Serverless Online Exam Management System (SOEMS)

Table of Contents

Comprehensive Development Roadmap for Google Antigravity IDE

Project: SOEMS - MERN Stack + Python Backend
Prepared By: Senior Full-Stack Engineer
Date: November 2025
Target Environment: Google Antigravity IDE / Cloud-Native Serverless Deployment

EXECUTIVE OVERVIEW

This roadmap translates the complete SRS into actionable development tasks organized by **module, sprint, and epic**, with clear dependencies, estimated effort, and AI-IDE integration points. The roadmap is structured for rapid prototyping and iterative development using Google's Antigravity IDE for visual programming and automated code generation.

PHASE 1: FOUNDATION & INFRASTRUCTURE (Weeks 1-2)

Epic 1.1: Project Initialization & Environment Setup

Task 1.1.1: Google Antigravity IDE Project Creation

- **Objective:** Initialize project structure in Antigravity IDE
- **Details:**
  - Create new MERN Stack project template
  - Set up project metadata: name, version, team members
  - Configure project-level environment variables (API keys, database credentials)
  - Enable version control integration (GitHub)
  - Set up cloud deployment targets (AWS Lambda, GCP Cloud Functions)
- **Deliverables:** Project scaffold with `.gitignore`, `package.json`, `pyproject.toml`
- **Effort:** 4 hours
- **Dependencies:** None
- **IDE Tools:** Project Wizard, Environment Manager

Task 1.1.2: Frontend Architecture Setup (React + TypeScript)

- **Objective:** Configure React frontend with TypeScript, routing, state management
- **Details:**
  - Create React app with `create-react-app` or Vite
  - Set up TypeScript strict mode
  - Configure React Router for multi-page navigation (Login, Dashboard, Exam, Results)
  - Install and configure Redux or Zustand for state management
  - Set up axios/fetch interceptors for API calls with error handling
  - Configure ESLint, Prettier for code quality
- **Deliverables:** React project structure with basic routing and state setup
- **Effort:** 6 hours
- **Dependencies:** Task 1.1.1
- **File Structure:**

```
frontend/
├── src/
│   ├── components/
│   │   ├── Auth/
│   │   ├── Dashboard/
│   │   ├── Exam/
│   │   └── Common/
│   ├── pages/
│    ├── hooks/
│    ├── context/ or store/
│    ├── services/ (API calls)
│    ├── styles/
│    └── utils/
```

Task 1.1.3: Backend Architecture Setup (Python FastAPI)

- **Objective:** Initialize Python FastAPI microservices framework
- **Details:**
  - Create Python project with `fastapi` scaffold
  - Set up virtual environment and `requirements.txt`
  - Configure FastAPI app factory pattern for modularity
  - Set up CORS, middleware for security
  - Configure logging and error handling globally
  - Set up database ORM (SQLAlchemy) with connection pooling
  - Create initial models: User, Exam, Question, Submission, Result
- **Deliverables:** FastAPI app skeleton with database connection
- **Effort:** 6 hours
- **Dependencies:** Task 1.1.1
- **File Structure:**

```
backend/
├── app/
│    ├── __init__.py
│    ├── main.py
│    ├── config.py
│    ├── database.py
│    ├── schemas/
│    ├── models/
│    ├── api/
│    │    ├── v1/
│    │    │    ├── auth/
│    │    │    ├── exams/
│    │    │    ├── questions/
│    │    │    ├── submissions/
│    │    │    └── analytics/
│    │    └── dependencies.py
│    └── services/
├── tests/
└── requirements.txt
```

Task 1.1.4: Database Schema & Migration Setup

- **Objective:** Design relational database schema and set up migrations
- **Details:**
  - Create SQLAlchemy models for all entities: User, Teacher, Student, Exam, Question, Submission, Answer, Result, ProctorEvent, AuditLog, Notification
  - Define relationships and foreign keys
  - Set up Alembic for database migrations
  - Create seed scripts for test data
  - Configure indexes for high-query tables (answers, submissions)
- **Deliverables:** Database schema migration files, seed scripts
- **Effort:** 4 hours
- **Dependencies:** Task 1.1.3
- **Key Tables:**
  - `users`: id, email, password_hash, role, created_at
  - `exams`: id, title, duration, scheduled_start, created_by, status
  - `questions`: id, exam_id, type, content, correct_answer, difficulty
  - `submissions`: id, exam_id, student_id, start_time, end_time, status
  - `answers`: id, submission_id, question_id, response, score
  - `audit_logs`: id, user_id, action, timestamp, ip_address

Task 1.1.5: Authentication & Authorization Framework

- **Objective:** Implement JWT-based auth with role-based access control
- **Details:**
  - Create JWT token generation/validation service (PyJWT)
  - Implement password hashing (bcrypt)
  - Set up role-based access control (RBAC) with decorators
  - Create authentication middleware for protected routes
  - Implement refresh token mechanism
  - Set up multi-factor authentication (MFA) infrastructure (optional)
- **Deliverables:** Auth service module, JWT utilities, RBAC decorators
- **Effort:** 5 hours
- **Dependencies:** Task 1.1.3, Task 1.1.4
- **Key Functions:**
  - `generate_jwt_token(user_id, role, exp_delta)`
  - `verify_jwt_token(token)`

    - `require_role(*allowed_roles)` decorator
    - `get_current_user()` dependency

Task 1.1.6: Environment Configuration & Secrets Management

- **Objective:** Set up configuration management for development, staging, production
- **Details:**
    - Create `.env` template file with required variables
    - Configure Python dotenv for local development
    - Set up AWS Secrets Manager or GCP Secret Manager integration for production
    - Define API base URLs, database credentials, JWT secret, notification service keys
    - Create configuration validation on startup
- **Deliverables:** Config module, `.env.example`, secrets setup documentation
- **Effort:** 2 hours
- **Dependencies:** Task 1.1.1, Task 1.1.3
- **Example Config:**

```
class Config:
    DATABASE_URL: str
    JWT_SECRET: str
    JWT_ALGORITHM: str = "HS256"
    TWILIO_API_KEY: str
    SENDGRID_API_KEY: str
    AWS_REGION: str
```

PHASE 2: CORE AUTHENTICATION & USER MANAGEMENT (Weeks 2-3)

Epic 2.1: User Authentication & Registration

Task 2.1.1: User Registration Endpoint

- **Objective:** Implement /register API for student/teacher signup
- **Details:**
    - Create POST `/api/v1/auth/register` endpoint
    - Accept: email, password, name, role (student/teacher)
    - Validate email uniqueness, password strength
    - Hash password with bcrypt
    - Save user to database
    - Send verification email (optional)
    - Return JWT token on success

- **Deliverables:** FastAPI route, request schema, response schema
- **Effort:** 3 hours
- **Dependencies:** Task 1.1.5, Task 1.1.4
- **Request Schema:**

```
class RegisterRequest(BaseModel):
    email: str
    password: str
    name: str
    role: Literal["student", "teacher"]
```

Task 2.1.2: User Login Endpoint

- **Objective:** Implement /login API with JWT token issuance
- **Details:**
    - Create POST `/api/v1/auth/login` endpoint
    - Accept: email, password
    - Verify credentials (password comparison)
    - Generate JWT access token + refresh token
    - Set refresh token as httpOnly cookie
    - Return access token and user profile
- **Deliverables:** Login endpoint, token service
- **Effort:** 3 hours
- **Dependencies:** Task 2.1.1

Task 2.1.3: Token Refresh & Logout

- **Objective:** Implement token refresh and logout flows
- **Details:**
    - Create POST `/api/v1/auth/refresh` endpoint to issue new access token
    - Create POST `/api/v1/auth/logout` endpoint to invalidate tokens
    - Maintain token blacklist in Redis for revocation
    - Test token expiry scenarios
- **Deliverables:** Refresh/logout endpoints, token blacklist service
- **Effort:** 2 hours
- **Dependencies:** Task 2.1.2

Task 2.1.4: Frontend Login & Registration UI

- **Objective:** Build React components for authentication
- **Details:**
  - Create LoginPage component (email, password fields, submit, forgot password link)
  - Create RegisterPage component (email, name, password, role selector)
  - Implement form validation
  - Handle API calls using axios service
  - Store JWT token in localStorage/sessionStorage
  - Redirect on success to dashboard
  - Display error messages
- **Deliverables:** React components, form handling, auth service
- **Effort:** 4 hours
- **Dependencies:** Task 1.1.2, Task 2.1.1, Task 2.1.2
- **Components:**
  - `LoginPage.tsx`
  - `RegisterPage.tsx`
  - `useAuth.ts` (custom hook)

Task 2.1.5: Protected Routes & Navigation Guard

- **Objective:** Implement client-side route protection
- **Details:**
  - Create ProtectedRoute component that checks JWT token
  - Redirect unauthenticated users to login
  - Create PrivateLayout for authenticated pages
  - Implement role-based route access (e.g., /admin only for admins)
  - Handle token expiry and refresh flow
- **Deliverables:** ProtectedRoute component, route configuration
- **Effort:** 3 hours
- **Dependencies:** Task 2.1.4

Task 2.1.6: Backend User Profile & Management Endpoints

- **Objective:** Create user CRUD endpoints
- **Details:**
  - GET `/api/v1/users/me` - fetch current user profile
  - PUT `/api/v1/users/me` - update user profile (name, email)

  - GET `/api/v1/users/{user_id}` - admin endpoint to fetch any user
  - DELETE `/api/v1/users/{user_id}` - admin endpoint to delete user
  - All endpoints require authentication; admin endpoints require admin role
- **Deliverables:** User endpoints, schemas
- **Effort:** 3 hours
- **Dependencies:** Task 1.1.5

Epic 2.2: Role-Based Dashboard Structure

Task 2.2.1: Student Dashboard Layout

- **Objective:** Build student home dashboard
- **Details:**
  - Display list of upcoming exams assigned to student
  - Show exam details: title, date, time, duration, status (not started, in progress, completed)
  - Add quick action buttons (Start Exam, View Results)
  - Show recent notifications/reminders
  - Add links to profile, settings
- **Deliverables:** StudentDashboard component with mock data
- **Effort:** 4 hours
- **Dependencies:** Task 1.1.2

Task 2.2.2: Teacher Dashboard Layout

- **Objective:** Build teacher home dashboard
- **Details:**
  - Display list of exams created by teacher
  - Show stats: total exams, questions, submissions, average score
  - Add buttons to create new exam, manage questions, view analytics
  - Show recent submissions and grading queue
  - Display notifications (new submissions, system alerts)
- **Deliverables:** TeacherDashboard component
- **Effort:** 4 hours
- **Dependencies:** Task 1.1.2

Task 2.2.3: Admin Dashboard Layout

- **Objective:** Build admin control panel
- **Details:**
  - Display system health metrics (uptime, load, active users)
  - Show user management section (add/remove users, role assignment)
  - Display audit logs viewer with search/filter
  - Show system configuration options
  - Include alerts and notifications section
- **Deliverables:** AdminDashboard component with sections
- **Effort:** 4 hours
- **Dependencies:** Task 1.1.2

PHASE 3: QUESTION BANK & EXAM MANAGEMENT (Weeks 3-4)

Epic 3.1: Question Management Module

Task 3.1.1: Question Model & Database Schema

- **Objective:** Define question data structure and database table
- **Details:**
  - Create Question model with fields: id, text, type (MCQ, descriptive, coding), options, correct_answer, difficulty, tags, created_by, created_at, updated_at
  - Create QuestionBank model: id, name, subject, description, created_by
  - Define relationship: QuestionBank has many Questions
  - Add indexes on (type, difficulty, created_by) for query optimization
- **Deliverables:** SQLAlchemy models, migration file
- **Effort:** 2 hours
- **Dependencies:** Task 1.1.4

Task 3.1.2: Question CRUD API Endpoints

- **Objective:** Create REST endpoints for question management
- **Details:**
  - POST `/api/v1/questions` - create new question (teacher only)
  - GET `/api/v1/questions/{question_id}` - fetch single question
  - PUT `/api/v1/questions/{question_id}` - update question (owner only)
  - DELETE `/api/v1/questions/{question_id}` - delete question (owner/admin)

    - GET `/api/v1/questions?bank_id=X&amp;type=Y&amp;difficulty=Z` - list questions with filters
- **Deliverables:** Question endpoints, schemas
- **Effort:** 4 hours
- **Dependencies:** Task 3.1.1, Task 1.1.5

Task 3.1.3: Question Import/Export (CSV)

- **Objective:** Enable bulk question upload/download
- **Details:**
    - Create POST `/api/v1/questions/import` endpoint that accepts CSV file
    - Parse CSV: text, type, options, answer, difficulty
    - Validate and batch insert into database
    - Return success/error summary
    - Create GET `/api/v1/questions/export` endpoint to export as CSV
- **Deliverables:** Import/export service, CSV parser
- **Effort:** 4 hours
- **Dependencies:** Task 3.1.2
- **CSV Format:**

```
text,type,options,correct_answer,difficulty
"What is 2+2?",MCQ,"A:3|B:4|C:5",B,easy
```

Task 3.1.4: Question Bank Management UI

- **Objective:** Build React interface for managing questions
- **Details:**
    - Create QuestionBankListPage showing all question banks (teacher only)
    - Create CreateQuestionBankModal to create new bank
    - Create QuestionListPage showing questions in a bank
    - Create QuestionFormModal for adding/editing questions (with rich text editor)
    - Create ImportQuestionModal for CSV upload
    - Implement search, filter by type/difficulty
- **Deliverables:** React components for question management
- **Effort:** 6 hours
- **Dependencies:** Task 1.1.2, Task 3.1.2

Task 3.1.5: Rich Content Support (Text, Images, Code)

- **Objective:** Enable multimedia content in questions
- **Details:**
  - Add media_url field to Question model
  - Create endpoint to upload images/PDFs to S3
  - Create endpoint to render LaTeX/code snippets
  - Update QuestionFormModal to include image uploader
  - Display media in question preview
- **Deliverables:** Media upload service, S3 integration, media display components
- **Effort:** 4 hours
- **Dependencies:** Task 3.1.4

Epic 3.2: Exam Creation & Scheduling

Task 3.2.1: Exam Model & Database Schema

- **Objective:** Define exam data structure
- **Details:**
  - Create Exam model: id, title, description, duration (minutes), scheduled_start, scheduled_end, created_by, status (draft, published, live, completed), instructions
  - Create ExamQuestion junction table: exam_id, question_id, order, randomize
  - Add indexes on (created_by, status, scheduled_start)
  - Define relationship: Exam has many ExamQuestions and Questions
- **Deliverables:** SQLAlchemy models, migration file
- **Effort:** 2 hours
- **Dependencies:** Task 1.1.4

Task 3.2.2: Exam CRUD API Endpoints

- **Objective:** Create REST endpoints for exam management
- **Details:**
  - POST `/api/v1/exams` - create new exam (teacher only)
  - GET `/api/v1/exams/{exam_id}` - fetch exam with questions
  - PUT `/api/v1/exams/{exam_id}` - update exam (owner only)
  - DELETE `/api/v1/exams/{exam_id}` - delete exam (owner/admin)
  - GET `/api/v1/exams?created_by=X&amp;status=Y` - list exams with filters
  - POST `/api/v1/exams/{exam_id}/publish` - publish exam (owner only)

- **Deliverables:** Exam endpoints, schemas
- **Effort:** 4 hours
- **Dependencies:** Task 3.2.1, Task 1.1.5

Task 3.2.3: Exam Scheduling & Time Management

- **Objective:** Implement exam scheduling logic
- **Details:**
  - Create POST `/api/v1/exams/{exam_id}/schedule` endpoint to set exact times
  - Add validation: scheduled_start < scheduled_end, duration is positive
  - Implement GET `/api/v1/exams/available` to list exams accessible to current student
  - Add time zone awareness (store in UTC, display in user's local time)
  - Create background job to mark exams as "live" when scheduled_start passes
- **Deliverables:** Scheduling service, background job
- **Effort:** 3 hours
- **Dependencies:** Task 3.2.2

Task 3.2.4: Exam Template & Exam Builder UI

- **Objective:** Build React interface for creating/editing exams
- **Details:**
  - Create ExamBuilderPage with wizard steps: Basic Info → Add Questions → Set Timing → Review & Publish
  - Step 1: Title, description, duration, instructions
  - Step 2: Search and add questions from question bank, set order, randomize toggle
  - Step 3: Set scheduled_start, scheduled_end, allow_late_entry toggle
  - Step 4: Preview exam structure, publish button
  - Add auto-save draft functionality
- **Deliverables:** ExamBuilderPage component with step management
- **Effort:** 6 hours
- **Dependencies:** Task 1.1.2, Task 3.2.2

Task 3.2.5: Exam Assignment to Students

- **Objective:** Assign exams to candidates
- **Details:**
  - Create POST `/api/v1/exams/{exam_id}/assign` endpoint
  - Accept list of student IDs or class/group identifiers
  - Validate student existence

    - Create ExamAssignment records
    - Create GET `/api/v1/exams/{exam_id}/assignments` to view assigned students
    - Implement bulk assignment UI in ExamBuilderPage (Step 4)
- **Deliverables:** Assignment endpoints and UI component
- **Effort:** 3 hours
- **Dependencies:** Task 3.2.4

PHASE 4: EXAM DELIVERY & PROCTORING (Weeks 5-6)

Epic 4.1: Exam Interface & Question Rendering

Task 4.1.1: Exam Portal & Question Display

- **Objective:** Build student exam-taking interface
- **Details:**
    - Create ExamPortal page that displays
        - Question text, type-specific UI (MCQ options, text area for descriptive, code editor for coding)
        - Question counter (e.g., "5 of 50")
        - Timer showing remaining time
        - Navigation buttons (Previous, Next)
        - Question list sidebar with answer status indicators (answered, unanswered, flagged)
    - Implement question randomization if enabled
    - Store answers locally (sessionStorage) to prevent loss on disconnect
- **Deliverables:** ExamPortal component, question renderer components
- **Effort:** 6 hours
- **Dependencies:** Task 1.1.2, Task 3.2.2

Task 4.1.2: Answer Submission & Validation

- **Objective:** Implement answer capture and submission logic
- **Details:**
    - Create form handling for each question type
    - Auto-save answers after every question change (AJAX)
    - Create POST `/api/v1/submissions/{submission_id}/answers` endpoint
    - Validate answer format per question type
    - Store answers with timestamps for audit

    - Implement "Submit Exam" endpoint that locks submission
- **Deliverables:** Answer submission service, API endpoints
- **Effort:** 4 hours
- **Dependencies:** Task 4.1.1

Task 4.1.3: Exam Timer & Auto-Submission

- **Objective:** Implement countdown timer and automatic submission
- **Details:**
    - Create Timer component showing remaining time in MM:SS format
    - Add warning when 5 minutes remain
    - Implement auto-submit when timer reaches zero
    - Add exam pause/resume functionality (with time deduction)
    - Show current time in student's local timezone
    - Add "Are you sure?" dialog before submission
- **Deliverables:** Timer component, auto-submit service
- **Effort:** 3 hours
- **Dependencies:** Task 4.1.2

Task 4.1.4: Exam Session Initialization

- **Objective:** Start exam and initialize proctoring
- **Details:**
    - Create POST `/api/v1/exams/{exam_id}/start` endpoint
    - Validate: exam is published, student is assigned, scheduled time is within window
    - Create Submission record with start_time
    - Return exam data and questions
    - Initialize proctoring session (start camera/screen capture)
- **Deliverables:** Exam start endpoint, session initialization
- **Effort:** 3 hours
- **Dependencies:** Task 3.2.3, Task 4.1.1

Epic 4.2: Real-Time Proctoring & AI Cheating Detection

Task 4.2.1: Proctoring Service Setup & WebRTC Configuration

- **Objective:** Set up real-time video streaming for proctoring
- **Details:**
  - Create Python FastAPI WebSocket endpoint `/ws/proctoring/{submission_id}`
  - Configure WebRTC for browser-to-server video streaming
  - Implement media stream capture: camera, microphone, screen
  - Add permission requests and error handling
  - Create ProctorEvent model to log all events
- **Deliverables:** WebSocket endpoint, WebRTC configuration, ProctorEvent model
- **Effort:** 6 hours
- **Dependencies:** Task 1.1.3, Task 4.1.4

Task 4.2.2: Face Recognition & Identity Verification

- **Objective:** Implement AI-based face verification at exam start
- **Details:**
  - Integrate OpenCV and face_recognition library
  - Capture student's face image at exam start
  - Compare with stored ID photo from registration
  - Return match confidence score
  - Flag if confidence < threshold (e.g., 80%)
  - Create ProctorEvent with event_type="face_verification"
- **Deliverables:** Face recognition service, OpenCV pipeline
- **Effort:** 5 hours
- **Dependencies:** Task 4.2.1

Task 4.2.3: Eye Gaze Tracking & Attention Monitoring

- **Objective:** Implement gaze-tracking for distraction detection
- **Details:**
  - Integrate MediaPipe face landmarks for gaze detection
  - Track eye position and gaze direction in real time
  - Flag if gaze leaves screen for >5 seconds
  - Flag if blink rate abnormal (potential proxy/cheating)
  - Create ProctorEvent with event_type="gaze_anomaly"
  - Log confidence scores for post-exam review
- **Deliverables:** Gaze tracking service, MediaPipe integration

- **Effort:** 5 hours
- **Dependencies:** Task 4.2.1

Task 4.2.4: Multiple Face & Person Detection

- **Objective:** Detect unauthorized people in exam environment
- **Details:**
  - Use YOLOv8 for multi-person detection
  - Flag if >1 person detected in view
  - Create ProctorEvent with event_type="multiple_faces"
  - Store confidence scores and box coordinates for review
  - Implement cooldown to prevent alert spam
- **Deliverables:** YOLOv8 person detection service
- **Effort:** 3 hours
- **Dependencies:** Task 4.2.1

Task 4.2.5: Screen Activity Monitoring

- **Objective:** Monitor screen sharing and application usage
- **Details:**
  - Capture screen in real time (no recording, just analysis)
  - Use pytesseract to detect text/images on screen
  - Flag if browser tab changes or unauthorized app opens
  - Create browser extension to monitor tab switches
  - Create ProctorEvent with event_type="screen_anomaly"
  - Log suspicious patterns (copy-paste attempts, searches, etc.)
- **Deliverables:** Screen monitoring service, browser extension
- **Effort:** 6 hours
- **Dependencies:** Task 4.2.1

Task 4.2.6: Audio Analysis & Anomaly Detection

- **Objective:** Monitor audio for suspicious patterns
- **Details:**
  - Capture audio stream from microphone
  - Detect voices (speech) vs silence vs noise
  - Flag if multiple voices detected (group cheating)
  - Flag if excessive background noise

    - Create ProctorEvent with event_type="audio_anomaly"
    - Use librosa for audio analysis
- **Deliverables:** Audio analysis service, librosa integration
- **Effort:** 4 hours
- **Dependencies:** Task 4.2.1

Task 4.2.7: Proctoring Event Logging & Alert System

- **Objective:** Log all proctoring events and notify proctor
- **Details:**
    - Create service to log ProctorEvent records with timestamp, event_type, confidence, details
    - Implement real-time WebSocket notification to proctor dashboard
    - Create severity levels: LOW (informational), MEDIUM (suspicious), HIGH (likely cheating)
    - Store event media (frames, audio clips) for post-exam review
    - Add alert threshold logic (e.g., 3 HIGH events = block exam)
- **Deliverables:** Event logging service, alert system
- **Effort:** 4 hours
- **Dependencies:** Task 4.2.2 through 4.2.6

Task 4.2.8: Proctor Live Monitoring Dashboard

- **Objective:** Build real-time proctor interface
- **Details:**
    - Create ProctorDashboard showing:
        - List of live exams with candidates
        - Live video feed of selected candidate
        - Screen share from candidate
        - Real-time alert log (scrollable)
        - Buttons to flag suspicious activity, pause exam, chat with candidate
    - Implement WebSocket connection to receive live events
    - Add filtering by event severity (High, Medium, Low)
- **Deliverables:** ProctorDashboard component, WebSocket integration
- **Effort:** 5 hours
- **Dependencies:** Task 1.1.2, Task 4.2.7

PHASE 5: EXAM ANALYTICS & RESULT PROCESSING (Weeks 7-8)

Epic 5.1: Result Processing & Grading

Task 5.1.1: Automated Grading Service (Objective Questions)

- **Objective:** Auto-grade MCQ and true/false questions
- **Details:**
  - Create GradingService class with grade_submission(submission_id) method
  - For each Answer, compare response to Question.correct_answer
  - Calculate score based on points and negative_marking if applicable
  - Support weighted sections (e.g., Section A = 40%, Section B = 60%)
  - Store score in Answer.score field
  - Create batch grading job to process all submitted answers
- **Deliverables:** Grading service, batch grading job
- **Effort:** 3 hours
- **Dependencies:** Task 4.1.2, Task 1.1.4

Task 5.1.2: Manual Grading Interface (Subjective Questions)

- **Objective:** Build UI for teachers to grade subjective answers
- **Details:**
  - Create GradingQueuePage showing unanswered subjective questions
  - Implement GradingForm with:
    - Display student's answer
    - Input field for score/marks
    - Comments/feedback textarea
    - Save button, next/previous navigation
  - Create PUT `/api/v1/submissions/{submission_id}/answers/{answer_id}/grade` endpoint
  - Track grading progress (e.g., "5 of 20 graded")
- **Deliverables:** GradingQueuePage component, grading endpoint
- **Effort:** 4 hours
- **Dependencies:** Task 1.1.2, Task 5.1.1

Task 5.1.3: Result Calculation & Publishing

- **Objective:** Calculate final results and publish to students
- **Details:**
  - Create ResultProcessingService that:
    - Sums all answers' scores to get final_score
    - Calculates percentage_score and grade (A, B, C, etc.) based on grade boundaries
    - Handles bonus/penalty marks
    - Stores in Result table
  - Create PUT `/api/v1/exams/{exam_id}/publish-results` endpoint (teacher only)
  - Add scheduled job to auto-publish results after deadline
  - Send email notifications to students when results published
- **Deliverables:** Result processing service, publishing endpoint
- **Effort:** 3 hours
- **Dependencies:** Task 5.1.2

Task 5.1.4: Result Transcript & Certificate Generation

- **Objective:** Generate PDF transcripts and certificates
- **Details:**
  - Create TranscriptGenerator service using reportlab or weasyprint
  - Generate PDF with: exam name, date, score, grade, cutoff, percentile
  - Add option to sign PDF with digital signature
  - Create CertificateGenerator for passing students
  - Implement GET `/api/v1/submissions/{submission_id}/transcript` endpoint to download PDF
  - Add email delivery of transcript
- **Deliverables:** Transcript/certificate generation service, PDF generation
- **Effort:** 4 hours
- **Dependencies:** Task 5.1.3

Task 5.1.5: Student Result View Page

- **Objective:** Build UI for students to view results
- **Details:**
  - Create ResultsPage showing:
    - List of completed exams with scores, grades
    - Details page for each exam showing:

            - Final score and grade
            - Correct/incorrect count for MCQs
            - Detailed answer review (question, your answer, correct answer)
            - Download transcript button
    - Implement GET `/api/v1/submissions/{submission_id}` to fetch submission details
    - Add search/filter by date, subject
- **Deliverables:** ResultsPage component, submission details endpoint
- **Effort:** 3 hours
- **Dependencies:** Task 1.1.2, Task 5.1.3

Epic 5.2: Analytics & Reporting

Task 5.2.1: Analytics Data Aggregation Service

- **Objective:** Aggregate exam performance data for analytics
- **Details:**
    - Create AnalyticsService that computes:
        - Per-exam stats: avg_score, pass_rate, time_taken, attempt_count
        - Per-question stats: difficulty_index, discrimination_index, pass_rate
        - Per-student stats: score trend, time trend, accuracy trend
        - Department-level aggregate: avg_score by subject, by batch
    - Implement caching (e.g., Redis) to speed up queries
    - Create background job to recompute analytics nightly
- **Deliverables:** Analytics service, caching layer
- **Effort:** 5 hours
- **Dependencies:** Task 5.1.1, Task 1.1.4

Task 5.2.2: Analytics Dashboard Backend (APIs)

- **Objective:** Create endpoints for analytics data
- **Details:**
    - GET `/api/v1/exams/{exam_id}/analytics` - exam-level stats
    - GET `/api/v1/questions/{question_id}/analytics` - question stats
    - GET `/api/v1/analytics/department` - department aggregates
    - GET `/api/v1/analytics/export` - export analytics as CSV/Excel
    - All endpoints support filtering by date range, department, subject
- **Deliverables:** Analytics endpoints

- **Effort:** 3 hours
- **Dependencies:** Task 5.2.1

Task 5.2.3: Teacher Analytics Dashboard UI

- **Objective:** Build analytics visualization for teachers
- **Details:**
  - Create AnalyticsDashboard page with:
    - Chart 1: Score distribution (histogram)
    - Chart 2: Pass/fail pie chart
    - Chart 3: Question performance (bar chart by question)
    - Chart 4: Time spent by question
    - Table: Top performers, struggling students
  - Use Recharts or D3.js for visualizations
  - Add filters: date range, sections, class
  - Add export to CSV button
- **Deliverables:** AnalyticsDashboard component, chart components
- **Effort:** 5 hours
- **Dependencies:** Task 1.1.2, Task 5.2.2

Task 5.2.4: Department Administrator Analytics

- **Objective:** Build cross-exam analytics for admins
- **Details:**
  - Create AdminAnalyticsPage showing:
    - System-wide performance trends
    - Department comparisons
    - Question quality analysis (difficulty, discrimination)
    - Proctor activity report (events, flags per exam)
    - User engagement metrics
  - Implement drill-down capability (click department → exams → questions)
- **Deliverables:** AdminAnalyticsPage component
- **Effort:** 4 hours
- **Dependencies:** Task 1.1.2, Task 5.2.2

PHASE 6: NOTIFICATIONS & REMINDERS (Weeks 8-9)

Epic 6.1: Multi-Channel Notifications

Task 6.1.1: Notification Service Architecture

- **Objective:** Build notification delivery system
- **Details:**
  - Create NotificationService with methods:
    - `send_email(recipient, subject, body, template)`
    - `send_sms(phone_number, message)`
    - `send_push_notification(user_id, title, body)`
  - Integrate with SendGrid (email) and Twilio (SMS)
  - Use Firebase Cloud Messaging for push notifications
  - Implement queue-based delivery (Celery + Redis) for scalability
  - Add retry logic with exponential backoff
  - Create Notification model to track sent notifications
- **Deliverables:** Notification service, queue integration
- **Effort:** 5 hours
- **Dependencies:** Task 1.1.3, Task 1.1.6

Task 6.1.2: Exam Reminder Scheduling

- **Objective:** Schedule and send exam reminders
- **Details:**
  - Create background job that runs every minute
  - Query exams with scheduled_start within next 30 minutes
  - Fetch assigned students and send reminder notifications
  - Support configurable reminder times (1 day, 1 hour, 30 min before)
  - Create RemindersConfiguration table to store reminder rules
  - Add UI for teachers to customize reminder templates
- **Deliverables:** Reminder scheduling job, configuration endpoints
- **Effort:** 3 hours
- **Dependencies:** Task 6.1.1, Task 3.2.3

Task 6.1.3: Notification Preferences & Unsubscribe

- **Objective:** Allow users to manage notification preferences
- **Details:**
    - Create NotificationPreference model: user_id, channel (email/sms/push), enabled, frequency
    - Create PUT `/api/v1/users/notification-preferences` endpoint
    - Build NotificationSettingsPage in frontend
    - Add unsubscribe link in email templates
    - Respect preferences when sending notifications
- **Deliverables:** NotificationPreference model, settings UI and endpoints
- **Effort:** 3 hours
- **Dependencies:** Task 6.1.2

Task 6.1.4: Result Notification

- **Objective:** Notify students when results are published
- **Details:**
    - Create notification trigger when Result is published
    - Send email with score, grade, transcript attachment
    - Include link to view detailed result in portal
    - Add in-app notification
    - Create POST `/api/v1/exams/{exam_id}/send-results-notification` endpoint (admin/teacher)
- **Deliverables:** Result notification service, trigger logic
- **Effort:** 2 hours
- **Dependencies:** Task 6.1.1, Task 5.1.3

PHASE 7: AUDIT LOGGING & COMPLIANCE (Weeks 9-10)

Epic 7.1: Comprehensive Audit Logging

Task 7.1.1: Audit Log Model & Middleware

- **Objective:** Log all system actions for compliance
- **Details:**
    - Create AuditLog model: id, user_id, action, resource_type, resource_id, timestamp, ip_address, user_agent, details (JSON)
    - Create AuditMiddleware that logs all API calls

    - Log before/after state changes (for updates/deletes)
    - Implement immutable storage (write-once-read-many) using S3 Object Lock
    - Set retention policy (7 years per Indian regulations)
    - Create search/export functionality for audits
- **Deliverables:** AuditLog model, middleware, immutable storage integration
- **Effort:** 4 hours
- **Dependencies:** Task 1.1.3, Task 1.1.4

Task 7.1.2: Audit Log Viewer & Export (Admin Only)

- **Objective:** Build UI for admins to review audit logs
- **Details:**
    - Create AuditLogViewerPage with:
        - Table showing all audit logs
        - Filters: date range, user, action, resource type
        - Search by resource ID
        - View detail modal showing full action details
    - Create endpoints: GET `/api/v1/audit-logs`, GET `/api/v1/audit-logs/export`
    - Support export to CSV with all log data
- **Deliverables:** AuditLogViewerPage component, export endpoints
- **Effort:** 3 hours
- **Dependencies:** Task 1.1.2, Task 7.1.1

Task 7.1.3: Proctoring Event Audit Trail

- **Objective:** Create immutable record of all proctoring events
- **Details:**
    - Log all ProctorEvents to AuditLog
    - Store media artifacts (screenshots, audio clips) in immutable storage
    - Create endpoints to retrieve archived proctoring data for dispute resolution
    - Implement access control: only relevant proctor, teacher, admin can view
    - Create report generation for suspected cheating incidents
- **Deliverables:** ProctorEvent logging integration, report generation
- **Effort:** 3 hours
- **Dependencies:** Task 7.1.1, Task 4.2.7

Task 7.1.4: Compliance Report Generation

- **Objective:** Generate compliance reports per regulations
- **Details:**
  - Create ComplianceReporter service that generates:
    - Data inventory report (what data is stored where)
    - Access report (who accessed what, when)
    - Incident report (suspected cheating, system errors)
    - Data retention report (what's being archived)
  - Create endpoints to download reports (admin only)
  - Schedule monthly report generation
- **Deliverables:** Compliance report service, scheduled job
- **Effort:** 3 hours
- **Dependencies:** Task 7.1.1

PHASE 8: DEPLOYMENT & INFRASTRUCTURE (Weeks 10-11)

Epic 8.1: Containerization & Serverless Deployment

Task 8.1.1: Docker Containerization (Backend)

- **Objective:** Containerize FastAPI backend
- **Details:**
  - Create Dockerfile for Python backend
  - Use python:3.11-slim as base image
  - Install dependencies from requirements.txt
  - Set working directory, expose port 8000
  - Use gunicorn for production WSGI server
  - Create docker-compose.yml for local development (backend + PostgreSQL + Redis)
- **Deliverables:** Dockerfile, docker-compose.yml, build scripts
- **Effort:** 2 hours
- **Dependencies:** Task 1.1.3

Task 8.1.2: Docker Containerization (Frontend)

- **Objective:** Containerize React frontend
- **Details:**
  - Create Dockerfile for React app
  - Multi-stage build: build stage (node) → runtime stage (nginx)
  - Copy built React app to nginx wwwroot
  - Configure nginx to serve index.html for SPA routing
  - Expose port 3000 (dev) or 80 (prod)
- **Deliverables:** Dockerfile for frontend
- **Effort:** 2 hours
- **Dependencies:** Task 1.1.2

Task 8.1.3: AWS Lambda Deployment Setup

- **Objective:** Configure serverless deployment to AWS Lambda
- **Details:**
  - Create AWS SAM template (template.yaml) defining Lambda functions, API Gateway, RDS, S3
  - Define Lambda layers for dependencies (e.g., opencv, tensorflow)
  - Configure environment variables and secrets manager integration
  - Set up CloudWatch logs and alarms
  - Create deployment script using AWS CLI / AWS SAM CLI
- **Deliverables:** SAM template, deployment scripts
- **Effort:** 4 hours
- **Dependencies:** Task 8.1.1, Task 1.1.6

Task 8.1.4: RDS Database Setup (AWS)

- **Objective:** Provision managed database in AWS
- **Details:**
  - Create RDS PostgreSQL instance with:
    - Multi-AZ for high availability
    - Read replicas for analytics queries
    - Automated backups (7-day retention)
    - Encryption at rest
  - Create Aurora read replica for scalability
  - Set up DMS for migration if needed

    - Create Terraform script for infrastructure-as-code (IaC)
- **Deliverables:** RDS configuration, Terraform scripts
- **Effort:** 3 hours
- **Dependencies:** Task 1.1.4

Task 8.1.5: S3 & CloudFront Setup (Static Assets & Media)

- **Objective:** Set up object storage and CDN
- **Details:**
    - Create S3 buckets: one for frontend assets, one for proctoring media
    - Enable versioning and lifecycle policies
    - Set up CloudFront distribution for fast content delivery
    - Enable CORS for cross-origin requests
    - Configure bucket policies for security
    - Set up Object Lock for immutable audit logs storage
- **Deliverables:** S3 configuration, CloudFront setup
- **Effort:** 2 hours
- **Dependencies:** Task 8.1.1, Task 8.1.2

Task 8.1.6: CI/CD Pipeline (GitHub Actions)

- **Objective:** Automate build and deployment
- **Details:**
    - Create .github/workflows/deploy.yml
    - Trigger on push to main branch
    - Steps:
        1. Checkout code
        2. Run unit tests (pytest for backend, jest for frontend)
        3. Build Docker images
        4. Push to ECR (Elastic Container Registry)
        5. Deploy to Lambda using SAM CLI
        6. Run smoke tests
    - Add manual approval gate for production
- **Deliverables:** GitHub Actions workflow file
- **Effort:** 3 hours
- **Dependencies:** Task 8.1.1, Task 8.1.2

Task 8.1.7: Environment Management (Dev, Staging, Prod)

- **Objective:** Set up separate environments
- **Details:**
  - Create separate AWS accounts/regions for dev, staging, prod
  - Use Terraform workspaces or separate stack for each environment
  - Configure environment-specific variables (database URLs, API keys)
  - Set up cross-environment promotion pipeline
  - Implement secrets management via AWS Secrets Manager
- **Deliverables:** Environment configuration, promotion workflow
- **Effort:** 2 hours
- **Dependencies:** Task 8.1.4, Task 8.1.5

PHASE 9: TESTING & QA (Weeks 11-12)

Epic 9.1: Comprehensive Testing

Task 9.1.1: Unit Tests (Backend)

- **Objective:** Write unit tests for all backend services
- **Details:**
  - Create tests for:
    - Authentication service (token generation, validation)
    - Grading service (score calculation)
    - Notification service (message formatting)
    - Analytics service (aggregation logic)
  - Use pytest framework
  - Aim for >80% code coverage
  - Mock external services (SendGrid, Twilio)
- **Deliverables:** pytest files, CI integration
- **Effort:** 6 hours
- **Dependencies:** Task 1.1.3

Task 9.1.2: Integration Tests (Backend)

- **Objective:** Test API endpoints end-to-end
- **Details:**
  - Create integration tests for all major flows:
    - User registration → login → create exam → start exam → submit exam → view results
    - Teacher create question → add to exam → grade submission
    - Admin view audit logs → export
  - Use test database (separate SQLite or Postgres container)
  - Test error scenarios (invalid inputs, missing permissions)
- **Deliverables:** Integration test files
- **Effort:** 6 hours
- **Dependencies:** Task 9.1.1

Task 9.1.3: Frontend Component Tests

- **Objective:** Test React components
- **Details:**
  - Use Jest and React Testing Library
  - Test key components:
    - LoginPage (form submission, error handling)
    - ExamPortal (question rendering, timer, answer submission)
    - GradingQueue (display, scoring, navigation)
    - AnalyticsDashboard (chart rendering)
  - Mock API calls using jest.mock
  - Aim for >70% coverage
- **Deliverables:** Jest test files
- **Effort:** 6 hours
- **Dependencies:** Task 1.1.2

Task 9.1.4: End-to-End (E2E) Tests

- **Objective:** Test complete user journeys
- **Details:**
  - Use Cypress or Selenium
  - Test scenarios:
    - Student: register → login → view dashboard → start exam → submit answers → view results

        - Teacher: login → create exam → assign to students → view submissions → grade → publish results
        - Admin: login → manage users → view audit logs → export
    - Run E2E tests in CI/CD pipeline
- **Deliverables:** Cypress/Selenium test files
- **Effort:** 6 hours
- **Dependencies:** Task 9.1.2, Task 9.1.3

Task 9.1.5: Performance Testing & Load Testing

- **Objective:** Validate system handles 1,000 concurrent users
- **Details:**
    - Use Apache JMeter or Locust
    - Create load test scenarios:
        - 1,000 concurrent students taking exam
        - Peak load during result publishing (1,000 downloads simultaneously)
        - Analytics query performance under load
    - Monitor response times, CPU, memory, database connections
    - Identify bottlenecks and optimize
    - Target: p95 latency <200ms, p99 latency <500ms
- **Deliverables:** Load test scripts, performance report
- **Effort:** 6 hours
- **Dependencies:** Task 5.1.1, Task 5.2.2

Task 9.1.6: Security Testing

- **Objective:** Identify and fix security vulnerabilities
- **Details:**
    - Use OWASP ZAP or Burp Suite for vulnerability scanning
    - Test for:
        - SQL injection (parameterized queries check)
        - XSS (input validation, output encoding)
        - CSRF (token validation)
        - Broken authentication (JWT validation, session management)
        - Sensitive data exposure (encryption check)
    - Implement fixes and re-test
    - Generate security report

- **Deliverables:** Security test report, vulnerability fixes
- **Effort:** 6 hours
- **Dependencies:** Task 1.1.5, Task 7.1.1

Task 9.1.7: Accessibility Testing (WCAG 2.1 AA)

- **Objective:** Ensure system is accessible to all users
- **Details:**
  - Use axe DevTools or WAVE browser extensions
  - Test for:
    - Color contrast (4.5:1 for normal text)
    - Keyboard navigation (tab order, focus visible)
    - Screen reader compatibility
    - Form labels and error messages
  - Fix accessibility issues
- **Deliverables:** Accessibility test report, fixes
- **Effort:** 4 hours
- **Dependencies:** Task 1.1.2

PHASE 10: DOCUMENTATION & DEPLOYMENT (Week 13)

Epic 10.1: Documentation

Task 10.1.1: API Documentation (OpenAPI/Swagger)

- **Objective:** Document all REST API endpoints
- **Details:**
  - Use FastAPI auto-generated Swagger UI at /docs
  - Document each endpoint: description, parameters, request/response schemas, error codes
  - Include authentication requirements and role-based access notes
  - Add example payloads and responses
  - Export as OpenAPI 3.0.0 YAML/JSON
- **Deliverables:** Swagger/OpenAPI documentation
- **Effort:** 3 hours
- **Dependencies:** Task 1.1.3

Task 10.1.2: User & Technical Documentation

- **Objective:** Create comprehensive user guides
- **Details:**
  - Create markdown docs:
    - User Manual: student guide, teacher guide, admin guide, proctor guide
    - Technical Architecture: system design, data flow, deployment
    - API Integration Guide: for third-party developers
    - Troubleshooting Guide: common issues and solutions
    - Data Privacy & Compliance: GDPR, SPDI Rules adherence
  - Host docs in GitHub Pages or Confluence
- **Deliverables:** Markdown documentation, hosted site
- **Effort:** 6 hours
- **Dependencies:** All previous phases

Task 10.1.3: Deployment Guide

- **Objective:** Provide step-by-step deployment instructions
- **Details:**
  - Document:
    - Prerequisites (AWS account, CLI setup, etc.)
    - Environment variable configuration
    - Database migration steps
    - SAM deployment commands
    - Post-deployment verification
    - Rollback procedures
    - Monitoring and alerting setup
- **Deliverables:** Deployment guide document
- **Effort:** 3 hours
- **Dependencies:** Task 8.1.1 through 8.1.7

Task 10.1.4: Video Tutorials & Onboarding

- **Objective:** Create video walkthroughs for users
- **Details:**
  - Produce videos:
    - "Getting Started as a Teacher" (creating questions, exams, grading)
    - "Taking an Exam as a Student" (login, exam interface, submission)

        - "Monitoring Exams as Proctor" (dashboard, alerts, flagging)
        - "System Administration" (user management, audit logs)
    - Host on YouTube or internal video server
    - Embed in help pages
- **Deliverables:** Video files, embedded help pages
- **Effort:** 6 hours
- **Dependencies:** All components

Epic 10.2: Production Deployment & Rollout

Task 10.2.1: Pilot Launch (Test Institution)

- **Objective:** Deploy to test institution for validation
- **Details:**
    - Select one college/department for pilot
    - Deploy all services to AWS Lambda and RDS
    - Conduct user acceptance testing (UAT) with teachers, students, admins
    - Gather feedback on usability, performance, issues
    - Monitor system metrics (uptime, response time, errors)
    - Fix critical bugs before full rollout
- **Deliverables:** Pilot deployment, UAT report
- **Effort:** 2 weeks (ongoing)
- **Dependencies:** Task 8.1.1 through 10.1.4

Task 10.2.2: Full Production Deployment

- **Objective:** Deploy to production for all institutions
- **Details:**
    - Execute SAM deployment to production Lambda stack
    - Run database migrations
    - Configure DNS and SSL certificates
    - Enable monitoring and alerting in CloudWatch
    - Conduct smoke tests
    - Notify users of deployment
- **Deliverables:** Production deployment, monitoring dashboards
- **Effort:** 1 day
- **Dependencies:** Task 10.2.1

Task 10.2.3: User Training & Support

- **Objective:** Train users and set up support channels
- **Details:**
  - Conduct live training webinars for teachers, admins
  - Distribute training materials and video tutorials
  - Set up support email and ticketing system
  - Create FAQ and help center
  - Assign support team to monitor and respond to issues
- **Deliverables:** Training webinars, support infrastructure
- **Effort:** 1 week
- **Dependencies:** Task 10.1.2, Task 10.1.4

Task 10.2.4: Performance Monitoring & Optimization

- **Objective:** Monitor production system and optimize
- **Details:**
  - Set up CloudWatch dashboards for:
    - Lambda function duration, errors, concurrency
    - API Gateway latency, requests
    - RDS CPU, connections, query performance
    - S3 GET/PUT latency
  - Set up alarms for anomalies
  - Analyze logs weekly for optimization opportunities
  - Implement caching improvements (Redis for repeated queries)
  - Optimize database queries (add indexes)
- **Deliverables:** Monitoring dashboards, optimization report
- **Effort:** Ongoing (4 hours/week after launch)
- **Dependencies:** Task 8.1.1 through 10.2.2

PHASE 11: ONGOING MAINTENANCE & ITERATION (Post-Launch)

Epic 11.1: Bug Fixes & Hot Fixes

Task 11.1.1: Critical Bug Response

- **Objective:** Address urgent production issues
- **Details:**
    - Maintain on-call rotation for critical bugs
    - Response time: <1 hour for SEV1 (system down), <4 hours for SEV2 (feature broken)
    - Implement hotfix branch, merge to main, deploy to production
    - Post-incident review for prevention
- **Deliverables:** Hotfix process documentation
- **Effort:** As needed (reactive)
- **Dependencies:** All phases

Epic 11.2: Feature Enhancements

Task 11.2.1: Mobile App (React Native)

- **Objective:** Extend platform to mobile (optional)
- **Details:**
    - Create React Native app for iOS/Android
    - Implement core flows: login, take exam, view results
    - Offline mode for exam taking with sync on reconnect
    - Push notifications for reminders and results
- **Deliverables:** React Native app, app store deployment
- **Effort:** 4-6 weeks (post-launch)
- **Dependencies:** Task 1.1.2, Task 6.1.1

Task 11.2.2: Adaptive Assessment Engine

- **Objective:** Add difficulty-adaptive questions (optional)
- **Details:**
    - Implement Item Response Theory (IRT) model
    - Adjust question difficulty based on student performance
    - Recommend practice questions for weak areas
    - Track learning curves over time
- **Deliverables:** IRT implementation, adaptive question engine
- **Effort:** 4-6 weeks (post-launch)
- **Dependencies:** Task 5.2.1

SUMMARY OF DELIVERABLES BY PHASE

Phase	Duration	Key Deliverables
1	Weeks 1-2	Project scaffold, frontend/backend setup, database schema, auth framework
2	Weeks 2-3	User registration/login, role-based dashboards
3	Weeks 3-4	Question management, exam creation, scheduling
4	Weeks 5-6	Exam interface, proctoring pipeline, cheating detection
5	Weeks 7-8	Auto-grading, manual grading UI, analytics dashboards
6	Weeks 8-9	Notification service, reminder scheduling
7	Weeks 9-10	Audit logging, compliance reporting
8	Weeks 10-11	Docker, Lambda deployment, CICD pipeline
9	Weeks 11-12	Unit tests, integration tests, E2E tests, performance testing
10	Week 13	API documentation, user guides, production deployment
11	Post-launch	Maintenance, bug fixes, feature enhancements

TECHNOLOGY STACK REFERENCE

Frontend:

- React 18 with TypeScript
- Redux Toolkit or Zustand (state management)
- React Router for navigation
- Material-UI or Tailwind CSS for styling
- Recharts for analytics visualization
- Axios for HTTP requests

Backend:

- Python 3.11+
- FastAPI for API framework
- SQLAlchemy for ORM
- PostgreSQL for primary database
- Redis for caching and session management
- Celery for async tasks and background jobs

AI/ML:

- OpenCV for image processing
- MediaPipe for pose/gaze detection
- YOLOv8 for object detection

- TensorFlow for ML inference
- librosa for audio analysis

Notification:

- SendGrid for email
- Twilio for SMS
- Firebase Cloud Messaging for push notifications

Deployment & Infrastructure:

- AWS Lambda for serverless compute
- AWS RDS for managed database
- AWS S3 for object storage
- AWS CloudFront for CDN
- AWS API Gateway for REST API management
- AWS CloudWatch for monitoring
- Docker for containerization
- GitHub Actions for CI/CD

Testing:

- pytest for backend unit tests
- Jest for frontend unit tests
- Cypress for E2E tests
- Apache JMeter for load testing
- OWASP ZAP for security testing

GOOGLE ANTIGRAVITY IDE INTEGRATION POINTS

Where Antigravity IDE Accelerates Development:

1. **Task 1.1.1-1.1.2:** Project Wizard auto-generates MERN scaffold
2. **Task 1.1.3-1.1.4:** Database Designer visually creates schema, auto-generates migration files
3. **Task 2.1.1-2.1.3:** API Builder creates auth endpoints from schema, generates schemas
4. **Task 3.1.2-3.1.4:** CRUD Generator creates question endpoints and React components
5. **Task 4.1.1-4.1.3:** UI Builder creates exam interface with drag-drop components
6. **Task 5.1.1-5.1.5:** Analytics Dashboard Builder creates charts and analytics pages
7. **Task 8.1.1-8.1.6:** Deployment Pipeline Generator creates Docker/SAM/CI-CD files
8. **Task 9.1.1-9.1.4:** Test Generator creates unit/integration/E2E test templates
9. **Task 10.1.1:** Swagger Generator auto-documents APIs from code

KEY MILESTONES & SUCCESS METRICS

✅ **Week 2:** Core infrastructure ready, database connected, initial API functioning
✅ **Week 3:** User authentication working, basic dashboards for all roles
✅ **Week 4:** Question and exam management fully functional
✅ **Week 6:** Exam delivery with proctoring online
✅ **Week 8:** Grading, analytics, and notifications working
✅ **Week 12:** All tests passing, security vulnerabilities fixed
✅ **Week 13:** Production deployment, pilot testing underway

Success Metrics:

- 1,000 concurrent users with <200ms p95 latency ✅
- 99.9% uptime in production ✅
- <1 second MCQ auto-grading ✅
- Email/SMS delivery >99% ✅
- Zero critical security vulnerabilities ✅
- User satisfaction score >4.5/5 ✅

This comprehensive roadmap provides clear, actionable tasks for your MERN-stack SOEMS project, optimized for rapid development in Google Antigravity IDE. Each task includes objectives, technical details, dependencies, and effort estimates to guide your team through 13 weeks of development to a production-ready system.

Ready to begin implementation?

[1]

⁂

1. paste.txt