(TWO-WEEK TIMELINE)

NYS VIRTUAL CAMPUS

Day	Milestone	Description
Day 1–2	Project Setup	Plan architecture, create Git repo, set up Node + React environments
Day 3–4	Backend Auth	Build user registration, login, JWT auth
Day 5–6	Course APIs	CRUD for courses, tutor linking, and student enrollment
Day 7	File Uploads	Integrate Cloudinary for materials & submissions
Day 8–9	Frontend UI	Build login/register pages and dashboard
Day 10–11	Course Management UI	Course list, course details, upload material pages
Day 12	Assignment Module	Add assignment creation and submission features
Day 13	Testing & QA	Test all routes, fix UI bugs, security review
Day 14	Deployment	Deploy backend + frontend, demo to stakeholders

Project overview

Features list
System architecture
Tech stack
Database schema
API endpoints
Frontend structure
Deployment plan
2-week timeline with milestones

PROJECT OVERVIEW

Project Name: NYS Virtual Campus

Goal: To provide a digital learning platform for NYS Kenya where tutors can manage courses

and students can access lessons, assignments, and exams — similar to KCA University's virtual campus.

OBJECTIVES

1. Digitize NYS courses and programs.
2. Allow tutors to upload notes, videos, and assignments.
3. Allow students to register, enroll, learn, and attempt quizzes online.
4. Enable real-time announcements and discussions.
5. Generate progress reports and analytics.

CORE FEATURES

For Tutors:

- Create and manage courses
- Upload materials (PDF, video, docs)
- Post announcements
- Create assignments & quizzes
- Grade submissions

For Students:

- Register & login
- Enroll in courses
- View lessons, notes, and videos

- Submit assignments
- Attempt quizzes
- View progress reports

Admin Panel:

- Manage users (students, tutors)
- Manage departments & courses
- Monitor analytics (active users, submissions, etc.)

SYSTEM ARCHITECTURE

```
Frontend (React.js + Tailwind)
     ↕
Backend API (Node.js/Express)
     ↕
Database (MongoDB)
     ↕
Storage (Cloudinary / AWS S3 for files)
```

Optional: Use **Firebase Authentication** for login and **JWT** for authorization.

TECHNOLOGY STACK

Layer	Technology
Frontend	React.js, Tailwind CSS, Axios, React Router, Redux Toolkit
Backend	Node.js, Express.js, JWT, bcrypt
Database	MongoDB (Mongoose ODM)
File Storage	Cloudinary or AWS S3

Authentication	JWT or Firebase Auth
Deployment	Frontend → Vercel / Netlify Backend → Render / Railway / AWS EC2
Version Control	GitHub
Documentation	Swagger / Postman for APIs

DATABASE SCHEMA (MongoDB)

1. Users

```
{
  _id: ObjectId,
  name: String,
  email: String,
  password: String,
  role: String, // 'student' | 'tutor' | 'admin'
  department: String,
  enrolledCourses: [ObjectId], // course IDs
  createdAt: Date
}
```

2. Courses

```
{
  _id: ObjectId,
  title: String,
  description: String,
  tutor: ObjectId, // User reference
  department: String,
  materials: [ObjectId], // references Materials
  students: [ObjectId],
  createdAt: Date
}
```

3. Materials

```
{
  _id: ObjectId,
  courseId: ObjectId,
  title: String,
  type: String, // 'pdf' | 'video' | 'link'
  fileUrl: String,
  uploadedAt: Date
}
```

4. Assignments

```
{
  _id: ObjectId,
  courseId: ObjectId,
  title: String,
  description: String,
  dueDate: Date,
  submissions: [ObjectId]
}
```

5. Submissions

```
{
  _id: ObjectId,
  assignmentId: ObjectId,
  studentId: ObjectId,
  fileUrl: String,
  grade: Number,
  feedback: String,
  submittedAt: Date
}
```

API ENDPOINTS

Auth Routes

Method	Endpoint	Description

POST	/api/auth/register	Register new user
POST	/api/auth/login	Login user and issue JWT

Course Routes

Method	Endpoint	Description
GET	/api/courses	Get all courses
POST	/api/courses	Create new course (Tutor/Admin)
GET	/api/courses/:id	Get course details
PUT	/api/courses/:id	Update course
DELETE	/api/courses/:id	Delete course

Materials

Method	Endpoint	Description
POST	/api/materials	Upload material
GET	/api/materials/:courseId	Get all materials for a course

Assignments

Method	Endpoint	Description
POST	/api/assignments	Create assignment
GET	/api/assignments/:courseId	Get assignments for course
POST	/api/assignments/submit/:id	Submit assignment

FRONTEND STRUCTURE

```
src/
|
```

```
├── components/
│   ├── Navbar.jsx
│   ├── Sidebar.jsx
│   ├── CourseCard.jsx
│   └── Loader.jsx
│
├── pages/
│   ├── Login.jsx
│   ├── Register.jsx
│   ├── Dashboard.jsx
│   ├── Courses.jsx
│   ├── CourseDetail.jsx
│   ├── Assignments.jsx
│   ├── Profile.jsx
│   └── AdminPanel.jsx
│
├── redux/
│   ├── store.js
│   ├── userSlice.js
│   └── courseSlice.js
│
├── utils/
│   ├── api.js
│   └── auth.js
│
└── App.jsx
```

DEPLOYMENT PLAN

1. **Frontend:**
    - Deploy on **Vercel** or **Netlify**
    - Configure environment variables (API base URL)
2. **Backend:**

- Deploy on **Render**, **Railway**, or **AWS EC2**
- Connect to **MongoDB Atlas**
- Enable CORS for frontend

3. **Domain Integration:**
   - Use **NYS Virtual Campus** subdomain (e.g., `virtual.nys.go.ke`)

FUTURE IMPROVEMENTS (After MVP)

- Add **live virtual classes** (Zoom / WebRTC)
- Add **chat/discussion forum**
- Include **analytics dashboards**
- Integrate **mobile app** using React Native

NYS Virtual Campus — SRS (Software Requirements Specification) & UML Diagrams

Author: Emmanuel Mukasa
Date: 15 October 2025

Table of Contents

    - Scope
    - Definitions, acronyms, abbreviations
    - References
2. Overall Description
    - Product perspective
    - Product functions (high level)
    - User classes and characteristics
    - Operating environment
    - Design and implementation constraints
    - Assumptions and dependencies
3. Functional Requirements (System Features)
    - Authentication & Authorization
    - User Management
    - Course Management
    - Material Management
    - Enrollment & Progress
    - Assignments & Submissions
    - Assessments (Quizzes)
    - Announcements & Notifications
    - Reporting & Analytics
    - Admin Functions
4. External Interface Requirements
    - User interfaces
    - Hardware interfaces
    - Software interfaces (APIs)
    - Communications interfaces
5. Nonfunctional Requirements
    - Performance
    - Security
    - Reliability & Availability
    - Maintainability
    - Usability
    - Scalability
6. Data Requirements and Data Model
    - Main entities
    - Sample schema (MongoDB style)
7. Acceptance Criteria
8. Appendix
    - Glossary
    - Revision history

---

1. Introduction

1.1 Purpose

This SRS describes the requirements for the **NYS Virtual Campus** — a web-based Learning Management System for the National Youth Service (NYS), Kenya. The system will enable NYS tutors to publish course materials, run assessments, manage enrollment, and interact with students. Students can register, enroll in NYS-offered courses, access materials, submit assignments, and track progress. The document is targeted at stakeholders, developers, and QA engineers.

1.2 Scope

The MVP scope (two-week timeline) includes:

- User registration and authentication (students, tutors, admins).
- Course creation and enrollment.
- Upload and serve course materials (PDF, video links, external links).
- Assignment creation, student submission, and grading.
- Quizzes (MCQ timed) with automatic grading.
- Announcements and basic notifications (email / in-app).
- Basic reports: enrollment list, submission list, student progress.

Out of scope for MVP: live video streaming (WebRTC), mobile apps, advanced analytics, SCORM support.

1.3 Definitions, Acronyms, Abbreviations

- LMS — Learning Management System
- NYS — National Youth Service (Kenya)
- MVP — Minimum Viable Product
- API — Application Programming Interface
- JWT — JSON Web Token

1.4 References

- KCA University Virtual Campus (feature inspiration)
- MongoDB documentation
- Cloudinary/AWS S3 docs (for file storage)

---

2. Overall Description

2.1 Product perspective

The NYS Virtual Campus will be a standalone web application with three primary user roles: **Student**, **Tutor**, and **Admin**. The application will use a RESTful backend (Node.js + Express) with MongoDB for storage and a single-page frontend application (React + Tailwind). File storage will be an external service (Cloudinary or S3).

2.2 Product functions (high level)

- Authentication & authorization.
- CRUD on courses and course materials.
- Enrollment management.
- Assignment creation, submission and grading.
- Quiz management and auto-grading for MCQs.
- Announcements / notifications.
- Reporting: course rosters, grades, completion statuses.

2.3 User classes and characteristics

- **Student**: low to moderate technical skill; needs clear, intuitive UI. Can register, enroll, download/view materials, submit assignments, take quizzes.
- **Tutor**: moderate technical skill; create/manage course content, assignments, grade students, post announcements.
- **Admin**: high-level operations—manage users, courses, site settings, and moderate content.

2.4 Operating environment

- Web browsers: Chrome (latest), Firefox (latest), Edge (latest), Safari (latest)
- Server: Node.js 18+ runtime; MongoDB Atlas for production
- Cloud storage (Cloudinary/AWS S3)

2.5 Design and implementation constraints

- GDPR-like privacy considerations for user data in Kenya.
- Use JWT for stateless sessions.
- File uploads limited to acceptable sizes (e.g., 500MB per video; PDF limit 20MB).

- Must support concurrent use for at least 500 active concurrent users for MVP (scale later).

2.6 Assumptions and dependencies

- NYS supplies official course catalog (titles, descriptions) or CSV for import.
- SMTP/email service available (SendGrid or SMTP relay).
- Domain and hosting accounts will be provided by NYS for deployment.

---

3. Functional Requirements (System Features)

Each feature shows: ID, Description, Priority, Preconditions, Postconditions, Functional Flow, Acceptance Criteria.

FR-1: Authentication & Authorization

- **ID:** FR-1
- **Description:** Users must register and login. Roles: student, tutor, admin.
- **Priority:** Must have
- **Preconditions:** None
- **Postconditions:** Authenticated user receives a JWT and role context.
- **Flow:** Register → verify email (optional) → login → JWT issued → access protected endpoints.
- **Acceptance:** User can login and receive JWT; protected endpoints return 401 when unauthorized.

FR-2: User Management

- **ID:** FR-2
- **Description:** Admin can create, edit, deactivate users; tutors can update own profiles.
- **Priority:** Must have
- **Acceptance:** Admin can list users and change roles.

FR-3: Course Management

- **ID:** FR-3

- **Description:** Tutors and Admins can create courses with title, description, department, syllabus, and attach materials.
- **Priority:** Must have
- **Acceptance:** Course shows in public catalog and tutor referenced as owner.

FR-4: Enrollment

- **ID:** FR-4
- **Description:** Students can enroll in available courses; tutors/admins can manually enroll or remove students.
- **Priority:** Must have
- **Acceptance:** Student enrolled shows in course roster; student can access course materials.

FR-5: Material Management

- **ID:** FR-5
- **Description:** Tutors can upload materials (PDFs, links, video URLs) and tag them by module/lesson.
- **Priority:** Must have
- **Acceptance:** Materials download/view and are linked from course detail.

FR-6: Assignments & Submissions

- **ID:** FR-6
- **Description:** Tutors create assignments with due date and instructions; students submit files; tutors grade and provide feedback.
- **Priority:** Must have
- **Acceptance:** Submission timestamp saved; tutor can grade and feedback saved; late submissions marked.

FR-7: Quizzes (MCQs)

- **ID:** FR-7
- **Description:** Tutors create timed MCQ quizzes; system auto-grades; tutor may review/override.
- **Priority:** Must have
- **Acceptance:** Quiz auto-grades and student sees score and explanations if enabled.

FR-8: Announcements & Notifications

- **ID:** FR-8
- **Description:** Tutors/admins post announcements at course-level and system-wide. Notifications appear in-app and optionally via email.
- **Priority:** Should have
- **Acceptance:** Announcement visible in course dashboard; email sent when enabled.

FR-9: Reporting & Analytics

- **ID:** FR-9
- **Description:** Generate course rosters, assignment submission reports, and student progress.
- **Priority:** Should have
- **Acceptance:** Admin/tutor can export roster as CSV and view simple progress metrics.

FR-10: Admin Functions

- **ID:** FR-10
- **Description:** Admin manages departments, course catalog, users, and site settings.
- **Priority:** Must have
- **Acceptance:** Admin panel accessible and actions persist in DB.

---

4. External Interface Requirements

4.1 User Interfaces

- Responsive web UI built with React + Tailwind.
- Role-based dashboards:
    - Student Dashboard: Enrolled courses, upcoming assignments, announcements.
    - Tutor Dashboard: My courses, pending submissions to grade, create content.
    - Admin Dashboard: System metrics, user management.

4.2 Hardware Interfaces

- Standard web hosting/computing environment. No special hardware required.

4.3 Software Interfaces (APIs)

- REST API endpoints (JSON). Example endpoints (URL prefix `/api`):
  - `POST /auth/register`
  - `POST /auth/login`
  - `GET /courses`
  - `POST /courses` (tutor/admin)
  - `POST /courses/:id/enroll` (student or admin)
  - `POST /materials` (multipart/form-data upload)
  - `POST /assignments` (create)
  - `POST /assignments/:id/submit`

Provide OpenAPI/Swagger for full API contract.

4.4 Communications Interfaces

- Email (SMTP/SendGrid) for registration, announcements, grade notifications.
- Optional SMS gateway for urgent alerts (Twilio or local provider).

---

5. Nonfunctional Requirements

5.1 Performance

- Page loads under 2 seconds on broadband (desktop) for cached resources.
- Backend handles 500 concurrent active users for MVP; scalable via horizontal autoscaling.

5.2 Security

- Passwords hashed with bcrypt (salted).
- JWT tokens with reasonable expiry (e.g., access token 1h, refresh token 7d).
- HTTPS required (TLS 1.2+).
- File upload antivirus/malware scanning recommended.
- Role-based access control on endpoints.

5.3 Reliability & Availability

- Uptime target: 99.5% for MVP environment.
- Automated daily backups for database.

5.4 Maintainability

- Clean modular code, documented APIs, linting rules and code style.
- Use CI (GitHub Actions) for tests and linting.

5.5 Usability

- Clear, accessible UI (WCAG AA recommended for future improvement).
- Minimal steps for common tasks (enroll, download material, submit assignment).

5.6 Scalability

- Use stateless backend nodes, MongoDB Atlas, CDN for static content.

---

6. Data Requirements and Data Model

6.1 Main Entities

- User (student/tutor/admin)
- Course
- Material
- Assignment
- Submission
- Quiz
- Enrollment
- Announcement

6.2 Sample MongoDB Schemas

User

```
{
_id: ObjectId,
name: String,
email: String,
passwordHash: String,
role: String, // 'student'|'tutor'|'admin'
department: String,
```

enrolledCourses: [ObjectId],
createdAt: Date
}

Course

{
_id: ObjectId,
title: String,
code: String,
description: String,
tutor: ObjectId,
department: String,
materials: [ObjectId],
assignments: [ObjectId],
quizzes: [ObjectId],
students: [ObjectId],
createdAt: Date
}

Assignment

{
_id: ObjectId,
courseId: ObjectId,
title: String,
description: String,
dueDate: Date,
maxScore: Number,
submissions: [ObjectId]
}

Submission

{
_id: ObjectId,
assignmentId: ObjectId,
studentId: ObjectId,
fileUrl: String,
submittedAt: Date,
grad