Team Quaternary:

ST10019838 - Damian Dare

ST10019972 - Guillaume Swanevelder

ST10091991 - Christiaan Versfeld

ST10158660 - Ruan Zwarts

Module: XBCAD7319

Lecturer: Ms. Mmaphuti Matau

Documentation: Security

# Table of Contents

[Table of Contents 2](#_Toc182735375)

[1. INTRODUCTION 3](#_Toc182735376)

[2. SECURITY ARCHITECTURE 4](#_Toc182735377)

[2.1 Frontend (React) 5](#_Toc182735378)

[2.2 Backend (Next.js) 5](#_Toc182735379)

[2.3 Database (Supabase) 6](#_Toc182735380)

[2.4 Third-Party Services 6](#_Toc182735381)

[2.5 Data Flow and Interaction 7](#_Toc182735382)

[3. GENERAL SECURITY CONSIDERATIONS 8](#_Toc182735383)

[5. POTENTIAL THREAT ACTORS AND VECTORS 9](#_Toc182735384)

[5.1 Potential Threat Actors 9](#_Toc182735385)

[5.2 POTENTIAL THREAT VECTORS 10](#_Toc182735386)

[6. MITIGATIONS FOR THREATS 11](#_Toc182735387)

[6.1 Economy of Mechanism: Mitigation strategies prioritize simplicity and effectiveness 11](#_Toc182735388)

[6.2 Balancing Security with Usability 11](#_Toc182735389)

[6.3 Security for Data Access 11](#_Toc182735390)

[7. COMPLETE MEDIATION 12](#_Toc182735391)

[8. CONCLUSION 13](#_Toc182735392)

[REFERENCE LIST 14](#_Toc182735393)

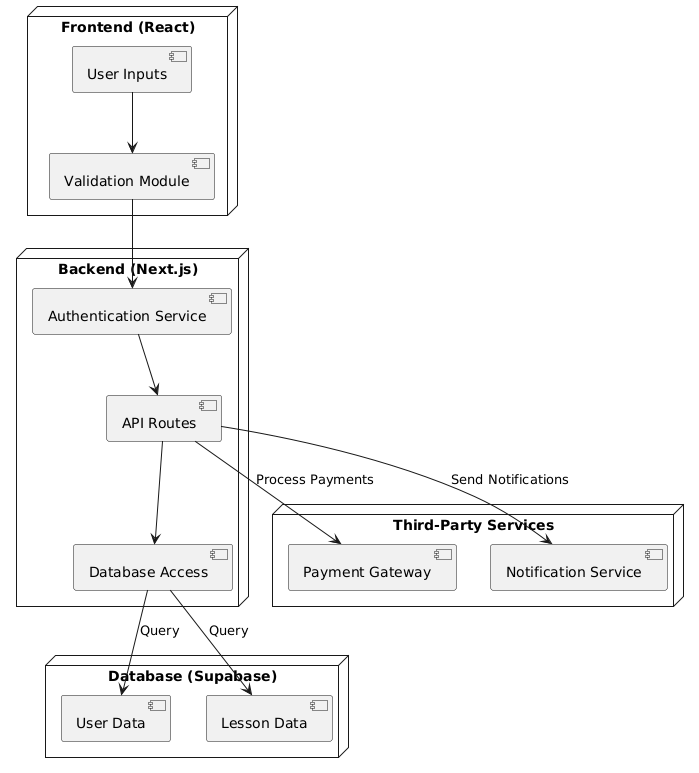
# INTRODUCTION

A crucial component of software development is security, especially when handling private user data, as in the lesson management program created for Crystal Ridge. To safeguard user data, guarantee system dependability, and preserve user and client organisation trust, system security is critical.

The possible security concerns, threat actors and vectors, mitigation strategies used, and security concepts included into the application are all thoroughly examined in this part. Along with a rational assessment of full mediation and safe data access, the conversation also emphasises the need to strike a balance between security and usability.

# SECURITY ARCHITECTURE

The provided diagram illustrates the key components of the application architecture, focusing on security mechanisms and data flow across the system. This architecture ensures secure data handling, robust user authentication, and reliable integration with external services. Each component has a specific role in maintaining the integrity, confidentiality, and availability of the system.



(PlantText, 2024)

## Frontend (React)

The Frontend represents the user-facing interface, built using React, and is responsible for collecting user inputs and ensuring data is validated before sending requests to the backend. It incorporates essential security measures to safeguard user interactions:

**User Inputs:**

* Collects information from users, such as login credentials, booking requests, or payment details.
* Ensures secure communication with the backend via HTTPS.

**Validation Module:**

* Implements client-side validation to prevent invalid or potentially malicious data from reaching the backend.
* Filters input to block injection attacks such as Cross-Site Scripting (XSS).
* Provides user-friendly error messages to enhance usability while maintaining secure defaults.

## Backend (Next.js)

The Next.js backend acts as the core processing layer, bridging the frontend with the database and third-party services. It handles business logic, data processing, and API integrations.

**Authentication Service:**

* Utilizes Supabase’s built-in authentication features integrated into the Next.js backend.
* Supports email/password login and third-party authentication providers (e.g., Google, Facebook).
* Manages session tokens to ensure secure communication between the frontend and backend.

**API Routes:**

* Next.js API routes handle RESTful requests and server-side processing.
* Implements middleware to validate API requests, ensuring only authenticated users can access protected routes.

**Database Access:**

* Manages secure interactions with the Supabase database for CRUD operations.
* Enforces Role-Based Access Control (RBAC) to limit database operations based on user roles (Admin, Coach, Client).
* Utilizes Next.js server-side logic to sanitize and validate all database queries, mitigating risks like SQL injection.

## Database (Supabase)

The **Supabase database** is a NOSQL instance that stores all critical application data.

**User Data**:

* Contains user profiles, roles, and authentication tokens.
* Implements row-level security (RLS) to ensure that users can only access their own data.
* Uses encryption at rest and in transit to protect sensitive information.

**Lesson Data**:

* Stores lesson schedules, booking information, and associated metadata.
* Uses structured relationships to link lessons with users and other relevant entities.
* Periodic backups are configured to ensure data recovery in case of failures.

## Third-Party Services

The system integrates securely with third-party services to handle specific functionalities:

**Payment Gateway**:

* Processes payments securely without storing sensitive payment information in the system.
* Next.js backend acts as an intermediary, verifying payment responses before updating booking statuses.

**Notification Service**:

Sends real-time alerts or reminders to users about lesson bookings, updates, or cancellations.

Utilizes secure API keys stored in environment variables.

## Data Flow and Interaction

This section describes the interactions between components:

1. **Frontend (React)**:
   * User inputs are validated client-side and sent to the Next.js backend via HTTPS.
2. **Backend (Next.js)**:
   * API routes process incoming requests, authenticate users, and validate data before accessing the database.
   * Server-side logic ensures only authorized actions are executed.
3. **Database (Supabase)**:
   * Data is accessed or updated based on requests from the backend, adhering to row-level security policies.
4. **Third-Party Services**:
   * External services like payment processors or notification APIs are called from the backend.

# GENERAL SECURITY CONSIDERATIONS

1. Authentication and Authorization:

* Authentication tokens are used for session management.
* Role-based access control ensures data and functionalities are accessed appropriately.

1. Data Encryption:

* All sensitive data is encrypted in transit using HTTPS and at rest using Supabase's built-in encryption.

1. Input Validation:

* Input is validated both at the client (frontend) and server (backend) to prevent injection attacks.

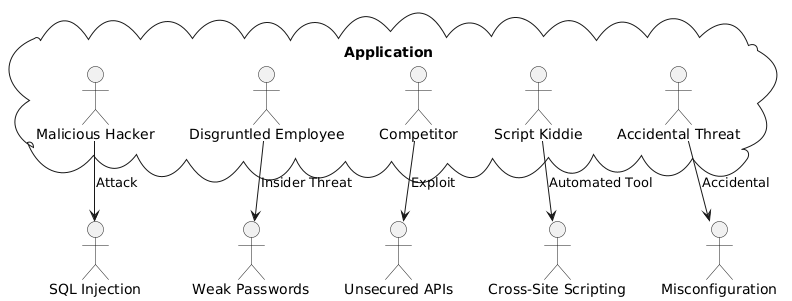
1. API Security:

* Third-party APIs are integrated securely with the use of environment-stored keys and tokens.

1. Complete Mediation:

* Every request is authenticated, authorized, and validated to ensure complete mediation at all layers.

# POTENTIAL THREAT ACTORS AND VECTORS



(PlantText, 2024)

## Potential Threat Actors

The following are possible threat actors for this application:

1. Malicious Hackers:

**Aim**: sensitive user data being accessed without authorisation or system functionality being interrupted.

**Methods:** taking advantage of flaws in login credentials, such as SQL injection or brute force attacks.

1. Disgruntled Employees:

**Aim:** Internal sabotage or unauthorized data access.

**Methods:** Leveraging insider knowledge to bypass access controls or manipulate data.

1. Competitors:

**Aim:** To gain insight into the business operations or disrupt client services.

**Methods:** Social engineering attacks or denial-of-service (DoS) attacks.

1. Script Kiddies:

**Aim:** Random exploitation for notoriety or learning purposes.

**Methods:** Automated tools to exploit known vulnerabilities.

1. Accidental Threats:

**Aim:** No malicious intent but can cause data breaches or downtime.

**Methods:** Mistakes such as sharing passwords or misconfiguring access permissions.

## POTENTIAL THREAT VECTORS

The subsequent are recorded potential vectors by which the application may be compromised:

1. **SQL Injection**:

* Exploitation of input fields to manipulate database queries.

1. **Cross-Site Scripting (XSS)**:

* Injection of malicious scripts into the application to exploit user sessions.

1. **Unsecured APIs**:

* APIs without proper authentication or encryption may expose sensitive endpoints.

1. **Weak Passwords**:

* Poor password policies can result in compromised user accounts.

1. **Third-Party Service Vulnerabilities**:

* Dependencies on external services like Supabase or payment gateways can introduce risks.

1. **Social Engineering**:

* Manipulating users to disclose credentials or other sensitive information.

# MITIGATIONS FOR THREATS

## Economy of Mechanism: Mitigation strategies prioritize simplicity and effectiveness

**Authentication and Role-Based Access Control (RBAC)**:

Ensures that users can only access features based on their roles (Admin, Coach, Client).

Mitigation: Multi-factor authentication (MFA) is recommended for Admins.

**Data Validation and Sanitization**:

Protects against SQL injection and XSS by validating and escaping user inputs.

**Mitigation:** Framework-provided libraries for sanitization in React and Supabase.

## Balancing Security with Usability

Security and usability are harmonised to optimise user experience while preserving protection

**Secure Defaults**:

Features such as password complexity requirements, session timeouts and secure cookies are enabled without user intervention.

**Clear Feedback**:

Error messages are informative but avoid exposing system details, preventing reconnaissance by attackers.

**Minimal Data Collection**:

Only essential user information is collected (name, email, phone number).

## Security for Data Access

**Access Tokens**:

Secure tokens manage user sessions, preventing unauthorized access.

**Database Layer Security**:

Supabase uses Row-Level Security (RLS) to enforce access rules at the database level.

**Encrypted Communication**:

All data transfers use HTTPS to protect data in transit.

# COMPLETE MEDIATION

Complete mediation ensures that every data access request is authenticated, authorized, and validated:

**Consistent Authentication:**

Every API request is validated against session tokens or API keys.

**Authorization Rules:**

Backend endpoints verify user roles before processing sensitive operations like booking or editing lessons.

**Audit Logging:**

All critical actions are logged for review and anomaly detection.

# CONCLUSION

The application's security policy is to reduce potential threats while preserving usability. By detecting threat actors and vectors, implementing cost-effective mitigations and comprehensive mediation, and utilising cloud-native security capabilities, the application attains a strong security posture. Ongoing monitoring, consistent upgrades, and security assessments will guarantee the application’s protection against emerging threats.

# REFERENCE LIST

PlantText, 2024. *PlantUML.* [Online]   
Available at: https://www.planttext.com/  
[Accessed 24 10 2024].