Programming Applications and Frameworks (IT3030)

Assignment – 2026 (Semester 1)

Group Coursework (Individual contribution assessed)

Important Details

- **Weight:** 30% of the final mark for IT3030
- **Mode:** Group work; each member is assessed individually (marks may differ within a group)
- **Assignment release date:** 24th March 2026
- **Viva / Demonstration:** Starting 11th April 2026 (TBA)
- **Submission deadline:** 11.45 PM (GMT +5.30), 27th April 2026 via Courseweb
- **Required stack:** Spring Boot REST API + React client web application
- **Version control:** GitHub repository + GitHub Actions workflow required

Assignment Description

Your team has been hired to design and implement a complete, production-inspired web system for a real-world business scenario. You must develop:

- **A Java (Spring Boot) REST API** using RESTful best practices (layered architecture, validation, error handling, security).
- **A React-based client web application** that consumes your API and provides a usable UI for the required workflows.

Business Scenario: Smart Campus Operations Hub

A university is modernizing its day-to-day operations. The university needs a single web platform to manage facility and asset bookings (rooms, labs, equipment) and maintenance/incident handling (fault reports, technician updates, resolutions). The platform must support a clear workflow, role-based access, and strong auditability.

Core Features (Minimum Requirements)

Module A – Facilities & Assets Catalogue

- Maintain a catalogue of bookable resources: lecture halls, labs, meeting rooms, and equipment (projectors, cameras, etc.).
- Each resource must have key metadata (type, capacity, location, availability windows, and status such as ACTIVE / OUT_OF_SERVICE).
- Support search and filtering (e.g., by type, capacity, and location).

Module B – Booking Management

- Users can request a booking for a resource by providing date, time range, purpose, and expected attendees (where applicable).

- Bookings must follow a workflow: PENDING → APPROVED/REJECTED. Approved bookings can later be CANCELLED.
- The system must prevent scheduling conflicts for the same resource (overlapping time ranges).
- Admin users can review, approve, or reject booking requests with a reason.
- Users can view their own bookings; Admin can view all bookings (with filters).

Module C – Maintenance & Incident Ticketing

- Users can create incident tickets for a specific resource/location with category, description, priority, and preferred contact details.
- Tickets can include up to 3 image attachments (evidence such as a damaged projector or error screen).
- Ticket workflow: OPEN → IN_PROGRESS → RESOLVED → CLOSED (Admin may also set REJECTED with reason).
- A technician (or staff member) can be assigned to a ticket and can update status and add resolution notes.
- Users and staff can add comments; comment ownership rules must be implemented (edit/delete as appropriate).

Module D – Notifications

- Users must receive notifications for booking approval/rejection, ticket status changes, and new comments on their tickets.
- Notifications must be accessible through the web UI (e.g., notification panel).

Module E – Authentication & Authorization

- Implement OAuth 2.0 login (e.g., Google sign-in).
- At minimum, support roles: USER and ADMIN. You may add extra roles (e.g., TECHNICIAN / MANAGER) for better design.
- Secure endpoints using role-based access control and protect the front-end routes accordingly.

Note: Your group may extend the system with additional value-adding features, but the minimum requirements must be fully satisfied first.

Tasks in the Assignment

You must complete all of the following:

- **Requirements:** Identify functional requirements for both the REST API and client web application, and define key non-functional requirements (security, performance, scalability, usability).
- **Architecture Design:** Provide an overall system architecture diagram (excluding mobile apps), plus detailed diagrams for the REST API and front-end architectures.
- **Implementation:** Develop the Spring Boot REST API and React web application according to your designs; ensure clean architecture and maintainable code.
- **Testing & Quality:** Provide evidence of testing (unit/integration tests and/or Postman collections) and demonstrate robust validation and error handling.
- **Version Control & CI:** Host the project on GitHub and implement a GitHub Actions workflow (build + test; optionally package/deploy).

Other Requirements

- The project must be version-controlled using Git and hosted on GitHub with an active commit history.

- Each member must implement at least four (4) REST API endpoints using different HTTP methods (GET, POST, PUT/PATCH, DELETE).
- Use consistent API naming, correct HTTP status codes, and meaningful error responses.
- Persist data using a database of your choice (SQL or NoSQL). Do not rely only on in-memory collections.
- Apply security best practices (authentication, authorization, input validation, safe file handling for attachments).
- UI/UX quality matters: your client app must be usable, clear, and logically structured.

Recommended Work Allocation (to support individual assessment)

To make individual contribution visible, allocate modules clearly. For example:

- Member 1: Facilities catalogue + resource management endpoints
- Member 2: Booking workflow + conflict checking
- Member 3: Incident tickets + attachments + technician updates
- Member 4: Notifications + role management + OAuth integration improvements

Your final repository and documentation must clearly indicate which member implemented which endpoints and UI components.

Submission Artifacts

- **GitHub Repository Link:** Public or accessible to evaluators; include a clear README with setup steps.
- **Final Report (PDF):** Requirements, architecture diagrams, endpoint list, testing evidence, and team contribution summary.
- **Running System:** Demonstrable locally; optional deployment is encouraged but not mandatory.
- **Evidence:** Screenshots (or short video link) for key workflows and OAuth login.

Naming & Packaging

- **Report file name:** IT3030_PAF_Assignment_2026_GroupXX.pdf
- **Repository name:** it3030-paf-2026-smart-campus-groupXX
- **Do not include compiled files** (e.g., node_modules, target) in the submission zip.

Examples of Acceptable Innovation (Optional)

- QR code check-in for approved bookings (simple verification screen).
- Admin dashboard with usage analytics (top resources, peak booking hours).
- Service-level timer for tickets (time-to-first-response, time-to-resolution).
- Notification preferences (enable/disable categories).

Academic Integrity & Viva Readiness

This assignment is designed to be assessed individually during progress review and viva. Any evidence of copying, repository duplication, or reusing seniors' solutions will result in severe penalties, including the possibility of zero marks for the affected components.

- Each member must be able to explain their own endpoints, database design, and UI components.
- Ensure commit history reflects true individual work. Avoid single-day bulk commits.
- Keep your README and report consistent with what is implemented.

End of Assignment