Security Requirements Generated (GEN1+GEN2)

1. Implement role-based access control to ensure only authorized users can modify class schedules or book exam sessions. – Access Management

2. Ensure that users cannot access or modify data outside their assigned roles (e.g., students cannot edit course schedules). – Secure Design

3. Regularly test access control mechanisms to identify and fix potential vulnerabilities. – Security Testing

4. Sanitize all user inputs to prevent SQL injection when querying class schedules or booking exam sessions. – Input Validation

5. Use parameterized queries to handle user inputs in database operations. – Secure Coding

6. Validate and sanitize all inputs in the communication module between students and professors. – Web Application Security - Input Handling

7. Implement multi-factor authentication for professors and course coordinators to secure their accounts. – Authentication

8. Ensure that passwords are hashed using a strong cryptographic algorithm before storage. – Data Protection

9. Regularly audit authentication mechanisms to ensure compliance with security best practices. – Web Application Security - Authentication

10. Implement role-based access control (RBAC) to ensure users can only access functionalities relevant to their role (e.g., students cannot modify class schedules). – Access Management

11. Enforce session timeouts and re-authentication for sensitive actions (e.g., modifying exam bookings or class schedules). – Access Management

12. Log and monitor all access attempts to sensitive functionalities (e.g., account management by technical staff). – Access Management

13. Implement multi-factor authentication (MFA) for all user roles, especially for professors and course coordinators. – Authentication

14. Enforce strong password policies (e.g., minimum length, complexity, and expiration). – Authentication

15. Prevent brute-force attacks by implementing account lockout after a defined number of failed login attempts. – Authentication

16. Encrypt all sensitive data at rest, including user credentials and personal information. – Data Protection

17. Ensure all data transmitted between the client and server is encrypted using TLS 1.2 or higher. – Data Protection

18. Implement data minimization practices to only collect and store necessary user information. – Data Protection

19. Validate and sanitize all user inputs (e.g., form submissions, messages) to prevent XSS attacks. – Input Validation

20. Use parameterized queries or prepared statements to prevent SQL injection in database interactions. – Input Validation

21. Restrict file uploads to specific formats and scan uploaded files for malware. – Input Validation

22. Conduct regular code reviews to identify and fix security vulnerabilities in the codebase. – Secure Coding

23. Use secure coding libraries and frameworks to handle common security tasks (e.g., authentication, encryption). – Secure Coding

24. Avoid hardcoding sensitive information (e.g., API keys, passwords) in the source code. – Secure Coding

25. Design the system with the principle of least privilege, ensuring users have only the necessary permissions. – Secure Design

26. Implement secure default configurations for all components (e.g., databases, servers). – Secure Design

27. Ensure all third-party components are vetted for security before integration. – Secure Design

28. Perform regular penetration testing to identify and remediate security vulnerabilities. – Security Testing

29. Conduct automated vulnerability scanning of the application and infrastructure. – Security Testing

30. Include security testing in the CI/CD pipeline to catch vulnerabilities early in development. – Security Testing

31. Implement CSP (Content Security Policy) to mitigate XSS and data injection attacks. – Web Application Security

32. Use secure cookies with HttpOnly and Secure flags to protect session tokens. – Web Application Security

33. Ensure all API endpoints enforce proper authentication and authorization checks. – Web Application Security

34. Implement rate limiting on authentication endpoints to prevent brute-force attacks. – Web Application Security - Authentication

35. Use secure password recovery mechanisms (e.g., time-limited tokens, no security questions). – Web Application Security - Authentication

36. Ensure session tokens are invalidated upon logout or after a period of inactivity. – Web Application Security - Authentication

37. Sanitize and validate all user-generated content (e.g., messages, notes) to prevent XSS. – Web Application Security - Input Handling

38. Enforce strict input validation on all form fields (e.g., class schedules, exam bookings). – Web Application Security - Input Handling

39. Use whitelisting for input fields to only allow expected characters and formats. – Web Application Security - Input Handling

Total: GEN1=9, GEN2=30, Total=39

Counting per category (total = gen1 + gen2):

- Access Management: total=4 (gen1=1, gen2=3)
- Authentication: total=4 (gen1=1, gen2=3)
- Data Protection: total=4 (gen1=1, gen2=3)
- Input Validation: total=4 (gen1=1, gen2=3)
- Secure Coding: total=4 (gen1=1, gen2=3)
- Secure Design: total=4 (gen1=1, gen2=3)
- Security Testing: total=4 (gen1=1, gen2=3)
- Web Application Security: total=3 (gen1=0, gen2=3)
- Web Application Security - Authentication: total=4 (gen1=1, gen2=3)
- Web Application Security - Input Handling: total=4 (gen1=1, gen2=3)