ROOMBOOK SWITCH

```
public class FuckingSwitch extends Application {

    private static Stage primaryStage;

    @Override
    public void start(Stage primaryStage) throws Exception {
        FuckingSwitch.primaryStage = primaryStage;


        Parent root =
FXMLLoader.load(getClass().getResource("/org/ukdw/login-view.fxml"));


        Scene scene = new Scene(root);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private static Parent loadFxml(String fxml) throws Exception {
        FXMLLoader loader = new
FXMLLoader(RoomBookApp.class.getResource(fxml));
        return loader.load();
    }

    public static void setShit(String fxml) throws Exception{
        Stage n = new Stage();
        n.setScene(new Scene( loadFxml(fxml) ));

        n.initOwner(primaryStage);
        n.initModality(Modality.WINDOW_MODAL);

        n.showAndWait();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

ROOMBOOK APP

```
public class RoomBookApp extends Application {
    private static Stage primaryStage;

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) throws IOException {
        primaryStage = stage;
        if (SessionManager.getInstance().isLoggedIn()) {
            FXMLLoader fxmlLoader = new
FXMLLoader(RoomBookApp.class.getResource("main-view.fxml"));
            Scene scene = new Scene(fxmlLoader.load());
            primaryStage.setTitle("Room Book");
            primaryStage.setScene(scene);
        } else {
            FXMLLoader fxmlLoader = new
FXMLLoader(RoomBookApp.class.getResource("login-view.fxml"));
            Scene scene = new Scene(fxmlLoader.load());
            primaryStage.setTitle("Login");
            primaryStage.setScene(scene);
        }
        primaryStage.show();
    }

    public static Scene loadFxml(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(RoomBookApp.class.getResource(fxml));
        Scene scene = new Scene(fxmlLoader.load());
        return scene;
//        primaryStage.setTitle("Room Book");
//        primaryStage.setScene(scene);
//        primaryStage.show();
    }
}
```

DB CONNECTION MANAGER

```
public class DBConnectionManager {
    private static final String DB_URL = "jdbc:sqlite:ukdwroombook.db";
    private static Connection connection;

    private DBConnectionManager() {
    }

    public static Connection getConnection() {
        try {
            if (connection == null || connection.isClosed()) {
                connection = DriverManager.getConnection(DB_URL);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return connection;
    }

    public static void closeConnection() {
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    public static void createTables() {
        String userTableSql = "CREATE TABLE IF NOT EXISTS users ("
                + "email TEXT NOT NULL PRIMARY KEY,"
                + "username TEXT NOT NULL UNIQUE,"
                + "password TEXT NOT NULL"
                + ")";
        String gedungTableSql = "CREATE TABLE IF NOT EXISTS gedung ("
                + "id INTEGER PRIMARY KEY AUTOINCREMENT,"
                + "nama TEXT NOT NULL,"
                + "alamat TEXT"
                + ")";
        String ruanganTableSql = "CREATE TABLE IF NOT EXISTS ruangan ("
                + "id INTEGER PRIMARY KEY AUTOINCREMENT,"
                + "nama TEXT NOT NULL,"
                + "gedung_id INTEGER NOT NULL,"
                + "FOREIGN KEY (gedung_id) REFERENCES gedung(id)"
                + ")";
        String pemesananTableSql = "CREATE TABLE IF NOT EXISTS pemesanan
("
                + "id INTEGER PRIMARY KEY AUTOINCREMENT,"
                + "user_email TEXT NOT NULL,"
                + "ruangan_id INTEGER NOT NULL,"
                + "checkindate TEXT NOT NULL,"
                + "checkoutdate TEXT NOT NULL,"
                + "checkintime TEXT NOT NULL,"
                + "checkouttime TEXT NOT NULL,"
                + "FOREIGN KEY (user_email) REFERENCES users(email),"
                + "FOREIGN KEY (ruangan_id) REFERENCES ruangan(id)"
                + ")";

        try (Statement stmt = getConnection().createStatement()) {
            stmt.execute(userTableSql);
            stmt.execute(gedungTableSql);
            stmt.execute(ruanganTableSql);
            stmt.execute(pemesananTableSql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public static int get_last_inserted_id() {
        String query = "Select last_insert_rowid();";

        try (PreparedStatement stmt =
connection.prepareStatement(query)) {

            ResultSet res = stmt.executeQuery();
            if (res.next()){
                return res.getInt(1);
            }

        } catch (SQLException e) {
            e.printStackTrace();
        } return 0;
    }
}
```

GEDUNG

```
public class Gedung implements Serializable {

    private int id;
```

```
    private String nama;
    private String alamat;

    public Gedung(int id, String nama, String alamat) {
        this.id = id;
        this.nama = nama;
        this.alamat = alamat;
    }

    public void setId(int id) {
      this.id = id;
    }
    public void setNama(String nama) {
        this.nama = nama;
    }
    public void setAlamat(String alamat) {
        this.alamat = alamat;
    }
    public String getNama() {
        return nama;
    }
    public String getAlamat() {
        return alamat;
    }
    public int getId() {
        return id;
    }
}
```

GEDUNG MANAGER

```
public class GedungManager {
   private Connection connection;

   public GedungManager(Connection connection) {
       this.connection = connection;
   }

   // Add methods for course management (create, edit, delete)
   public boolean addGedung(String nama, String alamat)  {
       String query = "INSERT INTO gedung (nama, alamat) VALUES (?,
?)";
       try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
           preparedStatement.setString(1, nama);
           preparedStatement.setString(2, alamat);
           int rowsAffected = preparedStatement.executeUpdate();
           return rowsAffected > 0;
       } catch (SQLException e) {
           e.printStackTrace();
           return false;
       }
   }

   public boolean editGedung(int gedungId, String newNama, String
newAlamat) {
       String query = "UPDATE gedung SET nama = ?, alamat = ? WHERE id
= ?";
       try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
           preparedStatement.setString(1, newNama);
           preparedStatement.setString(2, newAlamat);
           preparedStatement.setInt(3, gedungId);
           int rowsAffected = preparedStatement.executeUpdate();
           return rowsAffected > 0;
       } catch (SQLException e) {
           e.printStackTrace();
           return false;
       }
   }

   public boolean deleteGedung(int gedungId) {
       String query = "DELETE FROM gedung WHERE id = ?";
       try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
           preparedStatement.setInt(1, gedungId);
           int rowsAffected = preparedStatement.executeUpdate();
           return rowsAffected > 0;
       } catch (SQLException e) {
           e.printStackTrace();
           return false;
       }
   }

   public List<Gedung> getAllGedung() {
       List<Gedung> gedungs = new ArrayList<>();
       String query = "SELECT * FROM gedung";
       try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
           ResultSet resultSet = preparedStatement.executeQuery();
           while (resultSet.next()) {
               int id = resultSet.getInt("id");
               String nama = resultSet.getString("nama");
               String alamat = resultSet.getString("alamat");
               Gedung gedung = new Gedung(id, nama, alamat);
               gedungs.add(gedung);
           }
       } catch (SQLException e) {
           e.printStackTrace();
           // Handle database query error
       }
       return gedungs;
   }

   public String getIdGedung(String namagedung) {
       String sql = "select id from gedung where nama = ?";

       try(PreparedStatement stmt = connection.prepareStatement(sql)) {
           stmt.setString(1, namagedung);
           ResultSet res = stmt.executeQuery();

           return res.getString(1);
       } catch (SQLException e) {
          e.printStackTrace();
       } return "";
   }

   public String getNamaGedung(String id) {
       String sql = "select nama from gedung where id = ?";

       try(PreparedStatement stmt = connection.prepareStatement(sql)) {
           stmt.setString(1, id);
           ResultSet res = stmt.executeQuery();

           return res.getString(1);
       } catch (SQLException e) {
           e.printStackTrace();
       } return "";
   }

   public static void main(String[] args) {
       GedungManager gm = new
GedungManager(DBConnectionManager.getConnection());
       System.out.println(gm.getIdGedung("Koinonia"));
   }
}
```

GEDUNG CONTROLLER

```
public class GedungView implements Initializable {
   @FXML
   private Button btnTambah;
   @FXML
   private Button btnUbah;
   @FXML
   private TextArea descriptionTxtArea;
   @FXML
   private TextField idTxtFld;
   @FXML
   private TextField searchBox;
   @FXML
   private TableView<Gedung> table;
   @FXML
   private TableColumn<Gedung, Number> tblColId;
   @FXML
   private TableColumn<Gedung, String> tblColTitle;
   @FXML
   private TableColumn<Gedung, String> tblColDescription;
   @FXML
   private TextField titleTxtFld;

   Gedung gedung;
   private FilteredList<Gedung> filteredList;
   private GedungManager manager = new
GedungManager(DBConnectionManager.getConnection());

   @Override
   public void initialize(URL location, ResourceBundle resources) {
       filteredList = new
FilteredList<>(FXCollections.observableList(FXCollections.observableArr
ayList()));
       table.setItems(filteredList);
       searchBox.textProperty().addListener(
               (observableValue, oldValue, newValue) ->
filteredList.setPredicate(createPredicate(newValue))
       );
```

```
        tblColId.setCellValueFactory(new PropertyValueFactory<>("id"));
        tblColTitle.setCellValueFactory(new
PropertyValueFactory<>("nama"));
        tblColDescription.setCellValueFactory(new
PropertyValueFactory<>("alamat"));

        titleTxtFld.setEditable(true);
        descriptionTxtArea.setEditable(true);

        getAllData();
        table.getSelectionModel().selectedItemProperty().addListener(new
ChangeListener<Gedung>() {
            @Override
            public void changed(ObservableValue<? extends Gedung>
observable, Gedung oldValue, Gedung newValue) {
                if (newValue != null) {
                    gedung = observable.getValue();
                    idTxtFld.setText(String.valueOf(gedung.getId()));
                    titleTxtFld.setText(gedung.getNama());
                    descriptionTxtArea.setText(gedung.getAlamat());
                }
            }
        });
    }

    public void handleAddAction(ActionEvent actionEvent) {
        if (nama.isEmpty() || desc.isEmpty()) {
            showAlert(Alert.AlertType.ERROR, "Error", "Input tidak boleh
kosong.");
            return;
        }

        getAllData();
        boolean namaSudahAda = getObservableList().stream()
                .anyMatch(g -> g.getNama().equalsIgnoreCase(nama));

        if (namaSudahAda) {
            showAlert(Alert.AlertType.ERROR, "Error", "Gedung sudah
terdaftar.");
        }

        if (manager.addGedung(nama, desc)) {
            getAllData();
        }

    }

    public void handleDeleteAction(ActionEvent actionEvent) {
        Integer id = Integer.parseInt(idTxtFld.getText());
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Konfirmasi");
        alert.setHeaderText("Hapus Gedung?");
        alert.setContentText("Apakah anda yakin ingin menghapus data?");
        Optional<ButtonType> result = alert.showAndWait();
        if (result.get() == ButtonType.OK) {
            manager.deleteGedung(id);
            getAllData();
        } else {
            alert.close();
        }
        clearData();
    }

    public void handleEditAction(ActionEvent actionEvent) {
        Integer id = Integer.parseInt(idTxtFld.getText());
        String nama = titleTxtFld.getText();
        String desc = descriptionTxtArea.getText();
        if (nama.isEmpty() || desc.isEmpty()) {
            showAlert(Alert.AlertType.ERROR, "Error", "Input tidak boleh
kosong.");
            return;
        }

        getAllData();
        if (manager.editGedung(id, nama, desc)) {
            getAllData();
        }
        clearData();

    }

    public void handleClearSearchText(ActionEvent actionEvent) {
        idTxtFld.clear();
        titleTxtFld.clear();
        descriptionTxtArea.clear();
        table.getSelectionModel().clearSelection();
        gedung = null;
    }

    private ObservableList<Gedung> getObservableList() {
        return (ObservableList<Gedung>) filteredList.getSource();
    }

    private void getAllData() {
        getObservableList().clear();
        getObservableList().addAll(manager.getAllGedung());
    }

    private Predicate<Gedung> createPredicate(String searchText) {
        return gedung -> {
            if (searchText == null || searchText.isEmpty()) return true;
            return searchForGedung(gedung, searchText);
        };
    }

    private boolean searchForGedung(Gedung gedung, String searchText) {
        return (gedung.getId() == Integer.parseInt(idTxtFld.getText()))
||

gedung.getNama().toLowerCase().contains(searchText.toLowerCase()) ||

gedung.getAlamat().toLowerCase().contains(searchText.toLowerCase());
    }

    private void showAlert(Alert.AlertType alertType, String title,
String content) {
        Alert alert = new Alert(alertType);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(content);
        alert.showAndWait();
    }

    private void clearData(){
        descriptionTxtArea.clear();
        idTxtFld.clear();
        titleTxtFld.clear();

    }

}
```

RUANGAN

```
public class Ruangan implements Serializable {

    private int id;
    private String name;
    private int idGedung;
    private String namaGedung;

    public String getNamaGedung() {return namaGedung;}

    public void setNamaGedung(String namaGedung) {
        this.namaGedung = namaGedung;
    }

    public Ruangan(int id, String name, int idGedung) {
        this.id = id;
        this.name = name;
        this.idGedung = idGedung;
    }

    public Ruangan(int id, String name, int idGedung, String namaGedung)
{
        this.id = id;
        this.name = name;
        this.idGedung = idGedung;
        this.namaGedung = namaGedung;
    }

    public String getName() {return name;}
    public int getIdGedung() {return idGedung;}
    public int getId() {return id;}
    public void setId(int id) {this.id = id;}
    public void setName(String name) {this.name = name;}

    public void setIdGedung(int idGedung) {
        this.idGedung = idGedung;
    }
}
```

RUANGAN MANAGER

```
public class RuanganManager {
    private Connection connection;

    public RuanganManager(Connection connection) {
        this.connection = connection;
    }

    private static GedungManager gm = new
GedungManager(DBConnectionManager.getConnection());

    // Add methods for course management (create, edit, delete)
    public boolean addRuangan(String nama, int idGedung) {
        String query = "INSERT INTO ruangan (nama, gedung_id) VALUES (?,
?)";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
            preparedStatement.setString(1, nama);
            preparedStatement.setInt(2, idGedung);
            int rowsAffected = preparedStatement.executeUpdate();
            return rowsAffected > 0; // Course added successfully if
rows were inserted
        } catch (SQLException e) {
            e.printStackTrace();
            // Handle database query error
            return false;
        }
    }

    public boolean editRuangan(String newNama, int idGedung, int
idRuangan) {
        String query = "UPDATE ruangan SET nama = ?, gedung_id = ? WHERE
id = ?";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
            preparedStatement.setString(1, newNama);
            preparedStatement.setInt(2, idGedung);
            preparedStatement.setInt(3, idRuangan);
            int rowsAffected = preparedStatement.executeUpdate();
            return rowsAffected > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    public boolean deleteRuangan(int idRuangan) {
        String query = "DELETE FROM ruangan WHERE id = ?";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
            preparedStatement.setInt(1, idRuangan);
            int rowsAffected = preparedStatement.executeUpdate();
            return rowsAffected > 0;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    public ObservableList<Ruangan> getAllRuangan() {
        ObservableList<Ruangan> ruangans =
FXCollections.observableArrayList();
        String query = "SELECT * FROM ruangan";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                int id = resultSet.getInt("id");
                String nama = resultSet.getString("nama");
                int idGedung = resultSet.getInt("gedung_id");
                Ruangan ruangan = new Ruangan(id, nama, idGedung,
gm.getNamaGedung(String.valueOf(idGedung)));
                ruangans.add(ruangan);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return ruangans;
    }
}
```

RUANGAN VIEW

```
public class RuanganView implements Initializable {

    @FXML
    private Button btnHapus;

    @FXML
    private Button btnTambah;

    @FXML
    private Button btnUbah;

    @FXML
    private ChoiceBox<String> cbGedungId;

    @FXML
    private TableColumn<Ruangan, Number> colKelasId;
    @FXML
    private TableColumn<Ruangan, String> colNameKelas;
    @FXML
    private TableColumn<Ruangan, String> colLokasi;

    @FXML
    private Label lblKelas;

    @FXML
    private TextField namaIdTxtFld;

    @FXML
    private TextField searchBox;

    @FXML
    private TableView<Ruangan> table;


    Gedung gedung;
    Ruangan ruangan;
    private final GedungManager gedungManager = new
GedungManager(DBConnectionManager.getConnection());
    private final RuanganManager ruanganManager = new
RuanganManager(DBConnectionManager.getConnection());
    private final ObservableList<Ruangan> ruanganList =
FXCollections.observableArrayList();
    private FilteredList<Ruangan> filteredList;


    @Override
    public void initialize(URL location, ResourceBundle resources) {
        namaIdTxtFld.setEditable(true);

        colKelasId.setCellValueFactory(new
PropertyValueFactory<>("id"));
        colNameKelas.setCellValueFactory(new
PropertyValueFactory<>("name"));
        colLokasi.setCellValueFactory(new
PropertyValueFactory<>("namaGedung"));

        loadGedungIds();
        getAllData();

        filteredList = new FilteredList<>(ruanganList, p -> true);
        table.setItems(filteredList);



table.getSelectionModel().selectedItemProperty().addListener((obs,
oldVal, newVal) -> {
            if (newVal != null) {
                namaIdTxtFld.setText(newVal.getName());
                cbGedungId.setValue(newVal.getNamaGedung());

            }
        });

        searchBox.textProperty().addListener((obs, oldVal, newVal) -> {
            filteredList.setPredicate(ruangan -> {
                if (newVal == null || newVal.isEmpty()) return true;
                String lower = newVal.toLowerCase();
                return ruangan.getName().toLowerCase().contains(lower)
                        ||
ruangan.getNamaGedung().toLowerCase().contains(lower);
            });
        });
    }


    @FXML
    void handleAddAction(ActionEvent event) {

        String nama = namaIdTxtFld.getText().trim();
        String namaGedung = cbGedungId.getValue();
```

```
        if (nama.isEmpty() || namaGedung == null) {
            showAlert(Alert.AlertType.ERROR, "Validasi Gagal", "Nama dan
Gedung harus diisi.");
            return;

        }

        int idGedung =
Integer.parseInt(gedungManager.getIdGedung(namaGedung));

        boolean exists = ruanganList.stream()
                .anyMatch(r -> r.getName().equalsIgnoreCase(nama) &&
r.getIdGedung() == idGedung);

        if (exists) {
            showAlert(Alert.AlertType.ERROR, "Validasi Gagal", "Ruangan
sudah ada di gedung ini.");
            return;
        }

        if (ruanganManager.addRuangan(nama, idGedung)) {
            showAlert(Alert.AlertType.INFORMATION, "Sukses", "Ruangan
berhasil ditambahkan.");
            getAllData();
            clearForm();
        } else {
            showAlert(Alert.AlertType.ERROR, "Gagal", "Gagal menambahkan
ruangan.");
        }
    }

    @FXML
    void handleClearSearchText(ActionEvent event) {
        searchBox.clear();
    }

    @FXML
    void handleDeleteAction(ActionEvent event) {
        Ruangan selected = table.getSelectionModel().getSelectedItem();
        if (selected == null) {
            showAlert(Alert.AlertType.WARNING, "Peringatan", "Pilih
ruangan yang ingin dihapus.");
            return;
        }

        Alert confirm = new Alert(Alert.AlertType.CONFIRMATION);
        confirm.setTitle("Konfirmasi Hapus");
        confirm.setHeaderText("Yakin ingin menghapus ruangan?");
        confirm.setContentText("Ruangan: " + selected.getName());
        Optional<ButtonType> result = confirm.showAndWait();



        if (result.isPresent() && result.get() == ButtonType.OK) {
            if (ruanganManager.deleteRuangan(selected.getId())) {
                showAlert(Alert.AlertType.INFORMATION, "Berhasil", "Data
ruangan dihapus.");
                getAllData();
                clearForm();
            } else {
                showAlert(Alert.AlertType.ERROR, "Gagal", "Gagal
menghapus ruangan.");
            }
        }

    }


    @FXML
    void handleEditAction(ActionEvent event) {
        Ruangan selected = table.getSelectionModel().getSelectedItem();
        if (selected == null) {
            showAlert(Alert.AlertType.ERROR, "Error", "Pilih ruangan
yang ingin diubah.");
            return;
        }

        String newNama = namaIdTxtFld.getText().trim();
        String namaGedung = cbGedungId.getValue();

        if (newNama.isEmpty() || namaGedung == null) {
            showAlert(Alert.AlertType.ERROR, "Error", "Nama dan Gedung
harus diisi.");
            return;
        }

        int idGedung =
Integer.parseInt(gedungManager.getIdGedung(namaGedung));

        boolean exists = ruanganList.stream()
                .anyMatch(r -> r.getName().equalsIgnoreCase(newNama)
                        && r.getIdGedung() == idGedung
                        && r.getId() != selected.getId());

        if (exists) {
            showAlert(Alert.AlertType.ERROR, "Validasi Gagal", "Nama
ruangan sudah digunakan di gedung ini.");
            return;
        }

        if (ruanganManager.editRuangan(newNama, idGedung,
selected.getId())) {
            showAlert(Alert.AlertType.INFORMATION, "Sukses", "Data
ruangan berhasil diperbarui.");
            getAllData();
            clearForm();
        } else {
            showAlert(Alert.AlertType.ERROR, "Gagal", "Gagal mengubah
ruangan.");
        }
    }


    @FXML
    void tampilRuangan(ActionEvent event) {
        getAllData();
    }

    private void getAllData() {
        ruanganList.setAll(ruanganManager.getAllRuangan());
    }

    private void loadGedungIds() {

        List<Gedung> gedungs = gedungManager.getAllGedung();
        List<String> namaGedungs = new ArrayList<>();
        for (Gedung g : gedungs) {
            namaGedungs.add(g.getNama());
        }
        cbGedungId.getItems().setAll(namaGedungs);
    }

    private void filterTable(String keyword) {
        System.out.println(keyword.toLowerCase());
        try {
            RuanganManager ruanganManager = new
RuanganManager(DBConnectionManager.getConnection());
            List<Ruangan> allRuangan = ruanganManager.getAllRuangan();

            if (keyword == null || keyword.isBlank()) {
                table.getItems().setAll(allRuangan);
                return;
            }

            String lowerKeyword = keyword.toLowerCase();
            List<Ruangan> filtered = new ArrayList<>();
            for (Ruangan r : allRuangan) {
                if (r.getName().toLowerCase().contains(lowerKeyword) ||
                        String.valueOf(r.getId()).contains(lowerKeyword)
||

String.valueOf(r.getIdGedung()).contains(lowerKeyword) ||

r.getNamaGedung().toLowerCase().contains(lowerKeyword)
                ) {
                    filtered.add(r);
                }
            }

            table.getItems().setAll(filtered);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void clearForm() {
        namaIdTxtFld.clear();
        cbGedungId.setValue(null);
        table.getSelectionModel().clearSelection();
    }

    private void showAlert(Alert.AlertType type, String title, String
```

```
content) {
        Alert alert = new Alert(type);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(content);
        alert.showAndWait();
    }

}
```

MAIN VIEW

```
public class MainView implements Initializable {
    @FXML
    private AnchorPane rootPane;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
    }

    @FXML
    private void handleKeyInput(final InputEvent event) {
        if (event instanceof KeyEvent) {
            final KeyEvent keyEvent = (KeyEvent) event;
            if (keyEvent.isControlDown() && keyEvent.getCode() ==
KeyCode.A) {
                System.out.println("CTRL+A ditekan!");
            }
        }
    }

    @FXML
    private void doExit(final ActionEvent event) {
        Platform.exit();
    }

    @FXML
    private void doOnlineManual(final ActionEvent event) {
        try {
            Desktop.getDesktop().browse(new
URL("https://www.ukdw.ac.id").toURI());
        } catch (IOException | URISyntaxException e) {
            e.printStackTrace();
        }
    }

    @FXML
    private void doAbout(final ActionEvent event) {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Tentang Aplikasi");
        alert.setHeaderText("Aplikasi Reservasi Ruangan");
        alert.setContentText("Dikembangkan oleh Universitas Kristen Duta
Wacana\nVersi 1.0\n(c) 2025");
        alert.showAndWait();
    }

    @FXML
    private void handleGedungAction(final ActionEvent event) throws
Exception {
//            FXMLLoader loader = new
FXMLLoader(getClass().getResource("/org/ukdw/gedung-view.fxml"));
            FuckingSwitch.setShit("/org/ukdw/gedung-view.fxml");
    }

    @FXML
    private void handleUserAction(final ActionEvent event) throws
Exception {
            FuckingSwitch.setShit("/org/ukdw/user-view.fxml");
    }


    @FXML
    private void handleBookingAction(final ActionEvent event) {
        try {
            FuckingSwitch.setShit("/org/ukdw/pemesanan-view.fxml");
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    @FXML
    private void handleRuanganAction(final ActionEvent event) throws
Exception {
            FuckingSwitch.setShit("/org/ukdw/ruangan-view.fxml");
    }

    @FXML
    private void doLogout(final ActionEvent event) throws Exception{
        SessionManager.getInstance().logout();
        SessionManager.getInstance().setUsername(null);

        Stage stage = (Stage) rootPane.getScene().getWindow();
        stage.close();

        FuckingSwitch.setShit("/org/ukdw/login-view.fxml");
    }

    @FXML
    private void doLaporanPemesanan(ActionEvent event) throws Exception
{
        FuckingSwitch.setShit("/org/ukdw/laporan-pemesanan-view.fxml");
    }

}
```

UG 13

CATATAN

```
package org.week12.data;
public class Catatan {
    private int id;
    private String judul;
    private String konten;
    private String kategori;

    public static String CATATAN_SELF_DEVELOPEMENT = "Self Development";
    public static String CATATAN_BELANJA = "Belanja";
    public static String CATATAN_KHUSUS = "Khusus";
    public static String CATATAN_PERCINTAAN = "Percintaan";

    public Catatan(String judul, String konten) {
        this.judul = judul;
        this.konten = konten;
    }

    public Catatan(int id, String judul, String konten) {
        this.id = id;
        this.judul = judul;
        this.konten = konten;
    }

    public Catatan(int id, String judul, String konten, String kategori)
{
        this.id = id;
        this.judul = judul;
        this.konten = konten;
        this.kategori = kategori;
    }

    public Catatan(String judul, String konten, String kategori) {
        this.judul = judul;
        this.konten = konten;
        this.kategori = kategori;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getJudul() {
        return judul;
    }

    public void setJudul(String judul) {
        this.judul = judul;
    }

    public String getKonten() {
        return konten;
    }

    public void setKonten(String konten) {
        this.konten = konten;
    }

    public String getKategori() {
        return kategori;
    }

    public void setKategori(String kategori) {
        this.kategori = kategori;
    }
}
```

CATATAN REPOSITORY

```
package org.week12.repository;
import org.week12.data.Catatan;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class CatatanRepository implements Dao {
    Connection connection;

    public CatatanRepository(Connection connection) {
        this.connection = connection;
    }

    @Override
    public List<Catatan> getAllDataCatatan() {
        List<Catatan> catatanList = new ArrayList<>();
        String query = "SELECT * FROM catatan";
        try (PreparedStatement stmt =
connection.prepareStatement(query)) {
            ResultSet rs = stmt.executeQuery();
            while (rs.next()) {
                Catatan c = new Catatan(
                        rs.getInt("id"),
                        rs.getString("judul"),
                        rs.getString("konten"),
                        rs.getString("kategori")
                );
                catatanList.add(c);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return catatanList;
    }

    @Override
    public boolean addCatatan(Catatan catatan) {
        String query = "INSERT INTO catatan (judul, konten, kategori)
VALUES (?, ?, ?)";
        try (PreparedStatement stmt = connection.prepareStatement(query,
Statement.RETURN_GENERATED_KEYS)) {
            stmt.setString(1, catatan.getJudul());
            stmt.setString(2, catatan.getKonten());
            stmt.setString(3, catatan.getKategori());
            int affected = stmt.executeUpdate();
            if (affected > 0) {
                ResultSet rs = stmt.getGeneratedKeys();
                if (rs.next()) catatan.setId(rs.getInt(1));
                return true;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }

    @Override
    public boolean updateCatatan(Catatan oldCatatan, Catatan newCatatan)
{
        String query = "UPDATE catatan SET judul = ?, konten = ?,
kategori = ? WHERE id = ?";
        try (PreparedStatement stmt =
connection.prepareStatement(query)) {
            stmt.setString(1, newCatatan.getJudul());
            stmt.setString(2, newCatatan.getKonten());
            stmt.setString(3, newCatatan.getKategori());
            stmt.setInt(4, oldCatatan.getId());
            return stmt.executeUpdate() > 0;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }

    @Override
    public boolean deleteCatatan(Catatan catatan) {
        String query = "DELETE FROM catatan WHERE id = ?";
        try (PreparedStatement stmt =
connection.prepareStatement(query)) {
            stmt.setInt(1, catatan.getId());
            return stmt.executeUpdate() > 0;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }
}
```

DAO

```
package org.week12.repository;
import org.week12.data.Catatan;
import java.util.List;

public interface Dao {
    List<Catatan> getAllDataCatatan();
    boolean addCatatan(Catatan catatan);
    boolean updateCatatan(Catatan oldCatatan, Catatan newCatatan);
    boolean deleteCatatan(Catatan catatan);
```

```
}
```

DATABASE DRIVER

```
package org.week12.util;

import java.sql.Connection;

public interface DatabaseDriver {
   Connection getConnection();
   void closeConnection();
   void preparedSchema();

}
```

DATABASE UTIL

```
package org.week12.util;

import org.week12.data.Catatan;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class DatabaseUtil {
   private final String DB_URL = "jdbc:sqlite:catatanku.db";
   private Connection connection;

   private static volatile DatabaseUtil instance = null;

   private DatabaseUtil() {
   }

   public static DatabaseUtil getInstance() {
       if (instance == null) {
           // To make thread safe
           synchronized (DatabaseUtil.class) {
               // check again as multiple threads
               // can reach above step
               if (instance == null) {
                   instance = new DatabaseUtil();
                   instance.getConnection();
                   instance.createTable();
               }
           }
       }
       return instance;
   }

   public Connection getConnection() {
       if (connection == null) {
           try {
               connection = DriverManager.getConnection(DB_URL);
           } catch (SQLException e) {
               e.printStackTrace();
               // Handle database connection error
           }
       }
       return connection;
   }

   public void closeConnection() {
       if (connection != null) {
           try {
               connection.close();
           } catch (SQLException e) {
               e.printStackTrace();
               // Handle database connection closure error
           }
       }
   }

   /* Create database tables if they don't exist
    * */
   public void createTable() {
       String mhsTableSql = "CREATE TABLE IF NOT EXISTS catatan ("
               + "id INTEGER PRIMARY KEY AUTOINCREMENT,"
               + "judul TEXT NOT NULL,"
               + "konten TEXT NOT NULL,"
               + "kategori TEXT NOT NULL"
               + ")";
       try (Statement stmt = connection.createStatement()) {
           stmt.execute(mhsTableSql);
       } catch (SQLException e) {
           e.printStackTrace();
           // Handle table creation error
       }
   }


   public List<Catatan> getAllDataCatatan() {
       String query = "SELECT * FROM catatan";
       List<Catatan> catatanList = new ArrayList<>();
       try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
           ResultSet resultSet = preparedStatement.executeQuery();
           while (resultSet.next()) {
               int id = resultSet.getInt("id");
               String judul = resultSet.getString("judul");
               String konten = resultSet.getString("konten");
               String kategori = resultSet.getString("kategori");
               Catatan catatan = new Catatan(id, judul, konten,
kategori);
               catatanList.add(catatan);
           }
       } catch (SQLException e) {
           e.printStackTrace();
           // Handle database query error
       }
       return catatanList;
   }

   public boolean deleteCatatan(Catatan catatan) {
       String query = "DELETE FROM catatan WHERE id = ?";
       try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
           preparedStatement.setInt(1, catatan.getId());
           int rowsAffected = preparedStatement.executeUpdate();
           if (rowsAffected > 0) {
               return true;
           }
       } catch (SQLException e) {
           e.printStackTrace();
       }
       return false;
   }

   public boolean addCatatan(Catatan catatan) {
       String queryGetNextId = "SELECT seq FROM SQLITE_SEQUENCE WHERE
name = 'catatan' LIMIT 1";
       String queryInsert = "INSERT INTO catatan (judul, konten,
kategori) VALUES (?, ?, ?)";
       try {
           connection.setAutoCommit(false); // Start transaction
           try (PreparedStatement getNextIdStatement =
connection.prepareStatement(queryGetNextId);
                PreparedStatement insertStatement =
connection.prepareStatement(queryInsert)) {
               // Execute query to get the next ID
               ResultSet resultSet = getNextIdStatement.executeQuery();
               int nextId = 1; // Default value if no rows are returned
               if (resultSet.next()) {
                   nextId = resultSet.getInt("seq") + 1;
               }
               // Set parameters for insert query
               insertStatement.setString(1, catatan.getJudul());
               insertStatement.setString(2, catatan.getKonten());
               insertStatement.setString(3, catatan.getKategori());
               // Execute insert query
               int rowsAffected = insertStatement.executeUpdate();

               if (rowsAffected > 0) {
                   catatan.setId(nextId);
                   connection.commit(); // Commit transaction
                   return true;
               }
           } catch (SQLException e) {
               connection.rollback(); // Rollback transaction
               e.printStackTrace();
               // Handle database query error
           } finally {
               connection.setAutoCommit(true); // Reset auto-commit
mode
           }
       } catch (SQLException e) {
           e.printStackTrace();
       }
       return false;
   }

   public boolean updateCatatan(Catatan oldCatatan, Catatan newCatatan)
{
```

```
        String query = "UPDATE catatan SET judul = ?, konten = ?,
kategori = ? WHERE id = ?";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
            preparedStatement.setString(1, newCatatan.getJudul());
            preparedStatement.setString(2, newCatatan.getKonten());
            preparedStatement.setString(3, newCatatan.getKategori());
            preparedStatement.setInt(4, oldCatatan.getId());
            int rowsAffected = preparedStatement.executeUpdate();
            if (rowsAffected > 0) {
                return true;
            }
        } catch (SQLException e) {
            e.printStackTrace();
            // Handle database query error
        }
        return false;
    }
}
```

MARIA DB

```
package org.week12.util;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;


//ini adalah Singleton
public class MariaDBDriver implements DatabaseDriver {
    private static final String URL =
"jdbc:mariadb://localhost:3306/rplbo";
    private static final String USER = "root";
    private static final String PASSWORD = "";
    private Connection connection;

    private static volatile MariaDBDriver instance = null;

    private MariaDBDriver() {
    }

    public static MariaDBDriver getInstance() {
        if (instance == null) {
            // To make thread safe
            synchronized (MariaDBDriver.class) {
                // check again as multiple threads
                // can reach above step
                if (instance == null) {
                    instance = new MariaDBDriver();
                    instance.getConnection();
                    instance.preparedSchema();
                }
            }
        }
        return instance;
    }

    @Override
    public Connection getConnection() {
        try {
            if (connection == null || connection.isClosed()) {
                connection = DriverManager.getConnection(URL, USER,
PASSWORD);
            }
            return connection;
        } catch (SQLException e) {
            throw new RuntimeException("Failed to establish connection
to MariaDB database", e);
        }
    }

    @Override
    public void closeConnection() {
        try {
            if (connection != null && !connection.isClosed()) {
                connection.close();
            }
        } catch (SQLException e) {
            throw new RuntimeException("Failed to close connection", e);
        }
    }

    @Override
    public void preparedSchema() {
        // Create database tables if they don't exist
        // Implement this method to create tables for users, courses,
classes, and attendance records
        String mhsTableSql = "CREATE TABLE IF NOT EXISTS mahasiswa ("
                + "nim VARCHAR(20) PRIMARY KEY,"
                + "nama VARCHAR(100) NOT NULL,"
                + "nilai DOUBLE NOT NULL,"
                + "foto LONGBLOB"
                + ")";
        try (Statement stmt = connection.createStatement()) {
            stmt.execute(mhsTableSql);
        } catch (SQLException e) {
            e.printStackTrace();
            // Handle table creation error
        }
    }
}
```

SQLITE

```
package org.week12.util;

import java.sql.*;

public class SqliteDriver implements DatabaseDriver {
    private final String DB_URL = "jdbc:sqlite:catatanku.db";
    private Connection connection;

    private static volatile SqliteDriver instance = null;

    private SqliteDriver() {
    }

    public static SqliteDriver getInstance() {
        if (instance == null) {
            synchronized (SqliteDriver.class) {
                if (instance == null) {
                    instance = new SqliteDriver();
                    instance.getConnection();
                    instance.preparedSchema();
                }
            }
        }
        return instance;
    }

    public Connection getConnection() {
        if (connection == null) {
            try {
                connection = DriverManager.getConnection(DB_URL);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return connection;
    }

    public void closeConnection() {
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public void preparedSchema() {
        String mhsTableSql = "CREATE TABLE IF NOT EXISTS mahasiswa ("
                + "nim TEXT PRIMARY KEY,"
                + "nama TEXT NOT NULL,"
                + "nilai REAL NOT NULL,"
                + "foto BLOB"
                + ")";
        try (Statement stmt = connection.createStatement()) {
            stmt.execute(mhsTableSql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

SESSION MANAGER

```
package org.week12.util;

import java.io.*;
```

```
public class SessionManager implements Serializable {
    private static final long serialVersionUID = 1L;
    private static final String SESSION_FILE = "session.ser";

    private static volatile SessionManager instance;
    private boolean isLoggedIn;

    private SessionManager() {
        isLoggedIn = false;
    }

    public static SessionManager getInstance() {
        if (instance == null) {
            synchronized (SessionManager.class) {
                if (instance == null) {
                    instance = new SessionManager();
                    instance.createSessionFile();
                }
            }
        }
        return instance;
    }

    public void createSessionFile() {
        File file = new File(SESSION_FILE);
        if (!file.exists()) {
            saveSession();
        } else {
            loadSession();
        }
    }

    private void loadSession() {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(SESSION_FILE))) {
            SessionManager sessionManager = (SessionManager)
ois.readObject();
            this.isLoggedIn = sessionManager.isLoggedIn;
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error loading session: " +
e.getMessage());
        }
    }

    private void saveSession() {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(SESSION_FILE))) {
            oos.writeObject(this);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public boolean isLoggedIn() {
        return isLoggedIn;
    }

    public void login() {
        isLoggedIn = true;
        saveSession();
    }

    public void logout() {
        isLoggedIn = false;
        saveSession();
    }
}
```

ABOUT CONTROLLER

```
package org.week12;

import javafx.fxml.Initializable;

import java.net.URL;
import java.util.ResourceBundle;

public class AboutController implements Initializable {
    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {

    }
}
```

APPS

```
package org.week12;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Modality;
import javafx.stage.Stage;
import org.week12.util.SessionManager;

import java.io.IOException;

public class Apps extends Application {
    private static Stage primaryStage;

    @Override
    public void start(Stage stage) throws IOException {
        primaryStage = stage;
        primaryStage.setTitle("Title");
        if (SessionManager.getInstance().isLoggedIn()) {
            primaryStage.setScene(new
Scene(loadFXML("daftar-catatan-view")));
        } else {
            primaryStage.setScene(new Scene(loadFXML("login-view")));
        }
        primaryStage.show();
    }

    public static Stage getPrimaryStage() {
        return primaryStage;
    }

    private static Parent loadFXML(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(Apps.class
                .getResource(fxml + ".fxml"));
        return fxmlLoader.load();
    }
    public static void setRoot(String fxml, String title, boolean
isResizeable)
            throws IOException {
        primaryStage.getScene().setRoot(loadFXML(fxml));
        primaryStage.sizeToScene();
        primaryStage.setResizable(isResizeable);
        if(title !=null){
            primaryStage.setTitle(title);
        }
        primaryStage.show();
    }

    public static void openViewWithModal(String fxml, String title,
boolean isResizeable)
            throws IOException {
        Stage stage = new Stage();
        stage.setScene(new Scene(loadFXML(fxml)));
        stage.sizeToScene();
        stage.setTitle(title);
        stage.setResizable(isResizeable);
        stage.initOwner(primaryStage);
        stage.initModality(Modality.WINDOW_MODAL);
        stage.showAndWait();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

DAFTAR CATATAN

```
package org.week12;

import javafx.application.Platform;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.collections.transformation.FilteredList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import org.week12.data.Catatan;
import org.week12.repository.CatatanRepository;
import org.week12.repository.Dao;
//import org.week12.util.DatabaseUtil;
import org.week12.util.SessionManager;
```

```java
import org.week12.util.SqliteDriver;

import java.io.IOException;
import java.net.URL;
import java.sql.*;
import java.util.ResourceBundle;
import java.util.function.Predicate;

public class DaftarCatatanController implements Initializable {
    @FXML
    private TextField txtFldJudul;
    @FXML
    private TextArea txtAreaKonten;
    private FilteredList<Catatan> catatanFilteredList;
    @FXML
    private TableView<Catatan> table;
    @FXML
    private TableColumn<Catatan, String> id;
    @FXML
    private TableColumn<Catatan, String> judul;
    @FXML
    private TableColumn<Catatan, String> kategori;
    @FXML
    private ChoiceBox<String> cbKategori;
    @FXML
    public TextField searchBox;
    Catatan selectedCatatan;

    private Dao catatanRepository;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        catatanRepository = new
CatatanRepository(SqliteDriver.getInstance().getConnection());

        catatanFilteredList = new
FilteredList<>(FXCollections.observableList(FXCollections.observableArr
ayList()));
        table.setItems(catatanFilteredList);
        searchBox.textProperty().addListener(
                (observableValue, oldValue, newValue) ->
catatanFilteredList.setPredicate(createPredicate(newValue))
        );

        id.setCellValueFactory(new PropertyValueFactory<>("id"));
        judul.setCellValueFactory(new PropertyValueFactory<>("judul"));
        kategori.setCellValueFactory(new
PropertyValueFactory<>("kategori"));
        getAllData();
        table.getSelectionModel().selectedItemProperty().addListener(new
ChangeListener<Catatan>() {
            @Override
            public void changed(ObservableValue<? extends Catatan>
observableValue, Catatan course, Catatan t1) {
                if (observableValue.getValue() != null) {
                    selectedCatatan = observableValue.getValue();

txtFldJudul.setText(observableValue.getValue().getJudul());

txtAreaKonten.setText(observableValue.getValue().getKonten());

cbKategori.setValue(observableValue.getValue().getKategori());
                }
            }
        });

        //setup combo box
        cbKategori.getItems().add(Catatan.CATATAN_SELF_DEVELOPEMENT);
        cbKategori.getItems().add(Catatan.CATATAN_BELANJA);
        cbKategori.getItems().add(Catatan.CATATAN_KHUSUS);
        cbKategori.getItems().add(Catatan.CATATAN_PERCINTAAN);
        bersihkan();
    }

    private ObservableList<Catatan> getObservableList() {
        return (ObservableList<Catatan>)
catatanFilteredList.getSource();
    }

    private Predicate<Catatan> createPredicate(String searchText) {
        return catatan -> {
            if (searchText == null || searchText.isEmpty()) return true;
            return searchFindsCatatan(catatan, searchText);
        };
    }

    private boolean searchFindsCatatan(Catatan catatan, String
searchText) {
        return
(catatan.getJudul().toLowerCase().contains(searchText.toLowerCase()))
||

(catatan.getKonten().toLowerCase().contains(searchText.toLowerCase()))
||

(catatan.getKategori().toLowerCase().contains(searchText.toLowerCase())
);
    }

    private void getAllData() {
        getObservableList().clear();

getObservableList().addAll(catatanRepository.getAllDataCatatan());
    }

    private void bersihkan() {
        txtFldJudul.clear();
        txtAreaKonten.clear();
        txtFldJudul.requestFocus();
        table.getSelectionModel().clearSelection();
        selectedCatatan = null;
        cbKategori.getSelectionModel().clearSelection();
        cbKategori.getSelectionModel().select(0);
    }

    private boolean isCatatanUpdated() {
        if (selectedCatatan == null) {
            return false;
        }
        if
(!selectedCatatan.getJudul().equalsIgnoreCase(txtFldJudul.getText()) ||

!selectedCatatan.getKonten().equalsIgnoreCase(txtAreaKonten.getText())
||

!selectedCatatan.getKategori().equalsIgnoreCase(cbKategori.getSelection
Model().getSelectedItem())) {
            return true;
        } else {
            return false;
        }
    }

    @FXML
    protected void onBtnSimpanClick() {
        if (isCatatanUpdated()) {
            if (updateCatatan(selectedCatatan, new
Catatan(selectedCatatan.getId(), txtFldJudul.getText(),
txtAreaKonten.getText(),
cbKategori.getSelectionModel().getSelectedItem()))) {
                new Alert(Alert.AlertType.INFORMATION, "Catatan
Dirubah!").show();
            } else {
                new Alert(Alert.AlertType.ERROR, "Catatan gagal
Dirubah!").show();
            }
        } else {
            if (addCatatan(new Catatan(txtFldJudul.getText(),
txtAreaKonten.getText(),
cbKategori.getSelectionModel().getSelectedItem()))) {
                new Alert(Alert.AlertType.INFORMATION, "Catatan
Ditambahkan!").show();
            } else {
                new Alert(Alert.AlertType.ERROR, "Catatan gagal
Ditambahkan!").show();
            }
        }
        bersihkan();
    }

    @FXML
    protected void onBtnGrafik() throws IOException {
        Apps.openViewWithModal("pie-chart-view", "Pie Chart", false);
    }

    @FXML
    public void handleClearSearchText(ActionEvent event) {
        searchBox.setText("");
        event.consume();
    }

    @FXML
```

```
    protected void onBtnHapus() {
        if (selectedCatatan != null && deleteCatatan(selectedCatatan)) {
            Alert alert = new Alert(Alert.AlertType.INFORMATION,
"Catatan Dihapus!");
            alert.show();
            bersihkan();
        }
    }

    public boolean deleteCatatan(Catatan catatan) {
        if (catatanRepository.deleteCatatan(catatan)) {
            getObservableList().remove(catatan);
            return true;
        }
        return false;
    }

    private boolean addCatatan(Catatan catatan) {
        if (catatanRepository.addCatatan(catatan)) {
            getObservableList().add(catatan);
            return true;
        }
        return false;
    }

    private boolean updateCatatan(Catatan oldCatatan, Catatan
newCatatan) {
        if (catatanRepository.updateCatatan(oldCatatan, newCatatan)) {
            int iOldCatatan = getObservableList().indexOf(oldCatatan);
            getObservableList().set(iOldCatatan, newCatatan);
            return true;
        }
        return false;
    }

    @FXML
    protected void onActionMenuLogOut() throws IOException {
        SessionManager.getInstance().logout();
        if (!SessionManager.getInstance().isLoggedIn()) {
            Apps.setRoot("login-view", "Login", false);
        }
    }

    @FXML
    protected void onActionMenuAbout() throws IOException {
        Apps.openViewWithModal("about-view", "About Program", false);
    }
}
```

LOGIN CONTROLLER

```
package org.week12;

import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import org.week12.util.SessionManager;

import java.io.IOException;

public class LoginController {
    private static final String CORRECT_USERNAME = "admin";
    private static final String CORRECT_PASSWORD = "admin";

    @FXML private TextField txtUsername;
    @FXML private PasswordField txtPassword;

    @FXML
    protected void onKeyPressEvent(KeyEvent event) throws IOException {
        if( event.getCode() == KeyCode.ENTER ) {
            btnLoginClick();
        }
    }
    @FXML
    protected void btnLoginClick() throws IOException {
        Alert alert;
        if (txtUsername.getText().equals(CORRECT_USERNAME) &&
txtPassword.getText().equals(CORRECT_PASSWORD)) {
            alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setHeaderText("Information");
            alert.setContentText("Login success!!");
            SessionManager.getInstance().login();
            alert.showAndWait();
            Apps.setRoot("daftar-catatan-view", "",false);
        } else {
            alert = new Alert(Alert.AlertType.ERROR);
            alert.setHeaderText("Error");
            alert.setContentText("Login failed!! Please check again.");
            alert.showAndWait();
            txtUsername.requestFocus();
        }
    }
}
```

PIE CHART

```
package org.week12;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.chart.PieChart;
import java.net.URL;
import java.sql.*;
import java.util.ResourceBundle;

public class PieChartController implements Initializable {
    private final String DB_URL = "jdbc:sqlite:catatanku.db";
    private Connection connection;
    @FXML
    private PieChart pieChart;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        getConnection();
        preparedData();
    }

    public Connection getConnection() {
        if (connection == null) {
            try {
                connection = DriverManager.getConnection(DB_URL);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        return connection;
    }

    public void closeConnection() {
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    private void preparedData() {
        ObservableList<PieChart.Data> pieChartData =
FXCollections.observableArrayList();
        String query = "SELECT catatan.kategori, COUNT(*) AS
count_kategori\n" +
                "FROM catatan\n" +
                "GROUP BY catatan.kategori;";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                String mark = resultSet.getString("kategori");
                int jumlah = resultSet.getInt("count_kategori");
                pieChartData.add(new PieChart.Data(mark, jumlah));
            }
            pieChart.setData(pieChartData);
            pieChart.setTitle("Jumlah Catatan berdasarkan Kategori");
            pieChart.setClockwise(true);
            pieChart.setLabelLineLength(50);
            pieChart.setLabelsVisible(true);
            pieChart.setStartAngle(180);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

UG 14 - 15

GEDUNG

```
package org.ukdw.data;

import java.io.Serializable;

public class Gedung implements Serializable {

    private int id;
    private String nama;
    private String alamat;

    public Gedung(int id, String nama, String alamat) {
        this.id = id;
        this.nama = nama;
        this.alamat = alamat;
    }

    public void setId(int id) {
        this.id = id;
    }

    public void setNama(String nama) {
        this.nama = nama;
    }

    public void setAlamat(String alamat) {
        this.alamat = alamat;
    }

    public String getNama() {
        return nama;
    }

    public String getAlamat() {
        return alamat;
    }

    public int getId() {
        return id;
    }
}
```

PEMESANAN

```
public class Pemesanan implements Serializable {

    private SimpleIntegerProperty id;
    private SimpleStringProperty userEmail;
    private SimpleIntegerProperty idRuangan;
    private SimpleStringProperty checkInDate;
    private SimpleStringProperty checkOutDate;
    private SimpleStringProperty checkInTime;
    private SimpleStringProperty checkOutTime;
    private SimpleStringProperty namaruangan;
    private int total;

    public int getTotal() {
        return total;
    }

    public void setTotal(int total) {
        this.total = total;
    }

    /**
     * Reservation Model.
     *
     * @param userEmail    Email ID for Booking a
Reservation(Identifier).
     * @param idRuangan    Room Name.
     * @param checkInDate  Check in Date.
     * @param checkOutDate Check out Date
     * @param checkInTime  Check in time.
     * @param checkOutTime Check out time.
     */

    public Pemesanan(String namaruangan, int total) {
        this.namaruangan = new SimpleStringProperty(namaruangan);
        setTotal(total);
    }


    public Pemesanan(SimpleIntegerProperty id, SimpleStringProperty
userEmail, SimpleIntegerProperty idRuangan, SimpleStringProperty
checkInDate, SimpleStringProperty checkOutDate, SimpleStringProperty
checkInTime, SimpleStringProperty checkOutTime) {
        this.id = id;
        this.userEmail = userEmail;
        this.idRuangan = idRuangan;
        this.checkInDate = checkInDate;
        this.checkOutDate = checkOutDate;
        this.checkInTime = checkInTime;
        this.checkOutTime = checkOutTime;
        this.namaruangan = new SimpleStringProperty ((new
PemesananManager(DBConnectionManager.getConnection())).getRuanganName(t
his.id.get()) );
    }

    public int getId() {
        return id.get();
    }

    public String getNamaruangan() {
        return namaruangan.get();
    }

    public SimpleStringProperty namaruanganProperty() {
        return namaruangan;
    }

    public void setNamaruangan(String namaruangan) {
        this.namaruangan.set(namaruangan);
    }

    public SimpleIntegerProperty idProperty() {
        return id;
    }

    public void setId(int id) {
        this.id.set(id);
    }

    public String getCheckOutTime() {
        return checkOutTime.get();
    }

    public SimpleStringProperty checkOutTimeProperty() {
        return checkOutTime;
    }

    public void setCheckOutTime(String checkOutTime) {
        this.checkOutTime.set(checkOutTime);
    }

    public String getCheckInTime() {
        return checkInTime.get();
    }

    public SimpleStringProperty checkInTimeProperty() {
        return checkInTime;
    }

    public void setCheckInTime(String checkInTime) {
        this.checkInTime.set(checkInTime);
    }

    public String getCheckOutDate() {
        return checkOutDate.get();
    }

    public SimpleStringProperty checkOutDateProperty() {
        return checkOutDate;
    }

    public void setCheckOutDate(String checkOutDate) {
        this.checkOutDate.set(checkOutDate);
    }

    public String getCheckInDate() {
        return checkInDate.get();
    }

    public SimpleStringProperty checkInDateProperty() {
        return checkInDate;
    }

    public void setCheckInDate(String checkInDate) {
        this.checkInDate.set(checkInDate);
    }

    public int getIdRuangan() {
```

```
        return idRuangan.get();
    }

    public SimpleIntegerProperty idRuanganProperty() {
        return idRuangan;
    }

    public void setIdRuangan(int idRuangan) {
        this.idRuangan.set(idRuangan);
    }

    public String getUserEmail() {
        return userEmail.get();
    }

    public SimpleStringProperty userEmailProperty() {
        return userEmail;
    }

    public void setUserEmail(String userEmail) {
        this.userEmail.set(userEmail);
    }
}
```

RUANGAN

```
public class Ruangan implements Serializable {

    private int id;
    private String name;
    private int idGedung;
    private String namaGedung;

    public String getNamaGedung() {return namaGedung;}
    public void setNamaGedung(String namaGedung) {this.namaGedung =
namaGedung;}
    public Ruangan(int id, String name, int idGedung) {
        this.id = id;
        this.name = name;
        this.idGedung = idGedung;
    }

    public Ruangan(int id, String name, int idGedung, String
namaGedung) {
        this.id = id;
        this.name = name;
        this.idGedung = idGedung;
        this.namaGedung = namaGedung;
    }

    public String getName() {return name;}
    public int getIdGedung() {return idGedung;}
    public int getId() {return id;}
    public void setId(int id) {this.id = id;}
    public void setName(String name) {this.name = name;}
    public void setIdGedung(int idGedung) {this.idGedung = idGedung;}
}
```

USER

```
package org.ukdw.data;

public class User {
    private String email; //should be unique
    private String username;
    private String password;

    public User(String email, String username, String password) {
        this.setEmail(email);
        this.setUsername(username);
        this.setPassword(password);
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

GEDUNG MANAGER

```
public class GedungManager {
   private Connection connection;

   public GedungManager(Connection connection) {
       this.connection = connection;
   }

   // Add methods for course management (create, edit, delete)
   public boolean addGedung(String nama, String alamat)  {
       String query = "INSERT INTO gedung (nama, alamat) VALUES (?,
?)";
       try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
           preparedStatement.setString(1, nama);
           preparedStatement.setString(2, alamat);
           int rowsAffected = preparedStatement.executeUpdate();
           return rowsAffected > 0; // Course added successfully if
rows were inserted
       } catch (SQLException e) {
           e.printStackTrace();
           // Handle database query error
           return false;
       }
   }

   public boolean editGedung(int gedungId, String newNama, String
newAlamat) {
       String query = "UPDATE gedung SET nama = ?, alamat = ? WHERE id
= ?";
       try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
           preparedStatement.setString(1, newNama);
           preparedStatement.setString(2, newAlamat);
           preparedStatement.setInt(3, gedungId);
           int rowsAffected = preparedStatement.executeUpdate();
           return rowsAffected > 0; // Course edited successfully if
rows were affected
       } catch (SQLException e) {
           e.printStackTrace();
           // Handle database query error
           return false;
       }
   }

   public boolean deleteGedung(int gedungId) {
       String query = "DELETE FROM gedung WHERE id = ?";
       try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
           preparedStatement.setInt(1, gedungId);
           int rowsAffected = preparedStatement.executeUpdate();
           return rowsAffected > 0; // Course deleted successfully if
rows were affected
       } catch (SQLException e) {
           e.printStackTrace();
           // Handle database query error
           return false;
       }
   }

   public List<Gedung> getAllGedung() {
       List<Gedung> gedungs = new ArrayList<>();
       String query = "SELECT * FROM gedung";
       try (PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
           ResultSet resultSet = preparedStatement.executeQuery();
           while (resultSet.next()) {
               int id = resultSet.getInt("id");
               String nama = resultSet.getString("nama");
               String alamat = resultSet.getString("alamat");
               Gedung gedung = new Gedung(id, nama, alamat);
               gedungs.add(gedung);
           }
       } catch (SQLException e) {
           e.printStackTrace();
           // Handle database query error
```

```
        }
        return gedungs;
    }

    public String getIdGedung(String namagedung) {
        String sql = "select id from gedung where nama = ?";

        try(PreparedStatement stmt = connection.prepareStatement(sql)) {
            stmt.setString(1, namagedung);
            ResultSet res = stmt.executeQuery();

            return res.getString(1);
        } catch (SQLException e) {
           e.printStackTrace();
        } return "";
    }

    public String getNamaGedung(String id) {
        String sql = "select nama from gedung where id = ?";

        try(PreparedStatement stmt = connection.prepareStatement(sql)) {
            stmt.setString(1, id);
            ResultSet res = stmt.executeQuery();

            return res.getString(1);
        } catch (SQLException e) {
            e.printStackTrace();
        } return "";
    }

    public static void main(String[] args) {
        GedungManager gm = new
GedungManager(DBConnectionManager.getConnection());
        System.out.println(gm.getIdGedung("Koinonia"));
    }
}
```

VALIDATOR PEMESANAN

```
public class PemesananManager {
    private Connection connection;

    public PemesananManager(Connection connection) {
        this.connection = connection;
    }

    // Menambahkan metode untuk mengecek waktu overlap
    public boolean isOverlapping(int ruanganId, String checkinDate,
String checkoutDate, String checkinTime, String checkoutTime) {
        String query = "SELECT * FROM pemesanan WHERE ruangan_id = ?
AND (" +
                "(checkindate = ? AND checkintime < ? AND checkoutdate
> ?) OR " +
                "(checkoutdate = ? AND checkouttime > ? AND checkindate
< ?) OR " +
                "(checkindate < ? AND checkoutdate > ?) OR " +
                "(checkindate = ? AND checkoutdate = ? AND checkintime
< ? AND checkouttime > ?)" +
                ")";

        try (PreparedStatement stmt =
connection.prepareStatement(query)) {
            stmt.setInt(1, ruanganId);
            stmt.setString(2, checkinDate);
            stmt.setString(3, checkinTime);
            stmt.setString(4, checkoutDate);
            stmt.setString(5, checkoutDate);
            stmt.setString(6, checkoutTime);
            stmt.setString(7, checkinDate);
            stmt.setString(8, checkinDate);
            stmt.setString(9, checkoutDate);
            stmt.setString(10, checkoutDate);
            stmt.setString(11, checkinTime);
            stmt.setString(12, checkoutTime);

            ResultSet rs = stmt.executeQuery();

            // Jika ada hasil berarti ada pemesanan yang overlap
            return rs.next();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return false;
    }

    // Metode untuk menambahkan pemesanan
    public boolean addPemesanan(String userEmail, int idRuangan, String
checkInDate, String checkOutDate, String checkInTime, String
checkOutTime) {
        if (isOverlapping(idRuangan, checkInDate, checkOutDate,
checkInTime, checkOutTime)) {
            return false; // Jika waktu overlap, return false
        }

        String query = "INSERT INTO pemesanan (user_email, ruangan_id,
checkindate, checkoutdate, checkintime, checkouttime) VALUES (?, ?, ?,
?, ?, ?)";
        try (PreparedStatement stmt =
connection.prepareStatement(query)) {
            stmt.setString(1, userEmail);
            stmt.setInt(2, idRuangan);
            stmt.setString(3, checkInDate);
            stmt.setString(4, checkOutDate);
            stmt.setString(5, checkInTime);
            stmt.setString(6, checkOutTime);
            int result = stmt.executeUpdate();
            return result > 0;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }

    // Metode untuk mengambil ID ruangan berdasarkan nama ruangan
    public int getIdRuangan(String namaRuangan) {
        String query = "SELECT id FROM ruangan WHERE nama = ?";
        try (PreparedStatement stmt =
connection.prepareStatement(query)) {
            stmt.setString(1, namaRuangan);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return rs.getInt("id");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return 0;
    }

    // Metode lainnya (edit, delete, dll)
}


public class PemesananView implements Initializable {

    @FXML
    private Button btnBatal;

    @FXML
    private Button btnTambah;

    @FXML
    private Button btnUbah;

    @FXML
    private DatePicker checkin_date;

    @FXML
    private TextField checkin_time;

    @FXML
    private DatePicker checkout_date;

    @FXML
    private TextField checkout_time;

    @FXML
    private ChoiceBox<String> choicebox_ruangan;

    @FXML
    private TableColumn<Pemesanan, Integer> colId;

    @FXML
    private TableColumn<Pemesanan, String> colIdKelas;

    @FXML
    private TableColumn<Pemesanan, String> colIdKelas1;

    @FXML
    private TableColumn<Pemesanan, String> colIdStudent;

    @FXML
```

```
    private TextField email;

    @FXML
    private TextField searchBox;

    @FXML
    private TableView<Pemesanan> table;

    @FXML
    private Button btnDelete;

    private PemesananManager pemesananManager;
    private RuanganManager ruanganManager;

    private int oldid;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        this.ruanganManager = new
RuanganManager(DBConnectionManager.getConnection());
        this.pemesananManager = new
PemesananManager(DBConnectionManager.getConnection());

        colId.setCellValueFactory(new PropertyValueFactory<>("id"));
        colIdStudent.setCellValueFactory(new
PropertyValueFactory<>("userEmail"));
        colIdKelas.setCellValueFactory(new
PropertyValueFactory<>("namaruangan"));
        colIdKelas1.setCellValueFactory(new
PropertyValueFactory<>("checkInDate"));

        table.setItems(pemesananManager.getAllPemesanan());

        ObservableList<Ruangan> ruangans =
ruanganManager.getAllRuangan();
        ruangans.forEach(r ->
choicebox_ruangan.getItems().add(r.getName()));


table.getSelectionModel().selectedItemProperty().addListener((obs, old,
p) -> {
            if (p != null) {
                int id = p.getId();
                String get_ruangan =
pemesananManager.getRuanganName(id);
                String get_email = p.getUserEmail();
                String get_checkin_date = p.getCheckInDate();
                String get_checkout_date = p.getCheckOutDate();
                String get_checkin_time = p.getCheckInTime();
                String get_checkout_time = p.getCheckOutTime();

                DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd");

                choicebox_ruangan.setValue(get_ruangan);
                email.setText(get_email);
                checkin_date.setValue(LocalDate.parse(get_checkin_date,
formatter));

checkout_date.setValue(LocalDate.parse(get_checkout_date, formatter));
                checkin_time.setText(get_checkin_time);
                checkout_time.setText(get_checkout_time);

                this.oldid = id;
            }
        });

        searchBox.textProperty().addListener((obs, oldval, newval) ->
AmbilData(newval));
    }

    public void AmbilData(String search) {
        if (!search.isEmpty()) {
            ObservableList<Pemesanan> x =
pemesananManager.getBasedOnSearch(search);
            table.getItems().setAll(x);
        } else {

table.getItems().setAll(pemesananManager.getAllPemesanan());
        }
    }

    @FXML
    public void handleClearSearchText(ActionEvent actionEvent) {
        searchBox.setText("");
    }

    @FXML
    public void handleAddAction(ActionEvent actionEvent) {
        String get_ruangan = choicebox_ruangan.getValue().toString();
        String get_email = email.getText();
        String get_checkin_date = checkin_date.getValue().toString();
        String get_checkout_date = checkout_date.getValue().toString();
        String get_checkin_time = checkin_time.getText();
        String get_checkout_time = checkout_time.getText();

        int ruanganId = pemesananManager.getIdRuangan(get_ruangan);

        // Validasi jika waktu pemesanan tumpang tindih
        if (pemesananManager.isOverlapping(ruanganId, get_checkin_date,
get_checkout_date, get_checkin_time, get_checkout_time)) {
            showAlert(Alert.AlertType.ERROR, "Gagal", "Waktu pemesanan
tumpang tindih dengan pemesanan lainnya.");
            return;
        }

        boolean res = pemesananManager.addPemesanan(get_email,
ruanganId, get_checkin_date, get_checkout_date, get_checkin_time,
get_checkout_time);

        if (res) {
            System.out.println("Ruangan berhasil dipesan.");
            int last_id = DBConnectionManager.get_last_inserted_id();
            Pemesanan newPemesanan = new Pemesanan(
                    new SimpleIntegerProperty(last_id),
                    new SimpleStringProperty(get_email),
                    new SimpleIntegerProperty(ruanganId),
                    new SimpleStringProperty(get_checkin_date),
                    new SimpleStringProperty(get_checkout_date),
                    new SimpleStringProperty(get_checkin_time),
                    new SimpleStringProperty(get_checkout_time)
            );
            table.getItems().add(newPemesanan);
        } else {
            System.out.println("Ruangan gagal dipesan.");
        }
    }

    @FXML
    public void handleBatalAction(ActionEvent actionEvent) {
        email.clear();
        checkin_date.setValue(null);
        checkout_date.setValue(null);
        checkin_time.clear();
        checkout_time.clear();
        choicebox_ruangan.setValue(null);
        table.getSelectionModel().clearSelection();
    }

    @FXML
    public void handleEditAction(ActionEvent actionEvent) {
        String get_ruangan = choicebox_ruangan.getValue().toString();
        String get_email = email.getText();
        String get_checkin_date = checkin_date.getValue().toString();
        String get_checkout_date = checkout_date.getValue().toString();
        String get_checkin_time = checkin_time.getText();
        String get_checkout_time = checkout_time.getText();
        int ruangid = pemesananManager.getIdRuangan(get_ruangan);

        // Validasi jika waktu pemesanan tumpang tindih
        if (pemesananManager.isOverlapping(ruangid, get_checkin_date,
get_checkout_date, get_checkin_time, get_checkout_time)) {
            showAlert(Alert.AlertType.ERROR, "Gagal", "Waktu pemesanan
tumpang tindih dengan pemesanan lainnya.");
            return;
        }

        int id = oldid;

        if (pemesananManager.editPemesanan(id, get_email, ruangid,
get_checkin_date, get_checkout_date, get_checkin_time,
get_checkout_time)) {
            System.out.println("Berhasil di update");

table.getItems().setAll(pemesananManager.getAllPemesanan());
        } else {
            System.out.println("Gagal di update");
        }
    }

    @FXML
    public void handleDeleteAction(ActionEvent e) {
```

```
        if (table.getSelectionModel().getSelectedItem() != null) {
            pemesananManager.deletePemesanan(oldid);

table.getItems().setAll(pemesananManager.getAllPemesanan());
        }
    }

    private void showAlert(Alert.AlertType type, String title, String
content) {
        Alert alert = new Alert(type);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(content);
        alert.showAndWait();
    }
}
```

OBSERVER

```
package org.ukdw.observer;

public interface Observer {
    void update();
}


package org.ukdw.managers;

import org.ukdw.observer.Observer;
import java.util.ArrayList;
import java.util.List;

public class GedungManagerObservable {
    private List<Observer> observers = new ArrayList<>();

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void removeObserver(Observer observer) {
        observers.remove(observer);
    }

    public void notifyObservers() {
        for (Observer observer : observers) {
            observer.update();
        }
    }

    public boolean addGedung(String nama, String alamat) {
        // Logic for adding building
        notifyObservers();  // Notify all observers
        return true;
    }
}

package org.ukdw.view;

import org.ukdw.observer.Observer;
import org.ukdw.managers.GedungManagerObservable;

public class GedungView implements Observer {
    private GedungManagerObservable gedungManager;

    public GedungView(GedungManagerObservable gedungManager) {
        this.gedungManager = gedungManager;
        gedungManager.addObserver(this);  // Register this view as an
observer
    }

    @Override
    public void update() {
        // Handle data update (refresh view, for example)
        System.out.println("Data Gedung telah diperbarui.");
    }
}
```

COR

```
package org.ukdw.chain;

public abstract class Handler {
    protected Handler nextHandler;

    public void setNextHandler(Handler nextHandler) {
        this.nextHandler = nextHandler;
    }

    public abstract void handleRequest(String request);
}
package org.ukdw.chain;

public class GedungHandler extends Handler {
    @Override
    public void handleRequest(String request) {
        if ("addGedung".equals(request)) {
            System.out.println("Menambahkan gedung...");
            // Logic for adding building
        } else if (nextHandler != null) {
            nextHandler.handleRequest(request);  // Forward request to
next handler
        }
    }
}

public class RuanganHandler extends Handler {
    @Override
    public void handleRequest(String request) {
        if ("addRuangan".equals(request)) {
            System.out.println("Menambahkan ruangan...");
            // Logic for adding room
        } else if (nextHandler != null) {
            nextHandler.handleRequest(request);  // Forward request to
next handler
        }
    }
}
package org.ukdw.chain;

public class Client {
    public static void main(String[] args) {
        Handler gedungHandler = new GedungHandler();
        Handler ruanganHandler = new RuanganHandler();

        gedungHandler.setNextHandler(ruanganHandler);  // Setting up
the chain: GedungHandler → RuanganHandler

        // Sending request to the first handler in the chain
        gedungHandler.handleRequest("addGedung");
        gedungHandler.handleRequest("addRuangan");
    }
}
```