Project Structure Diagram:

```
server
│   │      └──────sql
│   │         │  calendar-schema-production.sql
│   │         │  calendar-schema-test.sql
│   │      └──────http
│   │         │  calendar-schema-production.sql
│   │         │  calendar-schema-test.sql
├──────src
        ├──────main
        │   ├──────java
        │   │   └──────learn

        │   │      └──────calendar
        │   │         │  App.java
        │   │         │
        │   │         ├──────controller
        │   │         │    UserController.java
        │   │         │    EventController.java
        │   │         │    CalendarController.java
        │   │         │    InviteController.java
        │   │         │    ErrorResponse.java
        │   │         │    GlobalExceptionHandler.java
        │   │         │    AttendeeControllerjava
        │   │         │    RoleController.java
        │   │         ├──────data
        │   │         │         ├──────mappers
        │   │         │                    UserMapper.java
        │   │         │                    EventMapper.java
        │   │         │                    CalendarMapper.java
        │   │         │                    AtendeeMapper.java
        │   │         │                    RoleMapper.java
        │   │         │                    InviteMapper.java
        │   │         │    UserJdbcTemplateRepository.java
        │   │         │    UserRepository.java
        │   │         │    CalendarJdbcTemplateRepository.java
        │   │         │    CalendarRepository.java
        │   │         │    EventJdbcTemplateRepository.java
        │   │         │    EventRepository.java
        │   │         │    AttendeeRepository.java
│   │         │   AttendeeJdbcTemplateRepository.java
│   │         │    RoleJdbcTemplateRepository.java
│   │         │    RoleRepository.java
│   │         │   InviteRepository.java
│   │         │   InviteJdbcTemplateRepository.java
│   │         │
│   │         ├───domain
│   │         │   UserService.java
│   │         │   EventService.java
│   │         │   CalendarService.java
│   │         │   AttendeeService.java
│   │         │   InviteService.java
│   │         │   RoleService.java
│   │         │   Response.java
│   │         │   Result.java
│   │         │   ResultType.java
│   │         │
│   │         ├───models
│   │         │    User.java
│   │         │    Calendar.java
│   │         │    Event.java
│   │         │    Attendee.java
│   │         │    Role.java
│   │         │    Attendee.java
│   │         │    RoleType.java
│   │         │    EventType.java
│   │         │    CalType.java
│   │         │    Invite.java
│   │         │
│   │App.java
│   └───resources
└───test
    └───java
        └───learn
            └───calendar
                ├───data
                │    UserJDBCRepositoryTestt.java
                │    CalendarJDBCRepositoryTest.java
                │    EventJDBCRepositoryTest.java
                │    AttendeeJDBCRepositoryTestt.java
```

```
                  │     RoleJDBCRepositoryTest.java
                  │     InviteJDBCRepositoryTest.java

                  │
                  └──────domain
                       UserServiceTest.java
                       CalendarServiceTest.java
                       InviteServiceTest.java
                       EventServiceTest.java
                       AttendeeServiceTest.java
                       RoleServiceTest.java

Client
├─────
     calendar
      ├─────src
      │   ├─────components
      │   │         Navbar.jsx
      │   │         ChatBot.jsx
      │   │         PopupModal.jsx
      │   ├─────pages
      │   ├         Invites
      │   ├         Home
      │   ├         NotFound
      │   ├─────services
      │   │         apiService.jsx
      │   │         openaiService.jsx
      │   ├─────Context
      │   │         CalenderContext.jsx
```

Class Details

App

- `public static void main(String[])` -- instantiate all required classes with valid arguments, dependency injection. run controller

Controllers:

controller.UserController

- `private Service UserService`
- `public List<User> findAll - GET all users`
- `public ResponseEntity<Object> findById - GET user by id`
- `public ResponseEntity<Object> add - Add a user (POST)`
- `public ResponseEntity<Object> edit - Edit a user (PUT)`
- `public ResponseEntity<Object> deleteById - delete a user (DELETE)`

controller.EventController

- `private Service EventService`
- `public List<Event> findAll - GET all Events`
- `public ResponseEntity<Object> findById - GET Event by id`
- `public ResponseEntity<Object> add - Add a Event (POST)`
- `public ResponseEntity<Object> edit - Edit a Event (PUT)`
- `public ResponseEntity<Object> deleteById - delete a Event (DELETE)`

controller.CalendarController

- `private Service CalendarService`
- `public List<Calendar> findAll - GET all Calendars`
- `public ResponseEntity<Object> findById - GET Calendar by id`
- `public ResponseEntity<Object> add - Add a Calendar (POST)`
- `public ResponseEntity<Object> edit - Edit a Calendar (PUT)`
- `public ResponseEntity<Object> deleteById - delete a Calendar (DELETE)`

controller.RoleController

- `private Service RoleService`
- `public List<Role> findAll - GET all Roles`

- `public ResponseEntity<Object> findById - GET Role by id`
- `public ResponseEntity<Object> add - Add a Role (POST)`
- `public ResponseEntity<Object> edit - Edit a Role (PUT)`
- `public ResponseEntity<Object> deleteById - delete a Role (DELETE)`

controller.AttendeeController

- `private Service AttendeeService`
- `public List<Attendee> findAll - GET all Attendees`
- `public ResponseEntity<Object> findById - GET Attendee by id`
- `public ResponseEntity<Object> add - Add a Attendee (POST)`
- `public ResponseEntity<Object> edit - Edit a Attendee (PUT)`
- `public ResponseEntity<Object> deleteById - delete a Attendee (DELETE)`

controller.InviteController

- `private Service InviteService`
- `public List<Invite> findAll - GET all Invites`
- `public ResponseEntity<Object> findById - GET Invite by id`
- `public ResponseEntity<Object> add - Add a Invite (POST)`
- `public ResponseEntity<Object> edit - Edit a Invite (PUT)`
- `public ResponseEntity<Object> deleteById - delete a Invite (DELETE)`

controller.GlobalExceptionHandler

- public ResponseEntity<ErrorResponse> handleException(DataIntegrityViolationException ex) – handler for DataIntegrityViolationException errors
- public ResponseEntity<ErrorResponse> handleException(Exception ex) – generic handler for all other exceptions

controller.ErrorResponse

- private final LocalDateTime timestamp – variable that tracks the exact time the error occurred
- private final String message -- variable for error message
- public String getMessage() – gets the error message

- public ErrorResponse() – gets the whole error response
- public static ResponseEntity<ErrorResponse> build(String message) - creates & returns an error response with the passed in message

Data:

data.UserJdbcTemplateRepository

- `private final jdbcTemplate`
- `public UserJdbcTemplateRepository(JdbcTemplate jdbcTemplate)`
- `public List<User> findById(String)` -- finds a user by id
- `public User add(User user)` -- create a User
- `public boolean update(User user)` -- update a User
- `public boolean deleteById(int)` -- delete a User by its id
- `private List<User> findAll()` -- finds all Users in the database

data.UserRepository (interface)

Contract for UserJdbcTemplateRepository

- `List<User> findByAll()` -- Finds all users
- `User findById(int id)` -- Finds user by id
- `User add(User user)` -- Adds a user
- `Boolean update(User user)` -- update a user
- `Boolean delete(int id)` -- delete a user

data.CalendarJdbcTemplateRepository

- `private final jdbcTemplate`
- `public CalendarJdbcTemplateRepository(JdbcTemplate jdbcTemplate)`
- `public List<Calendar> findById(String)` -- finds a calendar by id
- `public Calendar add(Calendar calendar)` -- create a calendar
- `public boolean update(Calendar calendar)` -- update a calendar
- `public boolean deleteById(int)` -- delete a calendar by its id
- `private List<Calendar> findAll()` -- finds all calendar in the database

data.CalendarRepository (interface)

Contract for CalendarJdbcTemplateRepository

- `List<Calendar> findByAll()` -- Finds all calendar
- `Calendar findById(int id)` -- Finds user by id
- `Calendar add(Calendar calendar)` -- Adds a calendar
- `Boolean update(Calendar calendar)` -- update a calendar
- `Boolean delete(int id)` -- delete a Event

data.EventJdbcTemplateRepository

- `private final jdbcTemplate`
- `public EventJdbcTemplateRepository(JdbcTemplate jdbcTemplate)`
- `public List<Event> findById(String)` -- finds a Event by id
- `public Event add(Event event)` -- create a event
- `public boolean update(Event event)` -- update a event
- `public boolean deleteById(int)` -- delete a event by its id
- `private List<Event> findAll()` -- finds all event in the database

data.EventRepository (interface)

Contract for EventJdbcTemplateRepository

- `List<Event> findByAll()` -- Finds all event
- `Event findById(int id)` -- Finds user by id
- `Event add(Event event)` -- Adds a event
- `Boolean update(Event event)` -- update a event
- `Boolean delete(int id)` -- delete a event

data.AttendeeJdbcTemplateRepository

- `private final jdbcTemplate`
- `public AttendeeJdbcTemplateRepository(JdbcTemplate jdbcTemplate)`
- `public List<Attendee> findById(String)` -- finds a attendee by id
- `public Attendee add(Attendee attendee)` -- create a attendee
- `public boolean update(Attendee attendee)` -- update a attendee
- `public boolean deleteById(int)` -- delete a attendee by its id
- `private List<Attendee> findAll()` -- finds all attendee in the database

data.AttendeeRepository (interface)

Contract for AttendeeJdbcTemplateRepository

- `List<Attendee> findByAll()` -- Finds all attendee
- `Attendee findById(int id)` -- Finds user by id
- `Attendee add(Attendee attendee)` -- Adds a attendee
- `Boolean update(Attendee attendee)` -- update a attendee
- `Boolean delete(int id)` -- delete a Event

data.RoleJdbcTemplateRepository

- `private final jdbcTemplate`
- `public RoleJdbcTemplateRepository(JdbcTemplate jdbcTemplate)`
- `public List<Role> findById(String)` -- finds a role by id
- `public Role add(Role role)` -- create a role
- `public boolean update(Role role)` -- update a role
- `public boolean deleteById(int)` -- delete a role by its id
- `private List<Role> findAll()` -- finds all role in the database

data.RoleRepository (interface)

Contract for RoleJdbcTemplateRepository

- `List<Role> findByAll()` -- Finds all role
- `Role findById(int id)` -- Finds role by id
- `Role add(Role role)` -- Adds a role
- `Boolean update(Role role)` -- update a role
- `Boolean delete(int id)` -- delete a Event

data.InviteJdbcTemplateRepository

- `private final jdbcTemplate`
- `public InviteJdbcTemplateRepository(JdbcTemplate jdbcTemplate)`
- `public List<Invite> findById(String)` -- finds a invite by id
- `public Invite add(Invite invite)` -- create a invite
- `public boolean update(Invite invite)` -- update a invite
- `public boolean deleteById(int)` -- delete a invite by its id
- `private List<Invite> findAll()` -- finds all invite in the database

data.InviteRepository (interface)

Contract for InviteJdbcTemplateRepository

- `List<Invite> findByAll()` -- Finds all invite
- `Invite findById(int id)` -- Finds invite by id
- `Invite add(Invite invite)` -- Adds a invite
- `Boolean update(Invite invite)` -- update a invite
- `Boolean delete(int id)` -- delete a Event

Domain

domain.UserResult

- `private ArrayList<String> messages` -- error messages
- `private User User` -- an optional User
- `public boolean isSuccess()` -- calculated getter, true if no error messages
- `public List<String> getMessages()` -- messages getter, create a new list
- `public User getUser()` -- User getter
- `public void setUser(User)` -- User setter
- `public void addMessage(String)` -- adds an error message to messages

domain.UserService

- `private UserRepository repository` -- required data dependency
- `public UserService(UserRepository)` -- constructor
- `public List<User> findBySection(String)` -- pass-through to repository
- `public UserResult add(User)` -- validate, then add via repository
- `public UserResult update(User)` -- validate, then update via repository
- `public UserResult deleteById(int)` -- pass-through to repository
- `private UserResult validate(User)` -- general-purpose validation routine

domain.RoleService

- `private RoleRepository repository` -- required data dependency
- `public RoleService(RoleRepository)` -- constructor
- `public List<Role> findBySection(String)` -- pass-through to repository
- `public RoleResult add(Role)` -- validate, then add via repository
- `public RoleResult update(Role)` -- validate, then update via repository
- `public RoleResult deleteById(int)` -- pass-through to repository
- `private RoleResult validate(Role)` -- general-purpose validation routine

domain.AttendeeService

- `private AttendeeRepository repository` -- required data dependency
- `public AttendeeService(AttendeeRepository)` -- constructor
- `public List<Attendee> findBySection(String)` -- pass-through to repository
- `public AttendeeResult add(Attendee)` -- validate, then add via repository
- `public AttendeeResult update(Attendee)` -- validate, then update via repository
- `public AttendeeResult deleteById(int)` -- pass-through to repository
- `private AttendeeResult validate(Attendee)` -- general-purpose validation routine

domain.CalendarService

- `private CalendarRepository repository` -- required data dependency
- `public CalendarService(CalendarRepository)` -- constructor
- `public List<Calendar> findBySection(String)` -- pass-through to repository
- `public CalendarResult add(Calendar)` -- validate, then add via repository
- `public CalendarResult update(Calendar)` -- validate, then update via repository
- `public CalendarResult deleteById(int)` -- pass-through to repository
- `private CalendarResult validate(Calendar)` -- general-purpose validation routine

domain.InviteService

- `private InviteRepository repository` -- required data dependency
- `public InviteService(InviteRepository)` -- constructor
- `public List<Invite> findBySection(String)` -- pass-through to repository
- `public InviteResult add(Invite)` -- validate, then add via repository
- `public InviteResult update(Invite)` -- validate, then update via repository
- `public InviteResult deleteById(int)` -- pass-through to repository
- `private InviteResult validate(Invite)` -- general-purpose validation routine

domain.EventService

- `private EventRepository repository` -- required data dependency
- `public EventService(EventRepository)` -- constructor
- `public List<Event> findBySection(String)` -- pass-through to repository

- `public EventResult add(Event)` -- validate, then add via repository
- `public EventResult update(Event)` -- validate, then update via repository
- `public EventResult deleteById(int)` -- pass-through to repository
- `private EventResult validate(Event)` -- general-purpose validation routine

Models:

models.RoleType

An enum with two types, admin & member

models.CalType

An enum with two types, personal & organization.

models.EventType

An enum with two types, personal & broadcast.

models.User

- `private int user_id`
- `private String first_name`
- `private String last_name`
- `private String email`
- `private String password`
- `private boolean isAdmin`
- Full getters and setters
- override `equals` and `hashCode`

models.Calendar

- `private int calendar_id`
- `private String name`
- `private Enum CalType`
- `private int admin_id`
- Full getters and setters
- override `equals` and `hashCode`

models.Events

- `private int event_id`
- `private LocalDate day`
- `private Timestamp startTime`
- `private Timestamp endTime`
- `private Enum EventType`
- `private String recurr_type`
- `private int calendar_id`
- `private int creator_id`
- `private String description`
- Full getters and setters
- override `equals` and `hashCode`

models.Attendee

- `private int attendee_id`
- `private int event_id`
- `private int user_id`
- `private boolean is_confirmed`
- Full getters and setters
- override `equals` and `hashCode`

models.Role

- `private int role_id`
- `private int calendar_id`
- `private int user_id`
- `private enum RoleType`
- Full getters and setters
- override `equals` and `hashCode`

Client

Components:

- Navbar
- ChatBot
- EventForm
- InvitationList
- Calendar

- PopupModal (this will

Pages:
- Home (shows main calendar)
- Create calendar
- Invites (shows calendar and event invites)

Services
- apiService.js
- openaiService.js

Context (a react feature that helps us manage states in multiple components)
- CalendarContext.js

App.jsx
index.js
package.json

Steps

1. Create a Maven project.
2. Add jUnit 5, Jupiter, as a Maven dependency and refresh Maven
3. Create a folder called server & create packages sql, controller, data & model
4. In the server package,create two sql classes, calendar-schema-production.sql & calendar-schema-test.sql.
5. In MySQL server, create the tables user, calendar, event, attendee, role & invitation in both classes. Populate the tables in both classes with test data that will be used in the java backend. calendar-schema-production.sql will be the production data & calendar-schema-test.sql will be for the test data.
6. Create the enums for RoleType, CalType & EventType in the Models package
7. Create models for Users, Calendar, Event, Attendee, Role & Invite.
8. Create the data layer's custom DataException
9. Create the interfaces UserRepository, CalendarRepository, EventRepository, AttendeeRepository, RoleRepository, InviteRepository in the Data package. These interfaces should all have similar findByAll, findById, add, update, delete classes that would implement the CRUD operations.

10. Create a package within Data called Mappers which would include the classes: UserMapper, EventMapper, CalendarMapper, AtendeeMapper, RoleMapper & InviteMapper. Each class will have mapRow(ResultSet resultSet, int i) methods that throw SQLExceptions & would set the attributes of each object from the model class based on the SQL data being parsed. Annotate each Repository with @Repository
11. Implement the interfaces in Data with JDBCRepository classes for each interface.
12. Generate test classes for every method, in each JDBCRepository class in test classes located in src/test/java/learn/calendar/data/. Include happy & unhappy paths to verify each method in the JDBCRepository classes.
13. In the domain package, create & implement Response, Result & ResultType classes that will be used by the service classes for results.
14. Create the service classes, EventService, InviteService, RoleService, AttendeeService & UserService. Each service class should have a constructor, findBySection, add, update, deleteById methods along with a validate helper method used for in the add, update & delete methods. Annotate each service with @Service.
15. Create test classes for each service class, the test classes should be located in src/test/java/learn/calendar/domain. Annotate each service test class with @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.NONE) & use @MockBean for the tests.
16. Create ErrorResponse & GlobalExceptionHandler in the Controller package for handling errors & exceptions. In GlobalExceptionHandler, create one specific exception handler for DataIntegrityViolationException & create another for every other exception.
17. Create controller classes for User, Event, Attendee, Calendar, Invite & Role. Each controller class with have a reference to the service class of the object & have findAll, findById, add, edit, deleteById classes which use these methods from the object's service classes. Annotate each controller class with @RestController, @CrossOrigins @RequestMapping & have each method annotated with either @PostMapping, @GetMapping, @PutMapping or @DeleteMapping.
18. Create a http package & create http classes for the 6 models in this project. Write http calls for the CRUD operations that would be linked to the controller in the controller package. Write happy & unhappy paths for each operations to be able to test you would

get the expected outputs for valid & invalid calls. Test each controller using the http classes.

19. Create a react project in the client folder using CRA (create-react-app) & remove the unnecessary classes.
20. Add Bootstrap and Tailwind CSS to the `public/index.html` file & install react-router-dom
21. Integrate Clerk for Authentication:
    - Install Clerk for user authentication.
    - Set up Clerk by wrapping the main application in its authentication provider.
    - Add routes for login, sign-up, and handling authenticated user views (e.g., event creation/editing views).
22. Set Up Global State Management
    - Create CalendarContext to manage calendar state
23. Design Layout and Pages (Using Tailwind CSS):
    - Build the main components and layout (e.g., Header, Calendar view, Sidebar, Invites section) using Tailwind CSS.
    - Structure the app with key views:
        - Home Page: Main calendar view with event management and toggle for different calendars.
        - Invites Page**:** Separate page for managing invites.
        - Not Found: Error page for when a page is not found
        - Event Creation/Editing Form: Modal or page for adding and editing events.
    - Implement mobile responsiveness using Tailwind's responsive classes.
24. Implement Calendar Display and Interactions:
    - Use react-big-calendar to render the main calendar.
    - Set up event listeners for adding, updating, or deleting events directly from the calendar view.
    - Implement features like filtering by selected calendars (checkboxes in the sidebar).

26. Integrate the Event Checkbox Filtering

- Create a sidebar with collapsible sections for My Calendars and Other Calendars.
- Implement a list of checkboxes that allow users to select which calendars to display.
- When a user checks/unchecks a calendar, update the state and filter the events displayed on the calendar.

27. Set Up Fetch API Requests (Java Spring Back-End Integration):

    - Create a services directory that will contain our Java Spring fetch API CRUD operations for the calendar, events, attendees, and invites.
    - Implement the API service for fetching, adding, updating, and deleting calendar events and data using fetch/axios.

28. OpenAI Chatbot Integration for Event Management:

    - Integrate OpenAI's API to add a chatbot feature for creating, editing, or deleting events.
    - The chatbot should interact with the user, collect information (like event name, date, etc.), and use the API to perform CRUD operations based on user input.
    - Display the chatbot in a text area on the main calendar page.

29. Build and Implement Invitations Page:

    - Create a dedicated page where users can view, accept, or decline invitations to events.
    - Fetch invitations from the back-end, display them in a list, and allow users to manage them with corresponding buttons (accept/decline).

30. Set Up Routing with React Router:

- Add routing to navigate between the Calendar view, Invites page, and possibly Profile page.
- Make sure routes are protected, i.e., only accessible to logged-in users.

31. Finalize UI/UX Design with Tailwind:

- Polish the UI by refining the calendar view, invites page, and sidebar for better user experience.
- Make sure the design is responsive to different views

Controller Perspective:

1. View Calendars by

a) Controller Action:

- Collect the calendar section type (e.g., "My Calendars" or "Other Calendars") from the client-side request.
- Use the service layer to fetch all calendars based on the section (either "My Calendars" for personal ones or "Other Calendars" for shared).
- Return the list of calendars

b) Controller Outline:

- Accept a request parameter (e.g., calendar type).
- Call the service to retrieve the list of calendars based on this section.
- Return the list to the front-end as a JSON response.

2. Add a Calendar

a) Controller Action:

- Receive the full calendar details from the client-side (name, type, creator, etc.).
- Pass the calendar data to the service layer for validation and saving.
- Send the result back to the client to confirm that the calendar was successfully added.

b) Controller Outline:

- Accept a request body containing the calendar details Validate the input (via annotations or manually).
- Call the service to save the new calendar.
- Return a success or failure response (with calendar details if successful).

3. Update a Calendar

a) Controller Action:

- Receive the section name or calendar ID from the client-side request.
- Use the service to fetch the calendar that needs updating.
- Allow the user to modify calendar properties (e.g., calendar name, type).
- Once the user makes changes, pass the updated calendar details to the service to save the changes.
- Send back the updated calendar or a success response.

b) Controller Outline:

- Accept a request body with the updated calendar data
- Fetch the calendar by its ID via the service.
- Apply the updates (make sure to validate the changes).
- Save the updated calendar using the service layer.
- Return the updated calendar or success status to the front-end.

4. Delete a Calendar

a) Controller Action:

- Receive the calendar ID from the request.
- Use the service layer to fetch the calendar.
- Confirm that the calendar can be deleted (e.g., only allow admins to delete).
- Pass the ID to the service to delete the calendar.
- Send back a success or failure message to the client.

b) Controller Outline:

- Accept a request parameter or path variable containing the calendar ID
- Check permissions (ensure that only admins can delete the calendar).
- Call the service layer to delete the calendar by its ID.
- Return success or failure as a response.

5. View Events by Calendar

a) Controller Action:

- Receive the calendar ID from the client request.
- Use the service layer to fetch all events associated with that calendar.
- Return the list of events to the view (front-end).

b) Controller Outline:

- Accept a path variable with the calendar ID
- Call the service to retrieve events associated with that calendar.
- Return the event list as a JSON response to the front-end.

6. Add an Even

a) Controller Action:

- Receive the event details from the client request (e.g., event name, date, time, type, etc.).
- Pass the event data to the service layer for validation and saving.
- Return the created event or a success message to the client.

b) Controller Outline:

- Accept a request body containing the event details
- Validate the event data.
- Call the service to save the new event.
- Return the saved event or a success response to the front-end.

7. Update an Event

a) Controller Action:

- Receive the event ID from the client request.
- Use the service to fetch the event.
- Allow the user to modify event details (e.g., name, date, time).
- Save the updated event using the service.
- Return the updated event or a success message to the client.

b) Controller Outline:

- Accept a path variable with the event ID and a request body with updated details
- Fetch the event using the service.
- Apply the changes to the event details.
- Call the service to save the updated event.
- Return the updated event or success status.

8. Delete an Event

a) Controller Action:

- Receive the event ID from the client request.
- Use the service to fetch the event.
- Delete the event using the service.
- Return a success or failure message to the client.

b) Controller Outline:

- Accept a path variable with the event ID
- Call the service to delete the event.
- Return success or failure as a response.

9. Manage Invitations for Events

a) Controller Action:

- Receive the event ID and user ID.
- Use the service to send or manage the invitation (accept/decline).
- Return the status of the invitation to the client.

b) Controller Outline:

- Accept a path variable with the event ID and request body with user actions (e.g., accept/decline).
- Call the service to handle the invitation logic (inviting users, tracking responses).
- Return the updated invitation status.