Module 4 – Backend Development and Logic Implementation

1. Introduction :

The **backend** serves as the **core logic layer** of the **QR-Based Attendance System**, responsible for handling authentication, QR generation, attendance validation, and secure data communication between the frontend and database.

In this module, we design and implement the backend using **Spring Boot (Java)**. This layer integrates with the **MongoDB database** (implemented in Module 3) and exposes **RESTful APIs** to the frontend for seamless communication.

The focus of this module is to ensure that the backend logic is **modular, secure, and scalable**, supporting multiple roles (Admin and Student) and providing real-time attendance tracking through QR codes.

---

2. Objectives :

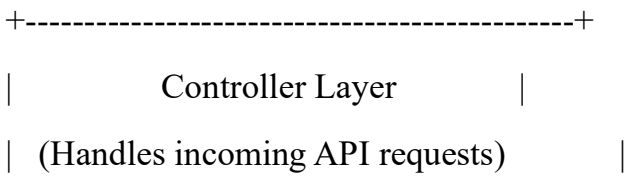
The primary objectives of this module are to:

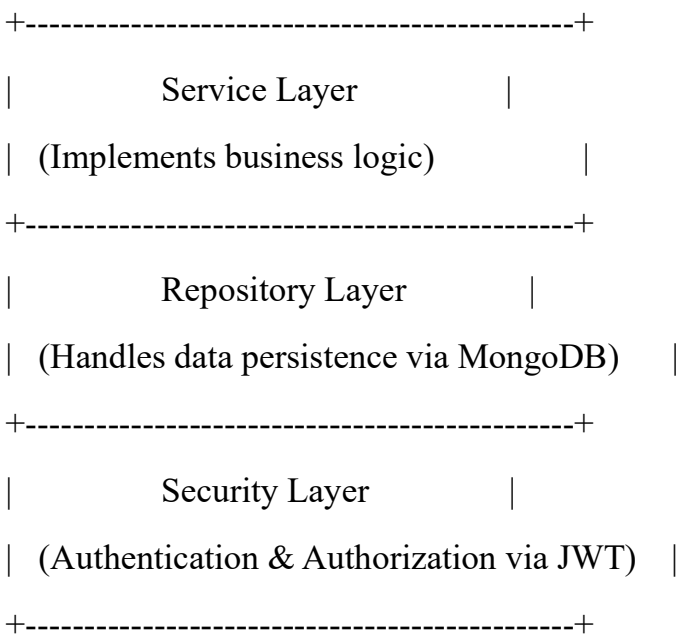
1. Implement the **core backend services** for user, session, and attendance management.
2. Develop **RESTful APIs** for frontend integration.
3. Integrate **QR code generation** and validation functionality.
4. Implement **JWT-based authentication** and role-based access control.
5. Ensure **data validation**, **error handling**, and **security** across all endpoints.

---

3. Backend Architecture Overview :

The backend follows a **modular layered architecture**:





---

4. Technology Stack :

Component	**Technology**
Framework	Spring Boot (Java)
Database	MongoDB
Authentication	Spring Security + JWT
QR Code Generation	ZXing / QRGen Library
Build Tool	Maven / Gradle
Server	Embedded Tomcat
API Format	REST (JSON)

---

5. Core Backend Modules :

5.1 Authentication Module

Handles secure login, registration, and token generation.

- **JWT (JSON Web Token):**
  Each user receives a signed token upon login, which must be provided for all further API requests.
- **Spring Security Integration:**
  Enforces authentication and authorization filters.

Login Flow:

1. User submits credentials via /api/auth/login.
2. Credentials are verified with stored hashed passwords.
3. On success, a JWT token is issued and sent in the response header.
4. All subsequent requests include this token for verification.

Login Endpoint Example:

```
@PostMapping("/login")
public ResponseEntity<?> loginUser(@RequestBody LoginRequest request) {
   Authentication authentication = authenticationManager.authenticate(
      new UsernamePasswordAuthenticationToken(request.getEmail(),
request.getPassword())
   );
   SecurityContextHolder.getContext().setAuthentication(authentication);
   String jwt = jwtUtils.generateJwtToken(authentication);
   return ResponseEntity.ok(new JwtResponse(jwt));
}
```

5.2 User Management Module :

Provides CRUD operations for users.

Endpoints:

Method	Endpoint	Description	Access
POST	/api/users/register	Register a new user	Public

Method	Endpoint	Description	Access
GET	/api/users	Get all users	Admin
DELETE	/api/users/{id}	Delete user by ID	Admin

Sample Controller Code:

```
@PostMapping("/register")
public ResponseEntity<?> register(@RequestBody User user) {
   if (userRepository.findByEmail(user.getEmail()).isPresent()) {
      return ResponseEntity.badRequest().body("Email already exists");
   }
   user.setPassword(passwordEncoder.encode(user.getPassword()));
   userRepository.save(user);
   return ResponseEntity.ok("User registered successfully");
}
```

5.3 QR Code Generation Module:

This module generates **unique QR codes** for each attendance session using the **ZXing** or **QRGen** library.

Flow:

1. Admin creates a new session through /api/qr/create.
2. A unique session ID is generated.
3. A QR code is generated encoding session metadata (session ID, expiry time).
4. The QR image is stored as a Base64 string in MongoDB.
5. The generated QR is displayed on the frontend for students to scan.

QR Generation Example (using ZXing):

```
public String generateQRCode(String text) {
```

```
    try {
        BitMatrix matrix = new MultiFormatWriter().encode(text,
BarcodeFormat.QR_CODE, 300, 300);
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        MatrixToImageWriter.writeToStream(matrix, "PNG", outputStream);
        return Base64.getEncoder().encodeToString(outputStream.toByteArray());
    } catch (Exception e) {
        throw new RuntimeException("QR generation failed", e);
    }
}
```

QR Session Creation Endpoint:

```
@PostMapping("/create")
@PreAuthorize("hasRole('ADMIN')")
public ResponseEntity<?> createQRSession(@RequestBody QRSession
session) {
    session.setValidUntil(LocalDateTime.now().plusMinutes(10));
    String qrCodeData = generateQRCode(session.getId());
    session.setQrCode(qrCodeData);
    qrSessionRepository.save(session);
    return ResponseEntity.ok("QR session created successfully");
}
```

5.4 Attendance Logging Module :

Handles the process when a student scans a QR code and the system records attendance.

Attendance Marking Flow:

1. Student scans QR and sends encoded data to /api/attendance/mark.

2. Backend decodes QR content to extract sessionId.
3. System verifies:
    - QR validity (validUntil not expired).
    - Whether student already marked attendance for that session.
4. On successful validation, attendance is logged with date and time.

Attendance Endpoint Example:

```
@PostMapping("/mark")
@PreAuthorize("hasRole('STUDENT')")
public ResponseEntity<?> markAttendance(@RequestBody AttendanceRequest request) {
   Optional<QRSession> session = qrSessionRepository.findById(request.getQrSessionId());
   if (session.isEmpty() || session.get().getValidUntil().isBefore(LocalDateTime.now())) {
      return ResponseEntity.badRequest().body("QR Code expired or invalid");
   }
   if (attendanceRepository.findByUserIdAndQrSessionId(request.getUserId(), request.getQrSessionId()).isPresent()) {
      return ResponseEntity.badRequest().body("Attendance already marked");
   }
   AttendanceLog log = new AttendanceLog(request.getUserId(), request.getQrSessionId(), new Date(), LocalTime.now().toString(), "PRESENT");
   attendanceRepository.save(log);
   return ResponseEntity.ok("Attendance marked successfully");
}
```

5.5 Admin Dashboard and Reporting Module :

Allows administrators to:

- View all attendance logs.
- Filter data by **date**, **class**, or **student**.
- Export attendance in **CSV** or **Excel** format.

Sample Endpoint:

```
@GetMapping("/logs")
@PreAuthorize("hasRole('ADMIN')")
public List<AttendanceLog> getLogs(
   @RequestParam(required = false) String date,
   @RequestParam(required = false) String userId) {
   if (userId != null)
      return attendanceRepository.findByUserId(userId);
   else
      return attendanceRepository.findAll();
}
```

---

6. Error Handling and Response Standardization :

To maintain consistency, all responses follow a **standard JSON structure**:

Success Response:

```
{
  "status": "success",
  "message": "Attendance marked successfully",
  "timestamp": "2025-11-12T10:05:21"
}
```

Error Response:

```
{
```

```
  "status": "error",
  "message": "QR Code expired or invalid",
  "timestamp": "2025-11-12T10:06:00"
}
```

Centralized Exception Handler Example:

```
@ControllerAdvice
public class GlobalExceptionHandler {
   @ExceptionHandler(Exception.class)
   public ResponseEntity<?> handleException(Exception e) {
      return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
         .body(Map.of("status", "error", "message", e.getMessage(), "timestamp", new Date()));
   }
}
```

---

7. Security Implementation :

7.1 Authentication Flow

- JWT token issued at login.
- Token verified for every API request via interceptor.
- Unauthorized users receive HTTP 401 response.

7.2 Role-Based Access Control

Role	Access Rights
Admin	Create QR, view logs, manage users
Student	Scan QR, mark attendance, view own history

7.3 Token Validation Filter

```
@Component
```

```
public class JwtAuthFilter extends OncePerRequestFilter {
    @Autowired
    private JwtUtils jwtUtils;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {
        String jwt = parseJwt(request);
        if (jwt != null && jwtUtils.validateJwtToken(jwt)) {
            String username = jwtUtils.getUserNameFromJwtToken(jwt);
            // Set authentication context
        }
        filterChain.doFilter(request, response);
    }
}
```

---

8. Testing and Validation :

Backend Testing Tools:

- **Postman** – API testing
- **JUnit & Mockito** – Unit and integration testing

Key Test Scenarios

Test Case	**Expected Result**	**Status**
User Registration	User successfully added to DB	✅
Duplicate Email	Error “Email already exists”	✅

Test Case	Expected Result	Status
JWT Login	Token issued on correct credentials	✅
Expired QR	Attendance rejected	✅
Duplicate Attendance	Prevents second scan	✅

---

9. Expected Deliverables of Module 4 :

1. Fully functional **Spring Boot backend**.
2. **QR code generation and validation** logic implemented.
3. **JWT-based authentication** with role-based access control.
4. Tested and documented **RESTful APIs** for all modules.
5. Robust **error handling** and standardized JSON responses.

---

10. Future Path (Next Module Overview) :

Module 5 – Frontend Development and User Interface

- Develop the **React.js frontend**.
- Integrate **QR scanner component** for students.
- Build **Admin Dashboard** with filters and reports.
- Connect frontend with backend APIs.

---

11. Conclusion :

This module successfully establishes the **core logic and backend foundation** of the QR-Based Attendance System.
With Spring Boot, JWT security, and MongoDB integration, the backend ensures reliable performance, data security, and scalable API communication.

By completing this phase, the system is now ready for the **Frontend Development and User Interface (Module 5)**, where the user-facing web application will be built and integrated with this backend.