**1.1 Project Overview**

**Short Description:**  
The system enables the registration, management, and coordination of dissertation requests between students and professors. Professors define registration sessions, and students submit requests to be supervised. Professors can approve or reject requests, and both parties can exchange signed documents.

**Primary Users:**

* **Student:** Can browse open sessions, submit coordination requests, upload a signed coordination form after approval.
* **Professor:** Can create registration sessions, view incoming requests, approve or reject requests, and upload a response document if needed.

**1.2 Key Features**

1. **User Authentication & Role Management:**
   * Users can log in with a username and password.
   * Each user has a role: student or professor.
   * Authentication via JWT-based token.
2. **Registration Sessions (Professor Only):**
   * Professors can create multiple sessions for dissertation coordination.
   * Each session has a start date, end date, and a maximum number of students that can be approved.
   * Sessions cannot overlap for the same professor.
3. **Dissertation Requests (Student to Professor):**
   * Students can submit multiple requests to different professors during active sessions.
   * A request can have states:
     + pending (initial)
     + approved
     + rejected
   * Once a request is approved by one professor, the student cannot receive approvals from other professors.
4. **Approvals and Rejections:**
   * Professor approves or rejects requests.
   * Approval is subject to max students allowed in that session.
   * Rejection requires a justification.
5. **Document Exchange:**
   * After approval, the student uploads a signed request form (PDF or another allowed format).
   * The professor can respond with a document upload as well.
   * If the professor is not satisfied with the document, they can reject it and request a new upload from the student.
6. **UI/UX Requirements:**
   * Single Page Application with React.
   * Mobile-responsive layout.
   * Clear navigation for both user roles.
   * Error messages, success notifications, and validation feedback.
7. **Data Persistence:**
   * MERN stack: MongoDB for storage.
   * Use Mongoose for modeling data.
   * Store file metadata (file URLs) in the database; actual files can be stored locally or in a cloud bucket.
8. **Deployment:**
   * Deploy backend (Node/Express) and frontend (React) to a cloud service.
   * Environment variables for database connection, JWT secret, and file storage paths.

**1.3 Functional Requirements**

**User & Authentication:**

* **FR-1:** Users must register or be seeded into the system with a role. (Registration feature may be optional if pre-seeded accounts are used.)
* **FR-2:** Users must log in with username and password to receive a JWT.
* **FR-3:** A user’s role determines what features they can access (professors manage sessions and requests, students submit and upload documents).

**Sessions Management (Professor Only):**

* **FR-4:** Professors can create a session with startDate, endDate, and maxStudents.
* **FR-5:** The system must prevent overlapping sessions for the same professor.
* **FR-6:** Professors can view and update their sessions.

**Requests (Student):**

* **FR-7:** Students can view all open sessions (where current date is between startDate and endDate).
* **FR-8:** Students can submit a pending request to a professor’s open session.
* **FR-9:** Students can cancel a pending request before it is decided upon.

**Approvals/Rejections (Professor):**

* **FR-10:** Professors can view all incoming requests for their sessions.
* **FR-11:** Professors can approve a request if they have not exceeded the maxStudents limit.
* **FR-12:** Professors can reject a request, providing a justification.
* **FR-13:** If a request is approved, no other professors can approve that student’s requests.

**Document Uploads:**

* **FR-14:** After approval, the student must upload a signed request document.
* **FR-15:** The professor may upload a document in response.
* **FR-16:** The professor can reject the uploaded student document, requiring re-upload.

**Constraints:**

* **FR-17:** A student cannot be approved by multiple professors simultaneously.
* **FR-18:** Session times and capacities must be respected.
* **FR-19:** Requests can only be created during active sessions.

**1.4 Data Model (MongoDB Collections & Fields)**

**User Collection:**

* \_id: ObjectId
* username: string (unique)
* passwordHash: string
* role: string (enum: student | professor)
* name: string

**Session Collection:**

* \_id: ObjectId
* professorId: ObjectId (ref to User)
* startDate: Date
* endDate: Date
* maxStudents: number
* **Index:** (professorId, startDate, endDate) to prevent overlaps.

**Request Collection:**

* \_id: ObjectId
* studentId: ObjectId (ref to User)
* professorId: ObjectId (ref to User)
* sessionId: ObjectId (ref to Session)
* status: string (enum: pending, approved, rejected)
* justification: string (for rejections)
* studentFileURL: string (for student upload)
* professorFileURL: string (for professor upload)

**1.5 API Endpoints Specification**

**Authentication:**

* POST /api/auth/login
  + **Description:** Authenticates a user.
  + **Body:** { username: string, password: string }
  + **Response:** { token: string, role: string, name: string }
  + **Errors:** 401 if credentials invalid.

**Users (Admin or Setup Purposes) – optional if we seed data:**

* POST /api/users
  + **Description:** Create a new user.
  + **Body:** { username: string, password: string, role: 'student'|'professor', name: string }
  + **Response:** Created user document.
  + **Protected:** Might be restricted (in production) or removed for final version.

**Sessions (Professor Only):**

* GET /api/sessions
  + **Description:** List all sessions (for admin/testing) or by professor filter.
  + **Response:** Array of sessions.
* GET /api/sessions/active
  + **Description:** List all currently active sessions (for students).
  + **Response:** { sessions: [ {id, startDate, endDate, maxStudents, ...} ]}
* POST /api/sessions (Professor only)
  + **Description:** Create a session.
  + **Body:** { startDate: Date, endDate: Date, maxStudents: number }
  + **Response:** Created session.
  + **Errors:** 400 if overlapping session detected.
* PUT /api/sessions/:id (Professor only)
  + **Description:** Update session details.
  + **Body:** Any combination of fields to update.
  + **Response:** Updated session.
* DELETE /api/sessions/:id (Professor only)
  + **Description:** Delete a session (cleanup).
  + **Response:** 200 on success.

**Requests:**

* GET /api/requests
  + **Description:** List requests. Professors see requests made to them, Students see their own requests.
  + **Response:** { requests: [...] } depends on role.
* POST /api/requests (Student only)
  + **Description:** Create a new request to a professor’s active session.
  + **Body:** { professorId: ObjectId, sessionId: ObjectId }
  + **Response:** Created request with status pending.
  + **Errors:** 400 if session not active, if already approved by another professor, or request invalid.
* PUT /api/requests/:id/approve (Professor only)
  + **Description:** Approve a pending request if within capacity.
  + **Response:** Updated request with status: "approved"
  + **Errors:** 400 if capacity exceeded, already approved by another professor, etc.
* PUT /api/requests/:id/reject (Professor only)
  + **Description:** Reject a request.
  + **Body:** { justification: string }
  + **Response:** Updated request with status: "rejected".
* PUT /api/requests/:id/upload-student-file (Student only)
  + **Description:** Upload the student’s signed dissertation form after approval.
  + **Form-data:** file
  + **Response:** Updated request with studentFileURL.
* PUT /api/requests/:id/upload-professor-file (Professor only)
  + **Description:** Professor uploads a document (feedback or next steps).
  + **Form-data:** file
  + **Response:** Updated request with professorFileURL.
* PUT /api/requests/:id/reject-student-file (Professor only)
  + **Description:** Reject the student’s uploaded file, requesting a new one.
  + **Body:** { justification: string }
  + **Response:** Request remains approved but indicates student must re-upload.
  + Note: Could also use justification field or maintain a secondary status. (Exact mechanism to be clarified in implementation.)

**Error Responses:**

* Typically { error: "Message" } with appropriate HTTP status codes.