. Integrated System Flow**

graph TD

subgraph "On User's Device (Edge)"

A[Request Term Explanation (FR-16)] --> B{Can be processed locally?};

B -->|Yes| C[Edge Model: Instant processing < 0.5s];

end

subgraph "On Cloud Server (Backend)"

B -->|No| D[API Gateway: Model Router];

D -->|Task: Simple| E[Fast/Cheap LLM (e.g., Gemini Flash)];

D -->|Task: Complex| F[Advanced LLM (e.g., Claude 3.5 Sonnet)];

G[Request Full Lesson] --> H{Is it in cache? (FR-17)};

H -->|Yes| I[Instant display from Cache < 2s];

H -->|No| J[Lesson Generation Engine];

J --> D;

K[Solve Homework (Image)] --> D;

L[Update Skill Profile] --> M[Firestore: Student Skill Profile];

end

subgraph "Nightly Operations (Cron Job)"

N(Analyze User Behavior) --> O(Proactive Generation of Popular Lessons);

O --> I;

end

C --> L;

E --> L;

F --> L;

I --> L;