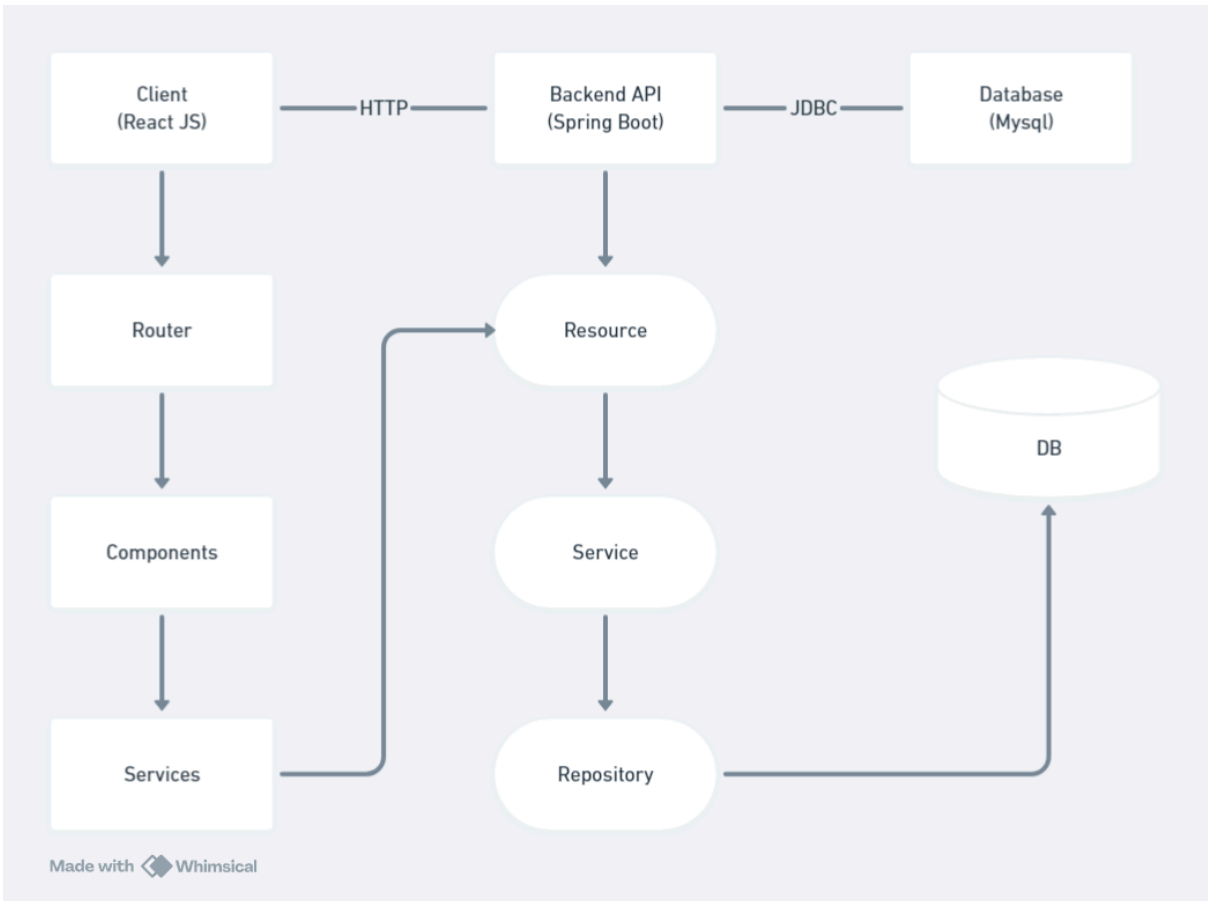
1. High-Level Design

The high-level design outlines the main components and their interactions within the application.



- **Client (React JS)**:
  - Provides the user interface for interacting with the application.
  - Includes components for user authentication (login, register) and managing notes.
- **Backend API (Spring Boot)**:
  - Acts as the server-side application handling HTTP requests.
  - Implements RESTful APIs for user registration, authentication, and note management.
  - Manages user sessions and ensures data integrity.
- **Database (MySQL)**:
  - Stores user information (username, hashed password) and notes data.
  - Utilizes JDBC for database connectivity to interact with Spring Boot.

2. Web App UI (Frontend with React JS)

The frontend design focuses on providing a responsive and intuitive user interface for the note-taking application.

UI Components Overview

- **Login Page**:
    - Allows users to enter their credentials (username, password) to authenticate.
    - Sends login requests to the backend API (/api/login endpoint).
- **Registration Page**:
    - Allows new users to register by providing a username and password.
    - Sends registration requests to the backend API (/api/register endpoint).
- **Notes Page**:
    - Displays a list of notes retrieved from the backend API (/api/notes endpoint).
    - Allows users to add new notes and delete existing notes.
    - Required fields in notes: **title** and **content**.

3. Data Model

The data model defines how notes are structured and stored in the database.

Database Schema (MySQL)

- **Note Table**:

```
CREATE TABLE notes (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id BIGINT,
    title TEXT,
    content TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

4. RESTful API (Backend with Spring Boot)

The backend implements RESTful APIs to manage user authentication and notes operations.

API Endpoints and Security

Notes Management Endpoint (/api/notes)

- **GET**:

  Description: Retrieves all notes for the authenticated user.
  Authorization: Bearer token in the request header.
  Response: 200 OK with an array of notes.

```
[
    {
        "title": "Feefo",
        "content": "Test for interview"
    }
]
```

- **POST**:

  Description: Creates a new note for the authenticated user.
  Authorization: Bearer token in the request header.
  Request Body:

```
{
    "title": "New note",
    "content": "Test create"
}
```

  Response: 200 OK with the created note object.

- **DELETE** /api/notes/{noteId}:

  Description: Deletes a specific note belonging to the authenticated user.
  Authorization: Bearer token in the request header.
  Response:
  - 200 OK on successful deletion.
  - 404 Not Found if the note with the provided noteId doesn't exist or doesn't belong to the user.

5. Web Server

The web server, based on Spring Boot, serves as the backend application responsible for processing HTTP requests from the frontend (built with React JS) and interacting with the MySQL database to manage user notes.

Package and Class Structure

In a Spring Boot project, organizing classes into packages helps maintain a structured and modular application. Below is a simplified structure of the packages and classes involved:

- Controller: NoteController.java
- Service: NoteService.java
- Repository: NoteRepository.java
- Entity: Note.java, User.java (assuming there is a User entity to represent users)

Business logic (Validation before saving a new note)

```
@Service
public class NoteService {

   @Autowired
   private NoteRepository noteRepository;

   public List<Note> getAllNotesByUserId(Long userId) {
       return noteRepository.findByUserId(userId);
   }

   public Note createOrUpdateNote(Long userId, NoteDTO noteDTO) {
       // Check if a note with the same title already exists for the user
       if (noteRepository.existsByUserIdAndTitle(userId,
noteDTO.getTitle())) {
           throw new IllegalArgumentException("Note with this title already
exists");
       }

       // Proceed with creating or updating the note
       Note note = new Note();
       note.setUser(new User(userId)); // Assuming User constructor with ID
       note.setTitle(noteDTO.getTitle());
       note.setContent(noteDTO.getContent());

       return noteRepository.save(note);
   }

   public void deleteNoteById(Long noteId, Long userId) {
       // Implement deletion logic
   }
}
```

```
@Repository
public interface NoteRepository extends JpaRepository<Note, Long> {
   boolean existsByUserIdAndTitle(Long userId, String title);
}
```

```java
@Entity
@Table(name = "notes", uniqueConstraints = {
        @UniqueConstraint(columnNames = {"user_id", "title"})
})
public class Note {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   @ManyToOne(fetch = FetchType.LAZY)
   @JoinColumn(name = "user_id")
   private User user;

   @Column(columnDefinition = "TEXT")
   private String title;

   @Column(columnDefinition = "TEXT")
   private String content;

   @CreationTimestamp
   @Column(name = "created_at", updatable = false)
   private LocalDateTime createdAt;

   @UpdateTimestamp
   @Column(name = "updated_at")
   private LocalDateTime updatedAt;

   // Constructors, getters, and setters
}
```