**1. High-Level Design (HLD)**

**Architecture Overview**

1. **Frontend:**  
   • Developed in **React** with components designed and prototyped using **Figma**.
2. **Backend:**  
   • Built using **Spring Boot** that exposes REST APIs for every module.
3. **Database:**  
   • Uses **MySQL** for persistent storage..
4. **Interaction Flow:**  
   • The React UI interacts with Spring Boot endpoints via RESTful APIs.  
   • User actions (e.g., signing up, registering for an event) trigger API calls that perform validations, database transactions, and session management.  
   • Security is maintained through session tokens or JWT, while all sensitive transactions (e.g., payments) are encrypted.

**2. Low-Level Design (LLD)**

**Key Entities & Database Schema (MySQL)**

* **Users Table:**
  + **Columns:** *user\_id, name, email, phone, password\_hash, created\_at, updated\_at*
  + **Purpose:** Store credentials and profile information.
* **Events Table:**
  + **Columns:** *event\_id, title, description, date, image\_url, category*
  + **Purpose:** Maintain details of upcoming events, tech fest, and special events.
* **Clubs Table:**
  + **Columns:** *club\_id, club\_name, description, activities, contact\_info*
  + **Purpose:** Hold club information and details for users to view and join.
* **Registrations Table:**
  + **Columns:** *registration\_id, user\_id, event\_id, registration\_date, status*
  + **Purpose:** Track user registrations for events.
* **Payments Table:**
  + **Columns:** payment\_id, registration\_id, amount, payment\_method, transaction\_status, payment\_date
  + **Purpose:** Record details of event payments.
* **Feedback Table:**
  + **Columns:** *feedback\_id, event\_id, user\_id (nullable for anonymous), rating, comments, submitted\_at*
  + **Purpose:** Collect post-event feedback.
* **FAQs Table:**
  + **Columns:** *faq\_id, question, answer, category*
  + **Purpose:** Store FAQ content with categorization for easier navigation.
* **Notifications Table:**
  + **Columns:** *notification\_id, message, event\_id (if applicable), created\_at, dismissed\_by (list of user\_ids)*
  + **Purpose:** Manage alert messages shown on the home page.

**API Endpoints (Spring Boot)**

* **User Authentication:**
  + **POST /api/auth/signup** – Registers a new user.
  + **POST /api/auth/login** – Authenticates a user and starts a session.
  + **POST /api/auth/forgot-password** – Initiates password recovery.
* **Event Discovery:**
  + **GET /api/events** – Returns a list of upcoming events (sorted by date, with filtering/search options).
* **Contact Support:**
  + **POST /api/support/contact** – Accepts and processes support inquiries.
* **Manage Account:**
  + **GET /api/user/profile** – Retrieves current user profile.
  + **PUT /api/user/profile** – Updates profile details.
  + **PUT /api/user/change-password** – Changes the user’s password.
* **Club Details:**
  + **GET /api/clubs** – Lists all clubs.
  + **GET /api/clubs/{clubId}** – Provides detailed information about a specific club.
* **Event Registration:**
  + **POST /api/events/{eventId}/register** – Registers a user for an event.
  + **PUT /api/registrations/{registrationId}** – Modifies or cancels an event registration.
* **Secure Payments:**
  + **POST /api/payments/checkout** – Processes payments for events and returns a receipt.
* **Event Feedback:**
  + **POST /api/events/{eventId}/feedback** – Submits user feedback for an event.
* **FAQs & Notifications:**
  + **GET /api/faqs** – Fetches categorized FAQs.
  + **GET /api/notifications** – Retrieves current event notification alerts.

**Sample Spring Boot Code (LLD Code Snippet)**

Below is an example of a simplified **UserController** using Spring Boot:

*@RestController*

*@RequestMapping("/api/auth")*

*public class UserController{*

*@Autowired*

*private UserService userService;*

*// Endpoint for user registration*

*@PostMapping("/signup")*

*public ResponseEntity<?> registerUser(@RequestBody UserDto userDto){*

*try{*

*User newUser=userService.registerUser(userDto);*

*return ResponseEntity.ok(newUser);*

*}catch (Exception e) {*

*return ResponseEntity.badRequest().body("Registration failed: " + e.getMessage());*

*}*

*}*

// Endpoint for user login

@PostMapping("/login")

public ResponseEntity<?> loginUser(@RequestBody LoginDto loginDto) {

try {

String token = userService.authenticateUser(loginDto);

return ResponseEntity.ok(new AuthResponse(token));

} catch (Exception e) {

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid credentials");

}

}

// Endpoint for password recovery

@PostMapping("/forgot-password")

public ResponseEntity<?> forgotPassword(@RequestBody Map<String, String> payload) {

String email = payload.get("email");

boolean status = userService.initiatePasswordRecovery(email);

return status ? ResponseEntity.ok("Recovery email sent")

: ResponseEntity.badRequest().body("Error in sending recovery email");

}

}

**Sample React Component (Login Form)**

Below is a simplified example of a React login component:

*import React, { useState } from 'react';*

*import axios from 'axios';*

*const LoginForm = ({ onLoginSuccess }) => {*

*const [email, setEmail] = useState('');*

*const [password, setPassword] = useState('');*

*const [error, setError] = useState('');*

*const handleLogin = async (e) => {*

*e.preventDefault();*

*try {*

*const response = await axios.post('/api/auth/login', { email, password });*

*onLoginSuccess(response.data.token);*

*} catch (err) {*

*setError('Invalid credentials. Please try again.');*

*}*

*};*

*return (*

*<form onSubmit={handleLogin}>*

*<div>*

*<label>Email:</label>*

*<input*

*type="email"*

*value={email}*

*onChange={(e) => setEmail(e.target.value)}*

*required*

*/>*

*</div>*

*<div>*

*<label>Password:</label>*

*<input*

*type="password"*

*value={password}*

*onChange={(e) => setPassword(e.target.value)}*

*required*

*/>*

*</div>*

*{error && <p className="error">{error}</p>}*

*<button type="submit">Login</button>*

*<p><a href="/forgot-password">Forgot Password?</a></p>*

*</form>*

*);*

*};*

*export default LoginForm.*

**4. Putting It All Together**

**Design Documentation Summary**

* **Frontend:**  
  The React app (designed in Figma) will contain dedicated components for each feature (authentication, event discovery, etc.). Each component will interact with the Spring Boot backend via REST APIs.
* **Backend:**  
  Spring Boot services are responsible for processing user requests, validating inputs, performing CRUD operations on MySQL, and ensuring secure transaction handling (e.g., through JWT or session tokens).
* **Database:**  
  The MySQL schema is designed with normalized tables for Users, Events, Clubs, Registrations, Payments, Feedback, FAQs, and Notifications. Foreign key constraints and proper indexing will ensure data integrity and efficient queries.

**Sample LLD Code (Spring Boot & React)**

The provided code snippets above illustrate a basic controller in Spring Boot for user authentication and a React login form that calls the backend API.

You can now take the Google Sheet structure and LLD code examples as a foundation. The Google Sheet can be shared with your team for collaboration on the detailed design, while the code samples provide a starting point for implementation.

This comprehensive approach ensures that both the UI/UX (React/Figma) and backend (Spring Boot/MySQL) are well-coordinated for secure and efficient functionality across all user stories.