MoveInSync SDE Case Study

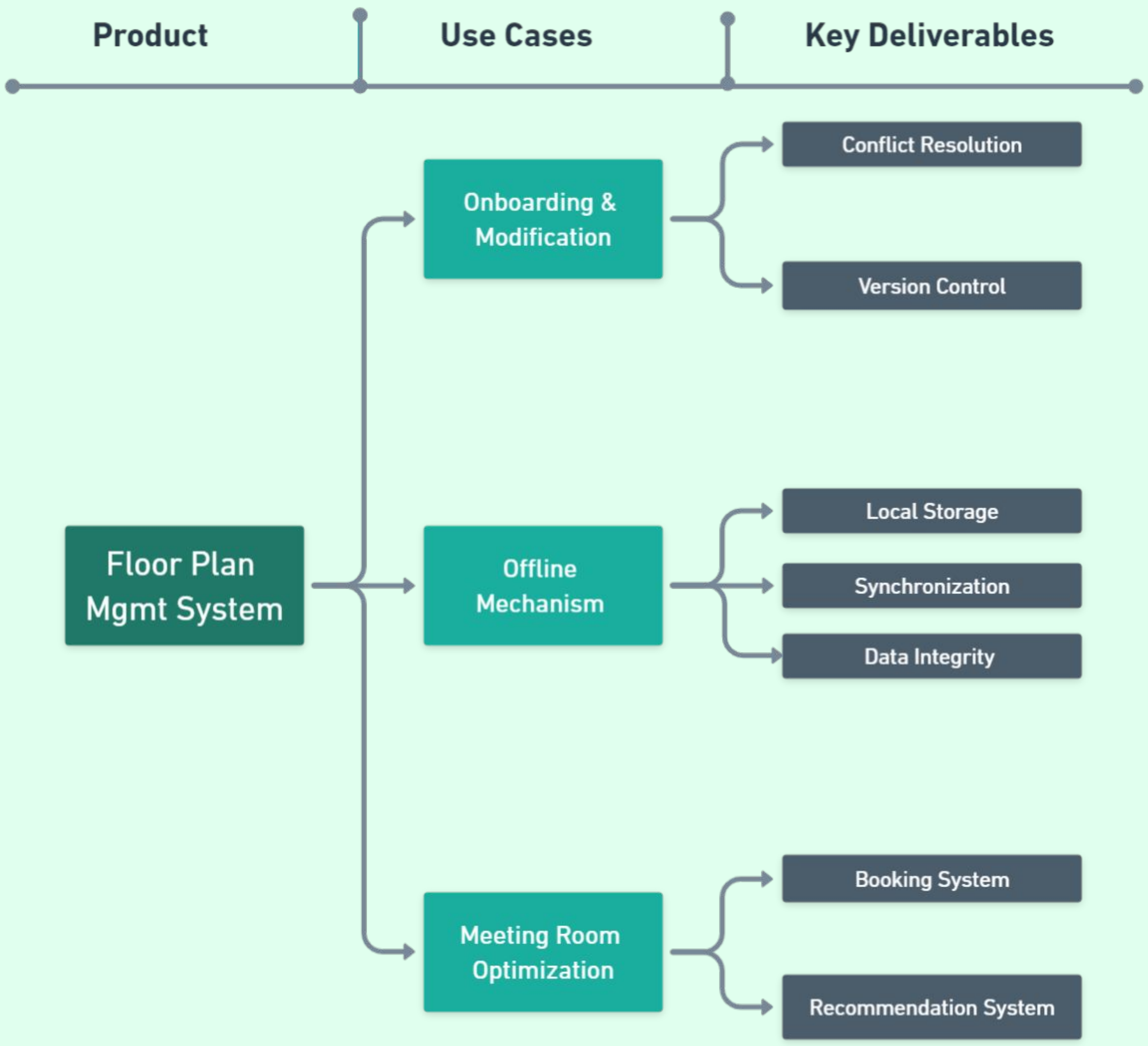
Floor Plan Management System

Kush Tyagi
2019B4A70689G

Introduction

The Floor Plan Management System presented herein is a **robust** solution catering to the diverse needs of administrators and users alike. The system seamlessly manages floor plans, meeting room bookings, and addresses challenges associated with conflicts during concurrent updates and offline scenarios.

The Intelligent floor plan management system offers various functionalities for the Admin and User.



Chapter 1 - Floor Plan Mgmt for Administrators

Conflict Resolution Mechanism

- **Class : 'ConflictResolver'**
  - **Method: 'resolveConflict' :** Prioritizes updates intelligently based on factors such as priority, timestamp, and user roles. Admins have precedence, and conflicts are resolved considering the priority and timestamp.

```
public class ConflictResolver {
    // Simulate conflict resolution based on priority, timestamp, or user roles
    public void resolveConflict(final FloorPlan localPlan, final FloorPlan serverPlan, final Admin admin)
{
        if (admin != null && admin.authenticate("admin_password") && admin.getRole() == UserRole.ADMIN) {
            log.info("Admin resolving conflict. Admin's version takes priority.");
            serverPlan.uploadPlan();
        } else {
            // Check priority first
            int priorityComparison = Integer.compare(localPlan.getPriority(), serverPlan.getPriority());

            if (priorityComparison > 0) {
                log.info("Conflict resolved. Uploading local version based on priority..");
                serverPlan.uploadPlan();
            } else if (priorityComparison < 0) {
                log.info("Conflict resolved. Merging server version based on priority..");
                localPlan.uploadPlan();
            } else {
                // If priorities are the same, check timestamp
                int timestampComparison =
localPlan.getLastModified().compareTo(serverPlan.getLastModified());

                if (timestampComparison > 0) {
                    log.info("Conflict resolved. Uploading local version based on timestamp..");
                    serverPlan.uploadPlan();
                } else if (timestampComparison < 0) {
                    log.info("Conflict resolved. Merging server version based on timestamp..");
                    localPlan.uploadPlan();
                } else {
                    log.error("Conflict detected. Additional resolution needed.");
                    throw new IllegalStateException("Conflict detected. Additional resolution needed.");
                }
            }
        }
    }

}
```

Version Control System

- **Class : ‘FloorPlan’**
  - **Attributes**
    - ‘Version’ represents the iteration of changes.
    - ‘lastModified’ : keeps track of the last modification timestamp.
  - **Method : ‘uploadPlan()’ :** Increments the version and updates the last modification timestamp upon uploading the floor plan.

```java
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class FloorPlan {
    private final int floorPlanId;
    private final String palanName;
    private final int version;
    private final List<Room> rooms;
    private final Date createdDate;
    private final Admin createdBy;
    private final DataCache dataCache;
    private final int priority;
    private String description;
    private List<String> tags;
    private Date lastModified;


    public FloorPlan(int floorPlanId, String planName, int version, Date lastModified,
                     List<Room> rooms, Date createdDate, Admin createdBy,
                     String description, List<String> tags) {
        this.floorPlanId = floorPlanId;
        this.planName = planName;
        this.version = version;
        this.lastModified = lastModified;
        this.rooms = new ArrayList<>(rooms);
        this.createdDate = new Date(createdDate.getTime());
        this.createdBy = createdBy;
        this.description = description;
        this.tags = new ArrayList<>(tags);
        this.dataCache= new DataCache();
    }
```

```
// Additional methods
    public void addRoom(Room room) {
        rooms.add(room);
        dataCache.addToCache("Room_" + room.getId(), room);
    }

    public void removeRoom(Room room) {
        rooms.remove(room);
         dataCache.removeFromCache("Room_" + room.getId());
    }

    public Room getRoomById(final int roomId) {
        final Room cachedRoom = (Room) dataCache.getFromCache("Room_" + roomId);
        if (cachedRoom != null) {
            log.info("Room found in cache!");
            return cachedRoom;
        }
        else{
        for (Room room : rooms) {
            if (room.getId() == roomId) {
                return room;
            }
        }
        }
        return null; // room not found
    }


    public void uploadPlan() {
        this.version++;
        this.lastModified = new Date();
        log.info("Floor plan uploaded. New version {}", version);

    }
```

- **Admin Class**
    - **Enum Roles:** Admin class contains enum to ‘REGULAR_USER’ and ‘ADMIN’ depending on the role of the user which gets set in the constructor upon object creation.
    - **Attributes :** ‘username’ , ‘hashedPassword’ , ‘role’ ,’conflictResolver’.
    - **Methods:**
        - **authenticate() :** Verifies whether the password after hashing matches the stored password.
        - **hashPassword() :** Hashes the password using ‘SHA-256’ hashing algorithm.
        - **resolveConflict() :** resolved conflict by taking floor plans and comparing them based on priority fields and timestamps.

```
 private String hashPassword(String password) {
        try {
            final MessageDigest md = MessageDigest.getInstance("SHA-256");
            final byte[] hashedBytes = md.digest(password.getBytes());
            final StringBuilder sb = new StringBuilder();

            for (byte b : hashedBytes) {
                sb.append(String.format("%02x", b));
            }

            return sb.toString();
        } catch (final NoSuchAlgorithmException e) {
            throw new RuntimeException("Error hashing password", e);
        }
    }

    // Simulate resolving conflicts during simultaneous updates
    public void resolveConflict(final FloorPlan localPlan, final FloorPlan serverPlan) {
        if (localPlan == null || serverPlan == null) {
            log.error("floor plan cant be null");
            throw new IllegalArgumentException("Floor plans cannot be null.");
        }
```

Chapter 2 - Offline Mechanism For Admins

Local Storage System

- **Class : ‘OfflineStorage’**
  - **Methods**
    - **1) savePlan() : Saves the floor plans locally.**
    - **2) loadPlans() : Loads locally stored plans.**
    - **3) clearStorage() : Clears local storage.**

```
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

class OfflineStorage {
    private final List<FloorPlan> localPlans;

    public OfflineStorage() {
        this.localPlans = new ArrayList<>();
    }

    public void savePlan(final FloorPlan floorPlan) {
        // Save the floor plan locally
        localPlans.add(floorPlan);
        log.info("Floor plan saved locally {} ",floorPlan.getPlanName());
    }

    public List<FloorPlan> loadPlans() {
        // Load locally stored floor plans
        log.info("Loading locally stored floor plans...");
        return new ArrayList<>(localPlans);
    }

    public void clearStorage() {
        // Clear local storage
        localPlans.clear();
        log.info("Local storage cleared.");
    }
}
```

Synchronization

- **Class : ‘ServerSyncer’**
  - **Methods**
    - **synchronizeWithServer() : Checks internet/server connection.Loads local plans, updates the server, and clears local storage after successful synchronization. It throws an exception if there is no internet connectivity.**

```
public class ServerSyncer {
    private final OfflineStorage offlineStorage;

    public ServerSyncer(final OfflineStorage offlineStorage) {
        this.offlineStorage = offlineStorage;
    }

    public void synchronizeWithServer() {
        // Assume logic to check internet/server connection
       final  boolean isConnected = checkInternetConnection();

        if (isConnected) {
            final List<FloorPlan> localPlans = offlineStorage.loadPlans();

            for (final FloorPlan floorPlan : localPlans) {
                updateFloorPlan(floorPlan);
                log.info("Synchronizing with server: {} ", floorPlan.getPlanName());
                }

            // Clear local storage after successful synchronization
            offlineStorage.clearStorage();
        } else {
           log.error("No internet connection. Synchronization postponed.");
            throw new IllegalStateException("No internet connection. Synchronization postponed.");
        }
    }
```

```
    private boolean checkInternetConnection() {
        // Assume logic to check internet connection
        return true;
    }

    public void updateFloorPlan(FloorPlan floorPlan) {
        // Logic to update the floor plan
        log.info("floor plan updated to {}",floorPlan);
    }
}
```

Chapter 3 - Meeting Room Optimization

Booking System

- **Class : ‘Booking’**
  - **Attributes : ‘bookingId’ ,’startTime’,’endTime’,’participants’,’description’,’createdBy’,’databaseUrl’**
  - **Methods :**
    - **bookRoom() : loads meeting rooms from web data base and iterates through the list to book available room.**
    - **loadMeetingsFromWebDatabase() : Creates HttpUrl connection and gets the meetingRoom information.**

```
public class Booking {

    private final int bookingId;
    private final List<MeetingRoom> meetingRooms;
    private final String startTime;
    private final String endTime;
    private final int participants;
    private final String description;
    private final Admin createdBy;
    private final String databaseUrl;

    public Booking(final int bookingId,final MeetingRoom meetingRoom,final  String startTime,final String
endTime,
    final int participants, String description, Admin createdBy,final String databaseUrl) {
        this.bookingId = bookingId;
        this.startTime = startTime;
        this.endTime = endTime;
        this.participants = participants;
        this.description = description;
        this.createdBy = createdBy;
        this.databaseUrl = databaseUrl;
    }

    public void bookRoom() {
        loadMeetingRoomsFromWebDatabase(databaseUrl);
        for(MeetingRoom meetingRoom : meetingRooms){
        if (meetingRoom.isAvailable(startTime, endTime) &&meetingRoom.hasCapacity(participants)) {

            log.info("Room booked: {}", meetingRoom.getRoomName());
            // Update booking status or perform other actions as needed
        } else {
            log.error("Booking failed. Room not available or capacity exceeded.");
        }
    }
}
public void loadMeetingRoomsFromWebDatabase(String databaseUrl) {
        try {
            URL url = new URL(databaseUrl);
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("GET");

            int responseCode = connection.getResponseCode();
            if (responseCode == HttpURLConnection.HTTP_OK) {
                BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
                String inputLine;
                StringBuilder response = new StringBuilder();

                while ((inputLine = reader.readLine()) != null) {
                    response.append(inputLine);
                }
                reader.close();

                // Parse the response and populate meetingRooms
                parseAndPopulateMeetingRooms(response.toString());
            } else {
                log.error("Error loading meeting rooms from the web database. Response Code: {}",
responseCode);
            }

            connection.disconnect();
        } catch (Exception e) {
            log.error("Exception while loading meeting rooms from the web database", e);
        }
    }
}
```

- **Class ‘MeetingRoom’**
    - **Attributes : ‘roomId’, ‘roomName’ , ‘capacity’ , ‘location’ , ‘bookings’**
    - **Methods -**
        - **isAvailable() : checks room availability.**
        - **hasCapacity() : checks capacity\**
        - **isTimeConflict() : checks time conflict**
        - **addBooking() : adds booking**
        - **getRoomNumber() : gets room number.**