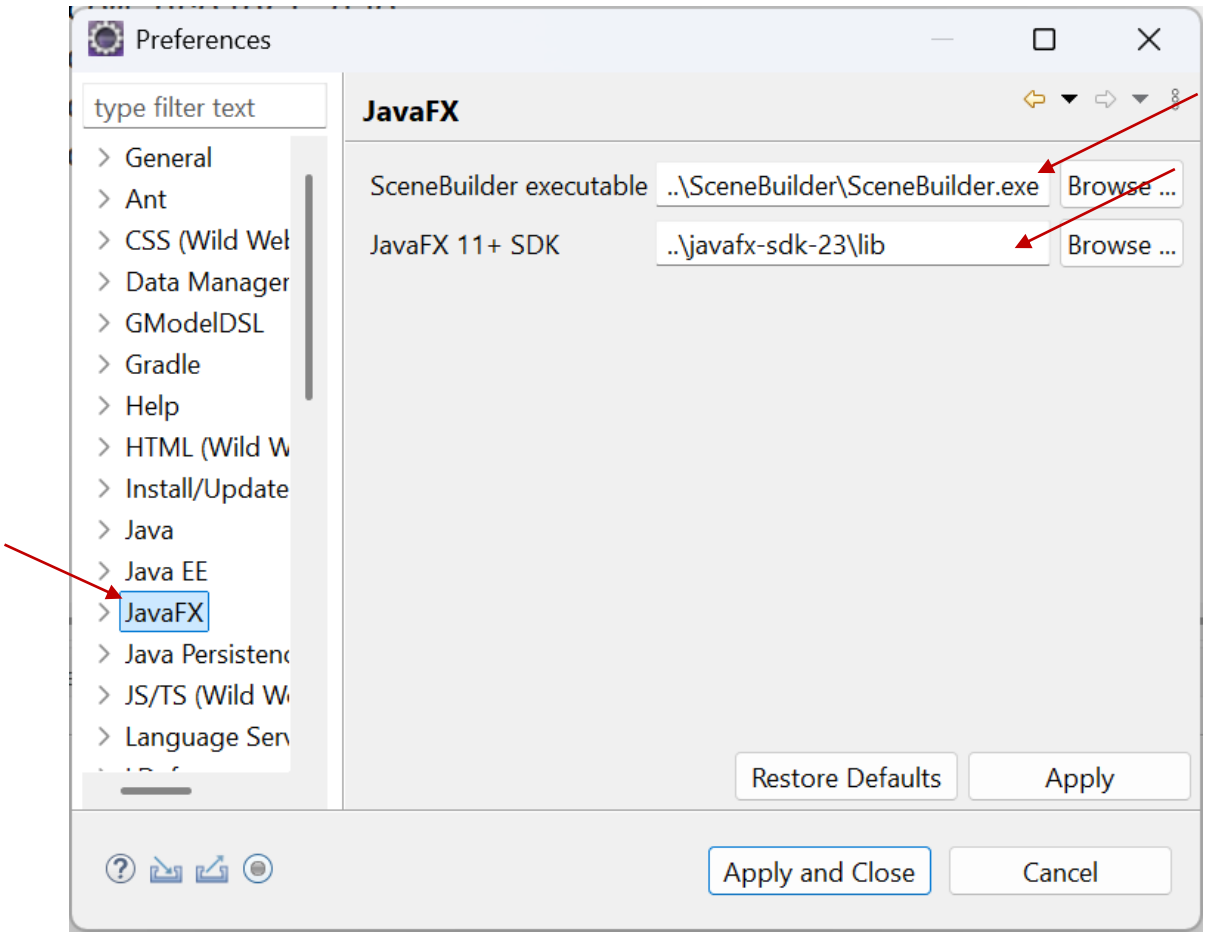
APLIKACJA JDBC

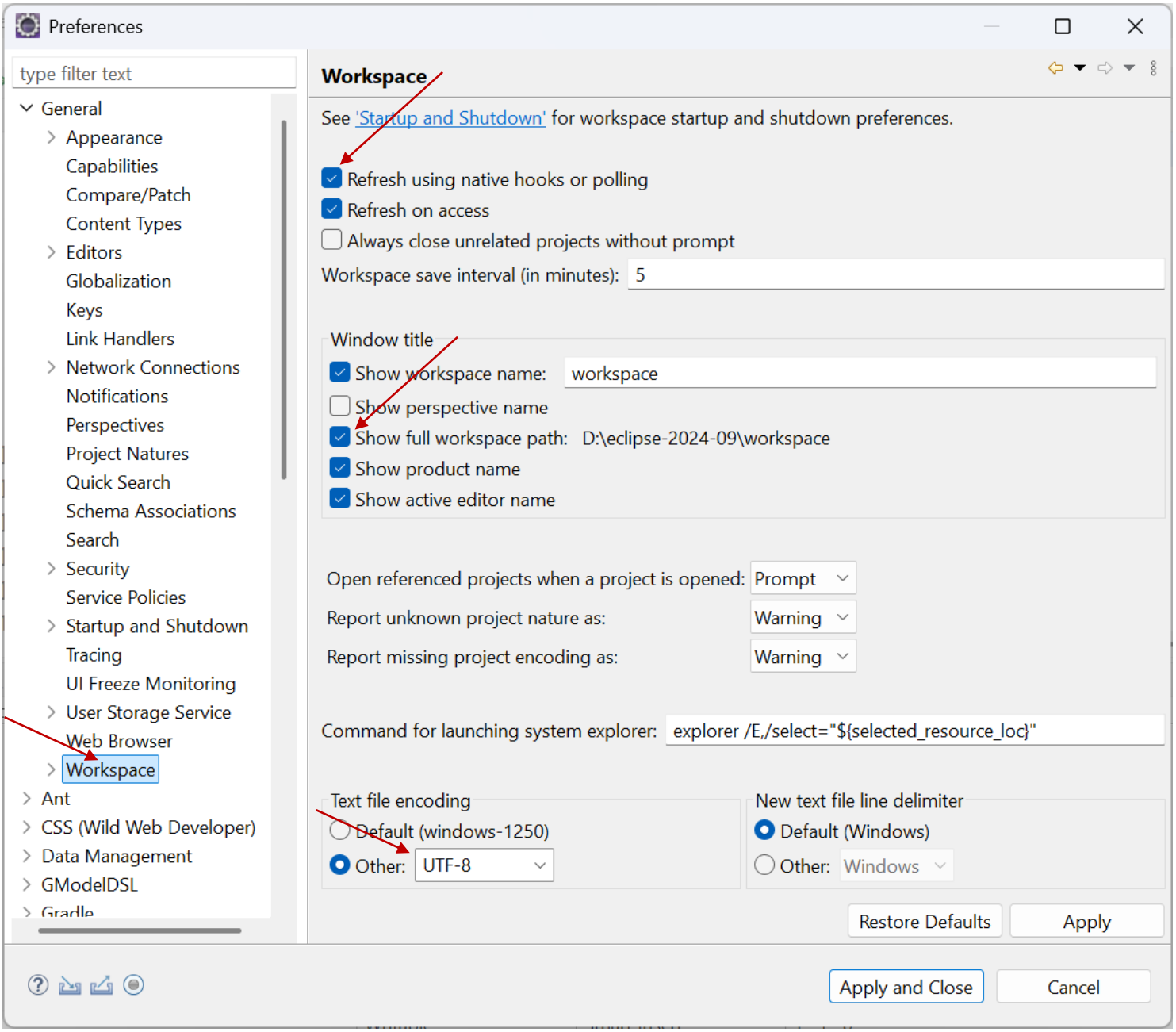
1. INSTALACJA

Najnowszą wersję oprogramowania wykorzystywanego na zajęciach laboratoryjnych (Eclipse 2024-09, JavaFX SDK 23, Scene Builder) można ściągnąć z

https://utpedupl-my.sharepoint.com/:u:/g/personal/dszczeg_o365_pbs_edu_pl/EfcqMZaTR9pDsIW0xeVeWUIBwE6joSRvknnYAQnTnNQ7vw?e=ERhp8y

Po uruchomieniu Eclipse'a z menu wybierz *Window -> Preferences* i sprawdź zaznaczone poniżej ustawienia.





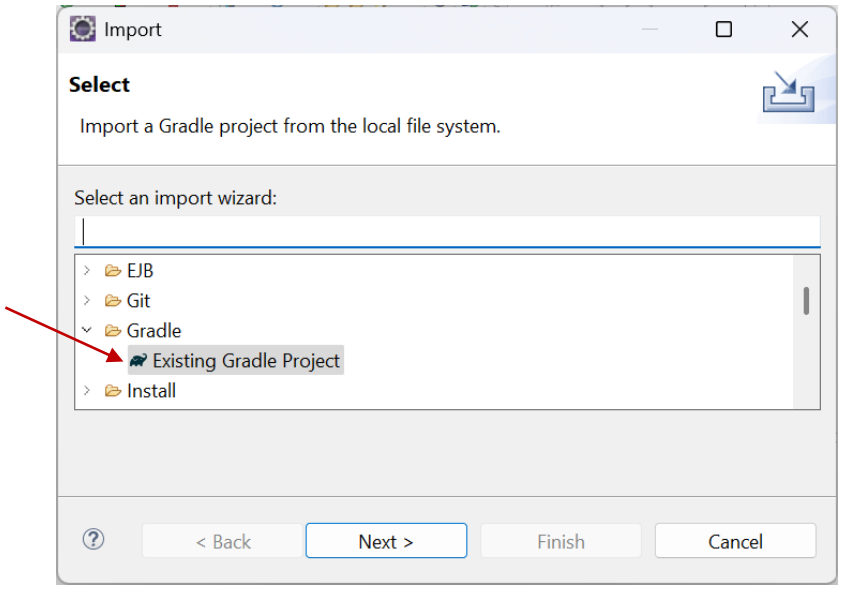
2. UTWORZENIE PROJEKTU

2.1. Ściągnij udostępnioną paczkę z projektem startowym z

https://utpedupl-my.sharepoint.com/:u:/g/personal/dszczeg_o365_pbs_edu_pl/EaqInisFpVNGoBfVeENWzCYBgtwknJWW89L2bTE9AS2L2g?e=HoGOir

Archiwum pobranego projektu rozpakuj bezpośrednio do katalogu *workspace* przechowującego projekty Eclipse'a (upewnij się, że podczas rozpakowywania nie doszło do zagnieżdżenia katalogów z projektem, sprawdź czy nie powstał podkatalog *project-jfx-client* wewnątrz katalogu o tej samej nazwie). Po uruchomieniu środowiska ścieżka do katalogu z projektami powinna być widoczna w górnej części okna, w jego pasku tytułu (o ile zaznaczyłeś wcześniej w ustawieniach opcję *Show full workspace path*). Możesz też sprawdzić lokalizację wybierając z menu *File -> Switch Workspace -> Other*.

Następnie w środowisku Eclipse wybierz z menu *File -> Import... -> Gradle / Existing Gradle Project*, wskaż główny katalog rozpakowanego projektu i naciśnij *Finish*.



2.2. Poniżej została przedstawiona zawartości pliku budującego *build.gradle*.

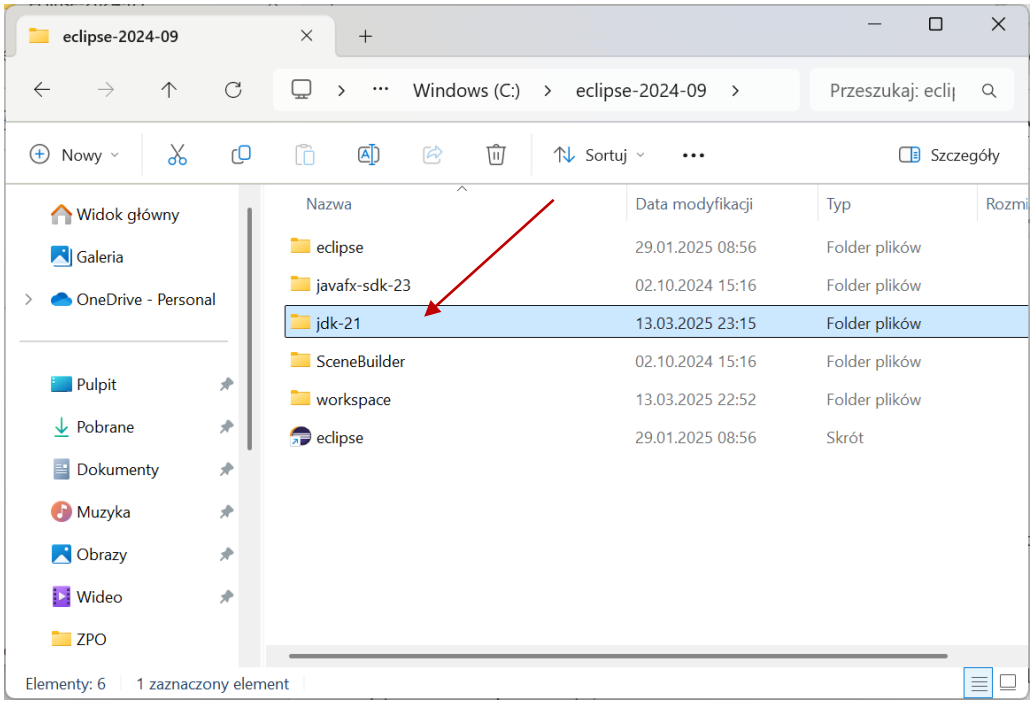
```
plugins {
        id 'application'
        id 'java'
        id 'org.openjfx.javafxplugin' version '0.1.0'
        id 'org.beryx.jlink' version '3.1.1'
}

group = 'com.project'
version = '1.0'

repositories {
    mavenCentral()
}

dependencies {
        implementation group: 'org.hsqldb', name: 'hsqldb', version: '2.7.4'
        implementation group: 'com.zaxxer', name: 'HikariCP', version: '6.2.1'
        implementation group: 'ch.qos.logback', name: 'logback-classic', version: '1.5.17'
        implementation group: 'ch.qos.logback', name: 'logback-core', version: '1.5.17'
        implementation group: 'org.slf4j', name: 'slf4j-api', version: '2.0.17'
}

java {
    sourceCompatibility = '21'
    modularity.inferModulePath = true //do automatycznego ustawiania 'module path', wymaga pliku 'module-info.java'
}

compileJava {
    options.encoding = 'UTF-8'
}

compileTestJava{
    options.encoding = 'UTF-8'
}

javafx {
    version = '23.0.2'
    modules = ['javafx.controls', 'javafx.fxml', 'javafx.base', 'javafx.graphics']
}

application {
    mainModule = 'project.jfx.client' // nazwa, identyfikator modułu w pliku 'module-info.java'
    mainClass = 'com.project.app.ProjectClientApplication' //klasa uruchomieniowa z metodą 'main' poprzedzona
                                                           //nazwą pakietu
}
```

```
jlink {
    javaHome = "$projectDir/../../jdk-21" //ścieżka do JDK, zmień jeśli trzeba!
    launcher {
        name = "$project.name" // nazwa głównego katalogu projektu, będzie również nazwą pliku EXE
                               // uruchamiającego aplikację
    }
}

eclipse.classpath.file {
    whenMerged {
        entries.findAll {
              it.properties.kind.equals('lib')
        }.each {
              it.entryAttributes['module'] = 'true'
         }
    }
}
```

2.3. Plik budujący *build.gradle* wymaga dodatkowego pakietu JDK, wersję 21 można ściągnąć z

https://utpedupl-my.sharepoint.com/:u:/g/personal/dszczeg_o365_pbs_edu_pl/EVSojVJ6uahFt8q9qIPnOqkBYoODaRDXrIGvFDdEzXzANg?e=6fDvOV

Pakiet JDK rozpakuj bezpośrednio do katalogu głównego z udostępnionym oprogramowaniem (zgodnie z poniższym obrazkiem). Alternatywnie, możesz umieścić JDK w innej lokalizacji, jednak w takim przypadku należy wskazać jego ścieżkę w pliku *build.gradle*, używając ścieżki względnej lub bezwzględnej określonej w parametrze *javaHome* w sekcji *jlink*.



2.4. Jeżeli zmodyfikowałeś plik *build.gradle* to kliknij prawym przyciskiem myszki na głównej ikonce projektu i wybierz *Gradle -> Refresh Gradle Project*.

3. PODŁĄCZENIE MECHANIZMU REJESTRACJI

3.1. W podkatalogu projektu *src\main\resources* znajduje się plik konfiguracyjny mechanizmu rejestracji o nazwie *logback.xml* (jest to domyślna nazwa i lokalizacja dla tego pliku) z przedstawioną poniżej zawartością.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration debug="true">
   <property name="LOG_FILE" value="project-jfx-client" />
   <property name="LOG_DIR" value="logs" />
   <property name="LOG_ARCHIVE" value="${LOG_DIR}/archive" />
```

```
  <!-- Send messages to System.out -->
  <appender name="STDOUT"
     class="ch.qos.logback.core.ConsoleAppender">
     <encoder>
        <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36}.%M\(%line\) - %msg%n</pattern>
     </encoder>
  </appender>

  <!-- Save messages to a file -->
  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
     <file>${LOG_DIR}/${LOG_FILE}.log</file>
     <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <!-- daily rollover -->
        <fileNamePattern>${LOG_ARCHIVE}/%d{yyyy-MM-dd}${LOG_FILE}.log.zip
        </fileNamePattern>
        <!-- keep 30 days' worth of history capped at 30MB total size -->
        <maxHistory>30</maxHistory>
        <totalSizeCap>30MB</totalSizeCap>
     </rollingPolicy>
     <encoder>
        <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36}.%M\(%line\) - %msg%n</pattern>
     </encoder>
  </appender>

  <!-- Dla pakietu 'com.project' i wszystkich jego podpakietow, zmien jeśli trzeba -->
  <logger name="com.project" level="INFO" additivity="false">
     <appender-ref ref="STDOUT" />
     <appender-ref ref="FILE" />
  </logger>

  <!-- By default, the level of the root level is set to INFO -->
  <root level="INFO">
     <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

3.2. Zamiast korzystać z *System.out.println(...);* można teraz używać mechanizmu rejestracji, który oprócz standardowego drukowania komunikatów w konsoli będzie zapisywał również ich zawartość w plikach podkatalogu *logs*, a także automatycznie je archiwizował. Pamiętaj, że we wszystkich klasach, które mają korzystać z mechanizmu rejestracji trzeba tworzyć zmienną za pomocą statycznej metody *LoggerFactory.getLogger* przekazując w jej parametrze odpowiednią klasę. Poniżej przedstawione zostały przykłady prezentujące korzystanie z mechanizmu rejestracji.

```
package ...
...
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
...
public class JakasKlasa {
   private static final Logger logger = LoggerFactory.getLogger(JakasKlasa.class);
...

logger.info("Uruchamianie programu ...");
...
logger.info("Wersja aplikacji: {}", 1.9);

logger.warn("Uaktualnij aplikację. Najnowsza dostępna wersja: {}", 2.0);

...
} catch (SQLException e) {
   logger.error("Błąd podczas zapisywania projektu!", e);
   ...
   int kodBledu = 7;
   logger.error("Błąd podczas zapisywania projektu (kod błędu: {})!", kodBledu, e);
}
```

5. IMPLEMENTACJA APLIKACJI

5.1. Utwórz pakiet *com.project.model* i zdefiniuj w nim klasę, która będzie odwzorowaniem bazodanowej tabeli *projekt*.
W klasie wystarczy zdefiniować same zmienne, natomiast konstruktory oraz tzw. gettery i settery można wygenerować automatycznie.

```
package com.project.model;

import java.time.LocalDate;
import java.time.LocalDateTime;

public class Projekt {
   private Integer projektId;
   private String nazwa;
   private String opis;
   private LocalDateTime dataCzasUtworzenia;
   private LocalDate dataOddania;

 /* TODO Wygeneruj dla powyższych zmiennych akcesory i mutatory (Source -> Generate Getters and Setters).
  * Dodaj również bezparametrowy konstruktor oraz drugi konstruktor uwzględniający wszystkie powyższe
  * zmienne, a także trzeci pomijający pole projektId.
  */
}
```

5.2. Utwórz pakiet *com.project.datasource* i zdefiniuj w nim klasę *DataSource*, która będzie zawierała mechanizm buforujący połączenia z bazą danych. Klasa jest Singletonem, który pozwala na utworzenie tylko jednej instancji klasy *HikariDataSource*. W klasie należy zdefiniować URL do bazy danych oraz hasło i nazwę użytkownika.

```
package com.project.datasource;

import java.sql.Connection;
import java.sql.SQLException;
import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;

public class DataSource {

   private final static String DB_DIR = "db";
   private final static String DB_NAME = "projekty";
   private final static String DB_USERNAME = "admin";
   private final static String DB_USER_PASSWORD = "admin";
   /*
   sql.syntax_pgs - this property, when set true, enables support for TEXT and SERIAL types.
    It also enables NEXTVAL, CURRVAL and LASTVAL syntax and also allow compatibility with some other aspects of this dialect.
   hsqldb.write_delay - If the property is true, the default WRITE DELAY property of the database is used, which is 500 ms.
    If the property is false, the WRITE DELAY is set to 0 seconds.
   */
   private final static String HSQL_ADDITIONAL_PARAMS = ";hsqldb.write_delay=false;sql.syntax_pgs=true";
   private final static String DB_URL = String.format("jdbc:hsqldb:file:%s/%s%s", DB_DIR, DB_NAME,
                                                                                  HSQL_ADDITIONAL_PARAMS);

   private final static HikariDataSource ds;

   static {
        HikariConfig config = new HikariConfig();
        config.setJdbcUrl(DB_URL);
        config.setUsername(DB_USERNAME);
        config.setPassword(DB_USER_PASSWORD);
        config.setMaximumPoolSize(1);
        ds = new HikariDataSource(config);
   }

   private DataSource() {}

   public static Connection getConnection() throws SQLException {
        return ds.getConnection();
   }

}
```

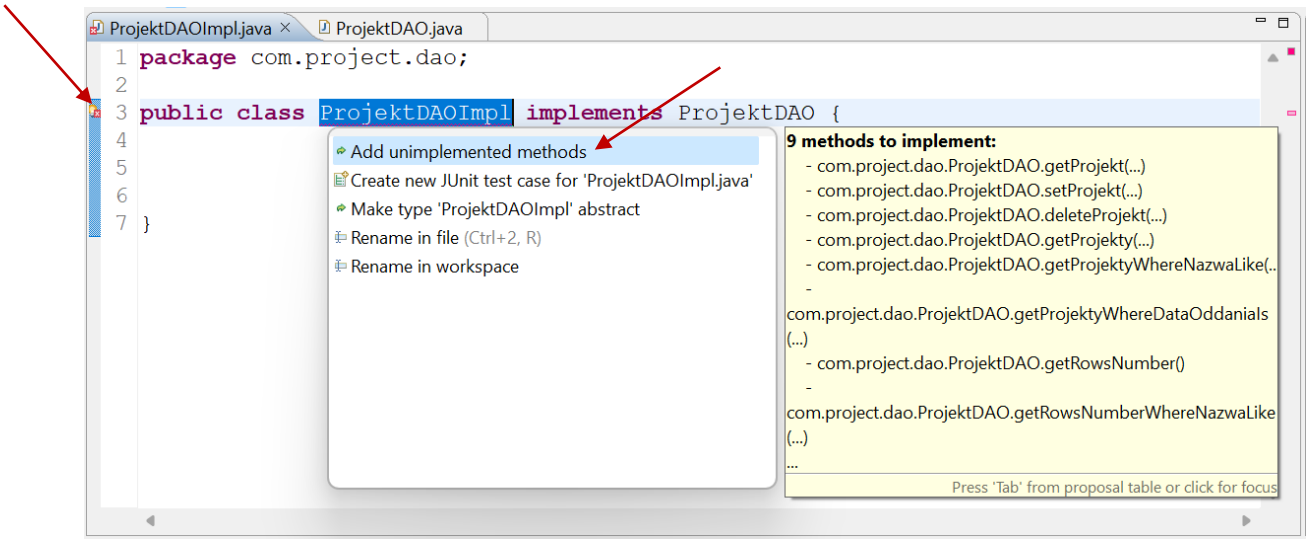
5.3. W pakiecie *com.project.datasource* dodaj kolejną klasę *DbInitializer*, której zadaniem będzie tworzenie bazy danych podczas uruchamiania aplikacji. Zawarte w tej klasie polecenia SQL gwarantują, że tabele, klucze i indeksy będą utworzone tylko gdy nie zdefiniowano wcześniej takiej struktury. Zwróć też uwagę, że polecenia są realizowane w ramach transakcji bazodanowej, więc zawsze wykonane zostaną wszystkie lub żadne z nich, w przypadku wystąpienia jakiegokolwiek błędu.

```
package com.project.datasource;

import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class DbInitializer {
	private static final Logger logger = LoggerFactory.getLogger(DbInitializer.class);
	private static final String[] queries = {
		"""
		CREATE TABLE IF NOT EXISTS projekt(
		 projekt_id SERIAL, --SERIAL lub INTEGER GENERATED BY DEFAULT AS IDENTITY(START WITH 1, INCREMENT BY 1)
		 nazwa VARCHAR(50) NOT NULL,
		 opis VARCHAR(1000),
		 dataczas_utworzenia TIMESTAMP DEFAULT now(),
		 data_oddania DATE,
		 CONSTRAINT projekt_pk PRIMARY KEY (projekt_id)
		);
		CREATE TABLE IF NOT EXISTS zadanie(
		 zadanie_id SERIAL, --SERIAL lub INTEGER GENERATED BY DEFAULT AS IDENTITY(START WITH 1, INCREMENT BY 1)
		 nazwa VARCHAR(50) NOT NULL,
		 opis VARCHAR(1000),
		 kolejnosc INTEGER,
		 dataczas_utworzenia TIMESTAMP DEFAULT now(),
		 projekt_id INTEGER NOT NULL,
		 CONSTRAINT zadanie_pk PRIMARY KEY (zadanie_id)
		);
		""",
		"""
		CREATE INDEX IF NOT EXISTS projekt_nazwa_idx ON projekt(nazwa);
		CREATE INDEX IF NOT EXISTS zadanie_nazwa_idx ON zadanie(nazwa);
		ALTER TABLE zadanie ADD CONSTRAINT IF NOT EXISTS zadanie_projekt_fk FOREIGN KEY (projekt_id)
			REFERENCES projekt (projekt_id) ON DELETE CASCADE;
		ALTER TABLE zadanie ADD CONSTRAINT IF NOT EXISTS unique_kolejnosc UNIQUE (kolejnosc, projekt_id);
		"""
	};

	private DbInitializer() {}

	public static void init() {
		try (Connection conection = DataSource.getConnection()) {
			boolean initialAutocommit = conection.getAutoCommit();
			conection.setAutoCommit(false);
			try (Statement stmt = conection.createStatement()) {
				for (int i = 0; i < queries.length; i++) {
					stmt.executeUpdate(queries[i]);
					logger.info("QUERY {}:\n{}", i + 1, queries[i]);
				}
				conection.commit();
			} catch (SQLException e) {
				conection.rollback();
				throw new RuntimeException(e);
			} finally {
				if (initialAutocommit)
					conection.setAutoCommit(true);
			}
		} catch (SQLException e) {
			throw new RuntimeException(e);
		}
	}

}
```

5.4. Utwórz pakiet *com.project.dao* i zdefiniuj w nim interfejs *ProjektDAO*, który będzie zawierał definicje wszystkich metod pobierających i modyfikujących dane projektów przechowywanych w bazie. Zadaniem interfejsu jest hermetyzacja konkretnych implementacji. Dzięki temu, gdy dokonane zostaną jakiekolwiek zmiany w sposobie pozyskiwania danych, pozostała część aplikacji nie będzie wymagać modyfikacji.

```java
package com.project.dao;

import java.util.List;
import com.project.model.Projekt;
import java.time.LocalDate;

public interface ProjektDAO {

	Projekt getProjekt(Integer projektId);

	void setProjekt(Projekt projekt);

	void deleteProjekt(Integer projektId);

	List<Projekt> getProjekty(Integer offset, Integer limit);

	List<Projekt> getProjektyWhereNazwaLike(String nazwa, Integer offset, Integer limit);

	List<Projekt> getProjektyWhereDataOddaniaIs(LocalDate dataOddania, Integer offset, Integer limit);

	int getRowsNumber();

	int getRowsNumberWhereNazwaLike(String nazwa);

	int getRowsNumberWhereDataOddaniaIs(LocalDate dataOddania);
}
```

5.5. Dodaj klasę *ProjektDAOImpl*, następnie po jej nazwie dopisz *implements ProjektDAO*. Aby dodać szkielety metod wymagających implementacji kliknij na znacznik błędu lewym przyciskiem myszki lub najedź kursorem na podkreślony fragment i wybierz *Add unimplemented methods*.

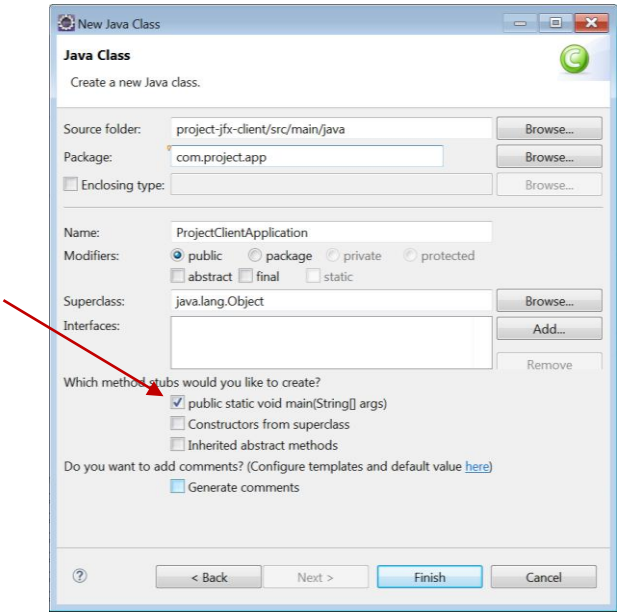


5.6. Poniżej zostały przedstawione przykładowe implementacje dwóch metod klasy *ProjektDAOImpl*. Na ich podstawie zaimplementuj pozostałe metody (pamiętaj, że nazwy pól i tabel bazy danych zawarte są w klasie *DbInitializer*).

```java
package com.project.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import com.project.datasource.DataSource;
import com.project.model.Projekt;

public class ProjektDAOImpl implements ProjektDAO {
```

```
    //...

    @Override
    public void setProjekt(Projekt projekt){
        boolean isInsert = projekt.getProjektId() == null;
        String query = isInsert ?
            "INSERT INTO projekt(nazwa, opis, dataczas_utworzenia, data_oddania) VALUES (?, ?, ?, ?)"
          : "UPDATE projekt SET nazwa = ?, opis = ?, dataczas_utworzenia = ?, data_oddania = ?"
                + " WHERE projekt_id = ?";
        try (Connection connect = DataSource.getConnection();
            PreparedStatement prepStmt = connect.prepareStatement(query, Statement.RETURN_GENERATED_KEYS)) {
            //Wstawianie do zapytania odpowiednich wartości w miejsce znaków '?'
            //Uwaga! Indeksowanie znaków '?' zaczyna się od 1!
            prepStmt.setString(1, projekt.getNazwa());
            prepStmt.setString(2, projekt.getOpis());
            if(projekt.getDataCzasUtworzenia() == null)
                projekt.setDataCzasUtworzenia(LocalDateTime.now());
            prepStmt.setObject(3,projekt.getDataCzasUtworzenia());
            prepStmt.setObject(4, projekt.getDataOddania());
            if(!isInsert) prepStmt.setInt(5, projekt.getProjektId());
            //Wysyłanie zapytania i pobieranie danych
            int liczbaDodanychWierszy = prepStmt.executeUpdate();
            //Pobieranie kluczy głównych, tylko dla nowo utworzonych projektów
            if (isInsert && liczbaDodanychWierszy > 0) {
                ResultSet keys = prepStmt.getGeneratedKeys();
                if (keys.next()) {
                    projekt.setProjektId(keys.getInt(1));
                }
                keys.close();
            }
        }catch(SQLException e) {
            throw new RuntimeException(e);
        }
    }

    @Override
    public List<Projekt> getProjekty(Integer offset, Integer limit){
        List<Projekt> projekty = new ArrayList<>();

        String query = "SELECT * FROM projekt ORDER BY dataczas_utworzenia DESC"
              + (offset != null ? " OFFSET ?" : "")
              + (limit != null ? " LIMIT ?" : "");

        try (Connection connect = DataSource.getConnection();
              PreparedStatement preparedStmt = connect.prepareStatement(query)) {
            int i = 1;
            if (offset != null) {
                preparedStmt.setInt(i, offset);
                i += 1;
            }
            if (limit != null) {
                preparedStmt.setInt(i, limit);
            }

            try (ResultSet rs = preparedStmt.executeQuery()) {
                while (rs.next()) {
                    Projekt projekt = new Projekt();
                    projekt.setProjektId(rs.getInt("projekt_id"));
                    projekt.setNazwa(rs.getString("nazwa"));
                    projekt.setOpis(rs.getString("opis"));
                    projekt.setDataCzasUtworzenia(rs.getObject("dataczas_utworzenia", LocalDateTime.class));
                    projekt.setDataOddania(rs.getObject("data_oddania", LocalDate.class));
                    projekty.add(projekt);
                }
            }
        }catch(SQLException e) {
            throw new RuntimeException(e);
        }
        return projekty;
    }

    //...


}
```

5.7. W pakiecie *com.project.app* utwórz klasę *ProjectClientApplication* zawierającą metodę *main*.



Opcjonalnie można też zweryfikować poprawność utworzonych metod bazodanowych. Poniżej zostały przedstawione przykłady dodawania nowego projektu oraz pobierania danych dla wskazanego identyfikatora. Przed uruchomieniem aplikacji należy sprawdzić czy w pliku *db/projekty.script* nie zostało zdefiniowane opóźnienie zapisu danych. Powinien on zawierać wpis `SET FILES WRITE DELAY 0` , a jeżeli jest zdefiniowany jakiś czas opóźnienia to trzeba zmienić na zero i zapisać zmodyfikowany plik.

Aby uruchomić aplikację kliknij prawym przyciskiem myszki wewnątrz okna z kodem źródłowym klasy *ProjectClientApplication* i wybierz *Run As -> Java Application*. A jeżeli w projekcie dodano już plik *module-info.java* to trzeba ją uruchomić za pomocą gradle'owskiego zadania *run* (patrz p. 6.5). Przejdź do widoku *Gradle Task* (jeżeli widok nie jest wyświetlany to z menu wybierz *Window -> Show View -> Other*, a następnie zaznacz *Gradle Tasks*), rozwiń węzeł *application* i dwukrotnie kliknij lewym przyciskiem myszki na *run*.

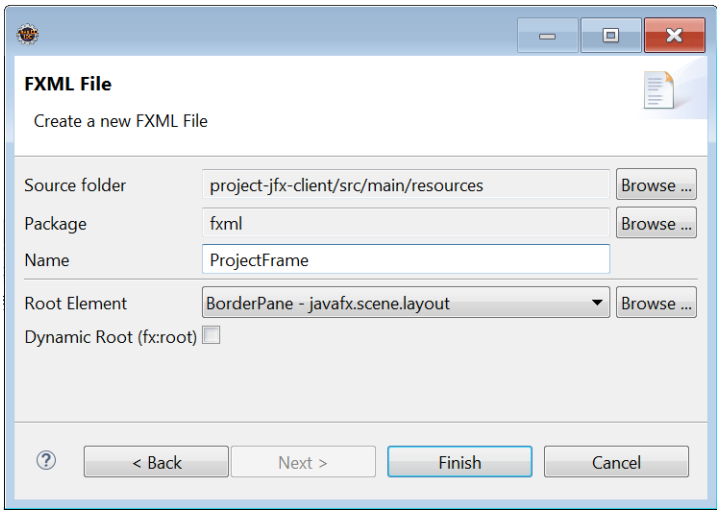
```
package com.project.app;

import java.time.LocalDate;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.project.dao.ProjektDAO;
import com.project.dao.ProjektDAOImpl;
import com.project.model.Projekt;

public class ProjectClientApplication {

   private static final Logger logger = LoggerFactory.getLogger(ProjectClientApplication.class);

   public static void main(String[] args) {

      DbInitializer.init();

      ProjektDAO projektDAO = new ProjektDAOImpl();
      Projekt projekt = new Projekt("Projekt testowy", "Opis testowy", LocalDate.of(2020, 06, 22));

      try {
         projektDAO.setProjekt(projekt);
         logger.info("Id utworzonego projektu: {}", projekt.getProjektId());


         Integer projektId = projekt.getProjektId();
         Projekt projekt2 = projektDAO.getProjekt(projektId);
         logger.info("Pobrany projekt - Id: {}, nazwa: {}, opis: {}",
                              projekt2.getProjektId(), projekt2.getNazwa(), projekt2.getOpis());

         // TODO SPRAWDZIĆ POZOSTAŁE METODY

      } catch (RuntimeException e) {
         logger.error("Błąd operacji bazodanowej!", e);
      }
   }
}
```

6. GRAFICZNY INTERFEJS UŻYTKOWNIKA

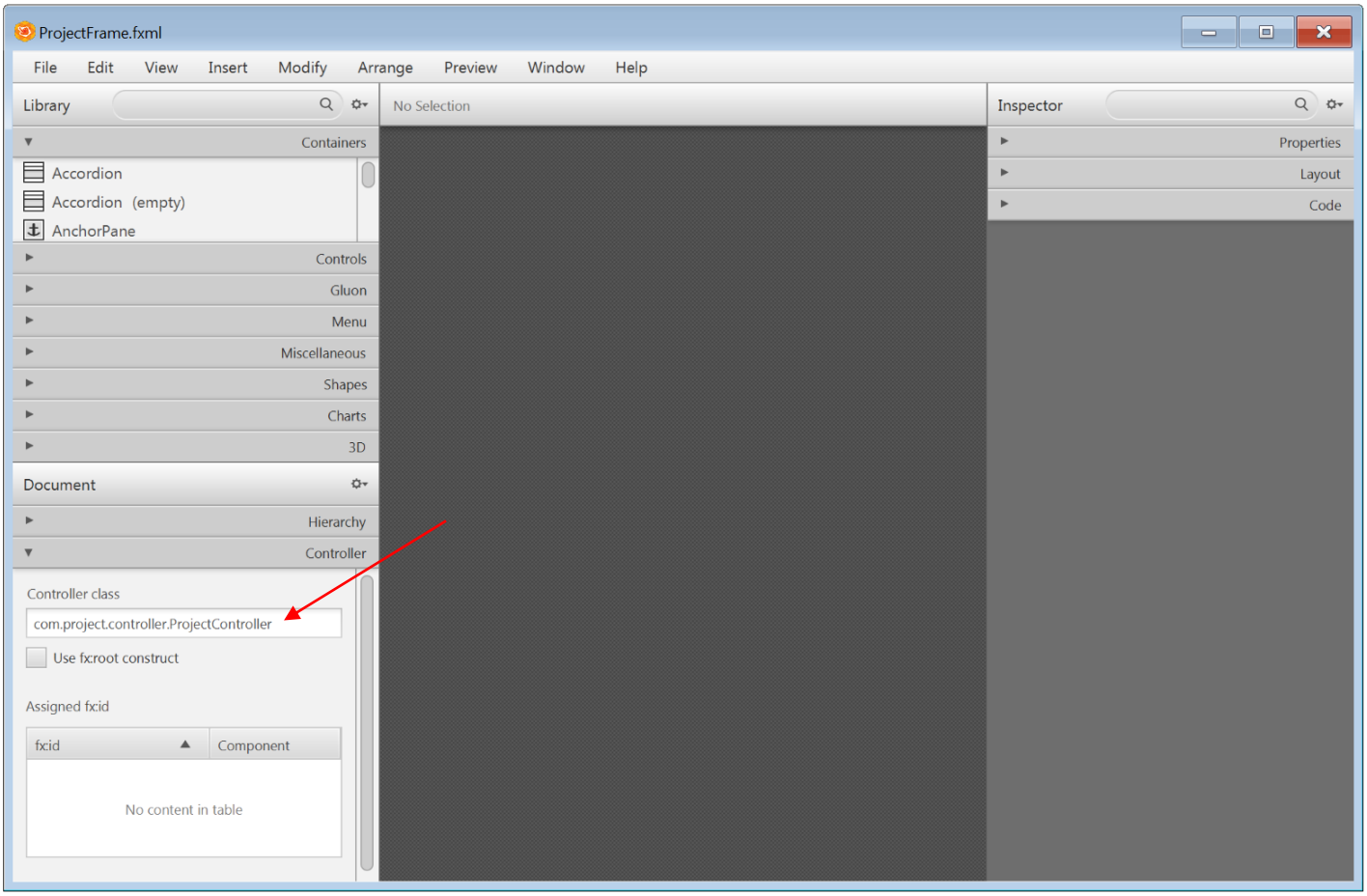
6.1. W katalogu *src\main\resources* utwórz podkatalog *css*, następnie dodaj do niego plik *application.css* z przedstawioną poniżej zawartością.

```
.table-column {
  -fx-alignment: CENTER_LEFT;
}

.table-row-cell {
    -fx-background-color: #F5F5F5;
    -fx-background-insets: 0.0, 0.0 0.0 1.0 0.0;
    -fx-text-fill: black;
    -fx-border-width: 1.0 1.0 0.0 0.0;
    -fx-border-color: #DBD7D2;
    -fx-table-cell-border-color: transparent;
}

.table-row-cell:odd {
       -fx-background-color: #EAE9E8;
}

.table-view:focused .table-row-cell:filled:focused:selected {
    -fx-background-color: -fx-focus-color, -fx-cell-focus-inner-border, -fx-selection-bar;
    -fx-background-insets: 0.0, 1.0, 2.0;
    -fx-background: -fx-accent;
    -fx-text-fill: -fx-selection-bar-text;
}
```

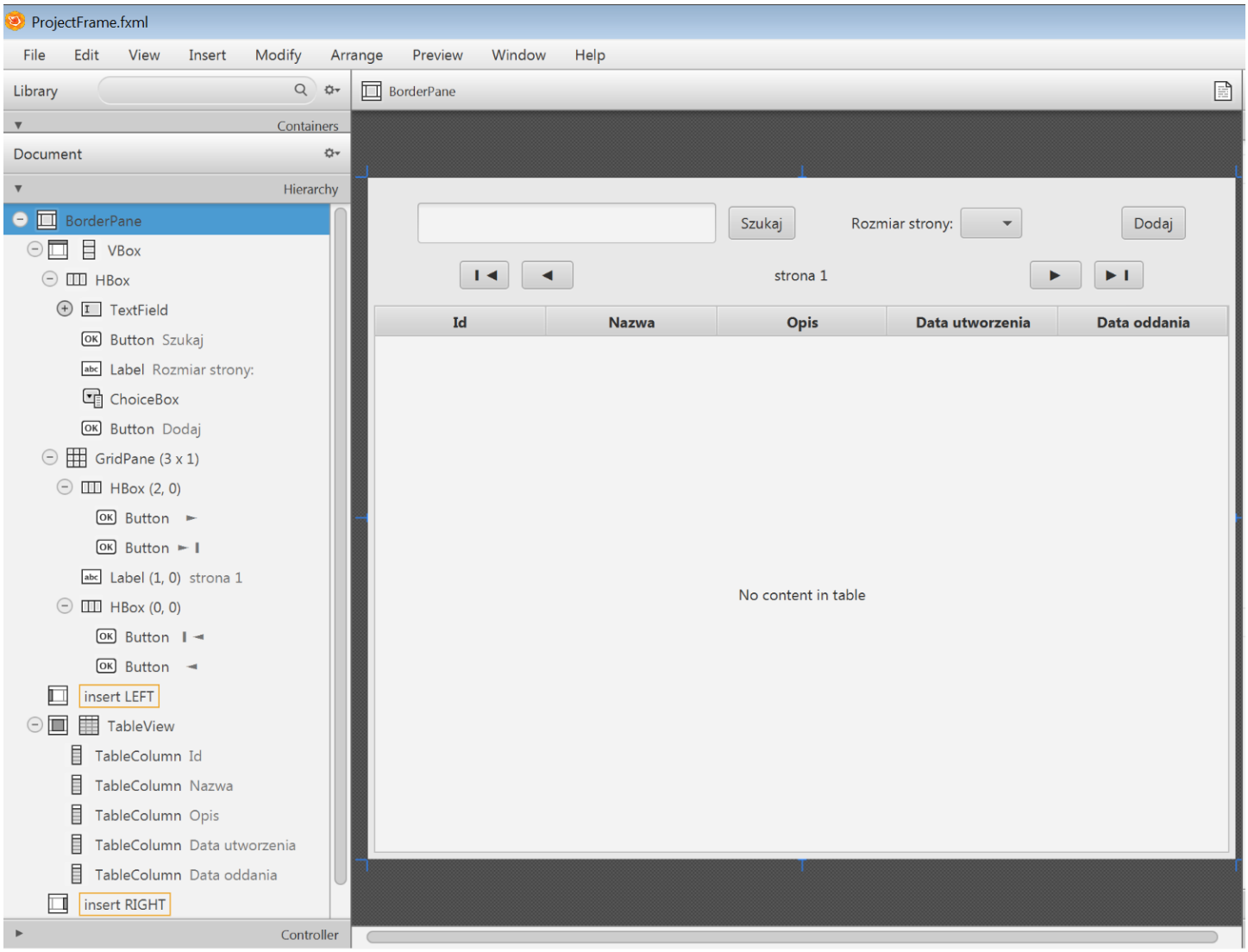
6.2. W katalogu *src\main\resources* utwórz podkatalog *fxml*. Wybierz z menu *File -> New -> Other* (lub użyj skrótu *CTRL + N*), następnie zaznacz *New FXML Document* (z *JavaFX*) i kliknij *Next*. W nowo otwartym oknie wpisz nazwę pliku *ProjectFrame*, a także z listy *Root Element* wybierz *BorderPane* i naciśnij *Finish*. Ponadto w pakiecie *com.project.controller* utwórz klasę kontrolera o nazwie *ProjectController*.



6.3. Kliknij prawym przyciskiem myszki na pliku *src\main\resources\fxml\ProjectFrame.fxml* i wybierz *Open with SceneBuilder*. W polu *Controller class* wpisz klasę kontrolera poprzedzoną nazwą pakietu.



Następnie zaprojektuj przedstawione poniżej okno aplikacji. Aby utworzyć tekst przycisków nawigacji można użyć znaków *Unicode* (najlepiej wpisać je bezpośrednio w pliku *ProjectFrame.fxml*) np. |◄ (trzeba wpisać jako ❙◄), ◄(◄), ►(►), ►| (►❙).
Pamiętaj, aby przed zamknięciem SceneBuildera zapisać zmiany.

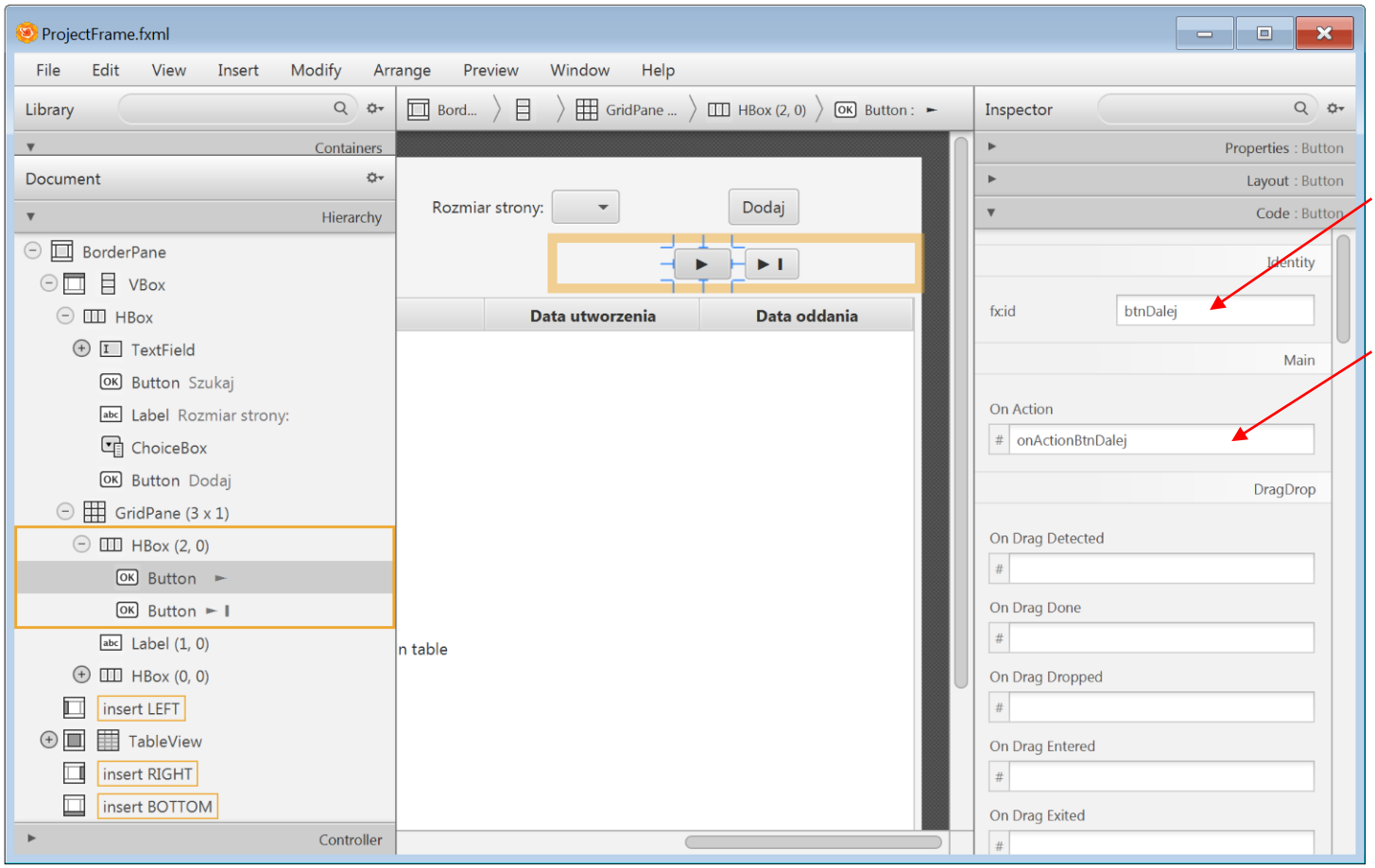


6.4. W klasie *ProjectController* trzeba dodać m.in. zmienne komponentów, do których dzięki adnotacji *@FXML* JavaFx wstrzyknie utworzone przez nią obiekty interfejsu użytkownika. Poza tym należy również zdefiniować grupę metod oznaczonych tą samą adnotacją, których celem będzie realizacja zadań wykonywanych po wciśnięciu przycisków, a także opatrzyć adnotacją *@FXML* automatycznie wywoływaną podczas tworzenia obiektu kontrolera metodę *initialize()*.

```
package com.project.controller;

import java.time.LocalDate;
import java.time.LocalDateTime;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.project.model.Projekt;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;

public class ProjectController {
   private static final Logger logger = LoggerFactory.getLogger(ProjectController.class);

   //Zmienne do obsługi stronicowania i wyszukiwania
   private String search4;
   private Integer pageNo;
   private Integer pageSize;

   //Automatycznie wstrzykiwane komponenty GUI
   @FXML
   private ChoiceBox<Integer> cbPageSizes;
```

```
    @FXML
    private TableView<Projekt> tblProjekt;
    @FXML
    private TableColumn<Projekt, Integer> colId;
    @FXML
    private TableColumn<Projekt, String> colNazwa;
    @FXML
    private TableColumn<Projekt, String> colOpis;
    @FXML
    private TableColumn<Projekt, LocalDateTime> colDataCzasUtworzenia;
    @FXML
    private TableColumn<Projekt, LocalDate> colDataOddania;
    @FXML
    private TextField txtSzukaj;
    @FXML
    private Button btnDalej;
    @FXML
    private Button btnWstecz;
    @FXML
    private Button btnPierwsza;
    @FXML
    private Button btnOstatnia;

    public ProjectController() { //Utworzeniu konstruktora jest obligatoryjne
    }

    @FXML
    public void initialize() { //Metoda automatycznie wywoływana przez JavaFX zaraz po wstrzyknięciu
        search4 = "";           //wszystkich komponentów. Uwaga! Wszelkie modyfikacje komponentów
        pageNo = 0;             //(np. cbPageSizes) trzeba realizować wewnątrz tej metody. Nigdy
        pageSize = 10;          //nie używaj do tego celu konstruktora.

        cbPageSizes.getItems().addAll(5, 10, 20, 50, 100);
        cbPageSizes.setValue(pageSize);
    }

    //Grupa metod do obsługi przycisków
    @FXML
    private void onActionBtnSzukaj(ActionEvent event) {
    }

    @FXML
    private void onActionBtnDalej(ActionEvent event) {
    }

    @FXML
    private void onActionBtnWstecz(ActionEvent event) {
    }

    @FXML
    private void onActionBtnPierwsza(ActionEvent event) {
    }

    @FXML
    private void onActionBtnOstatnia(ActionEvent event) {
    }

    @FXML
    private void onActionBtnDodaj(ActionEvent event) {
    }
}
```

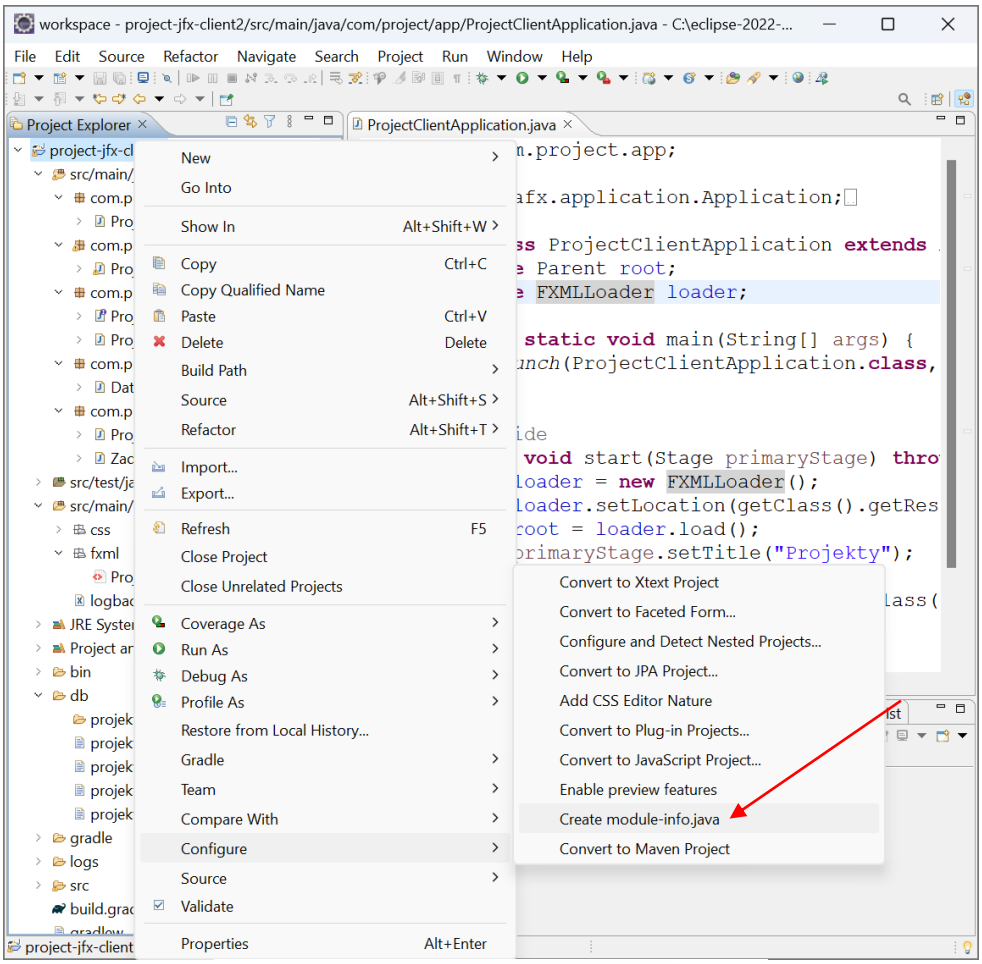
Uwaga! Nazwy zmiennych oraz metod obsługujących zdarzenia w klasie *ProjectController* muszą być identyczne jak te przypisane w atrybutach *fx:id* i *onAction* komponentów zdefiniowanych w pliku *src\main\resources\fxml\ProjectFrame.fxml*. Aby utworzyć atrybuty edytuj plik FXML za pomocą SceneBuildera, wybieraj kolejne elementy klikając na nich i wpisuj odpowiednie nazwy w zaznaczonych na poniższym rysunku polach tekstowych zakładki *Code*.

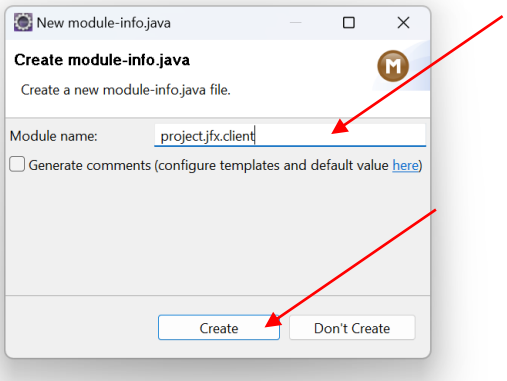


6.5. Zmodyfikuj utworzoną w punkcie 5.7 klasę *ProjectClientApplication* tak jak pokazano poniżej.

```
package com.project.app;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class ProjectClientApplication extends Application {
   private Parent root;
   private FXMLLoader loader;

   public static void main(String[] args) {
      DbInitializer.init();
      launch(ProjectClientApplication.class, args);
   }

   @Override
   public void start(Stage primaryStage) throws Exception {
      loader = new FXMLLoader();
      loader.setLocation(getClass().getResource("/fxml/ProjectFrame.fxml"));
      root = loader.load();
      primaryStage.setTitle("Projekty");
      Scene scene = new Scene(root);
      scene.getStylesheets().add(getClass().getResource("/css/application.css").toExternalForm());
      primaryStage.setScene(scene);
      primaryStage.sizeToScene();
      primaryStage.show();
   }

}
```

W następnym kroku, jeśli plik *module-info.java* jeszcze nie istnieje, utwórz go klikając prawym przyciskiem myszki na głównej ikonie projektu i wybierając z menu *Configure -> Create module-info.java*. W nowo otwartym oknie wpisz nazwę modułu ***project.jfx.client*** i naciśnij przycisk *Create*.



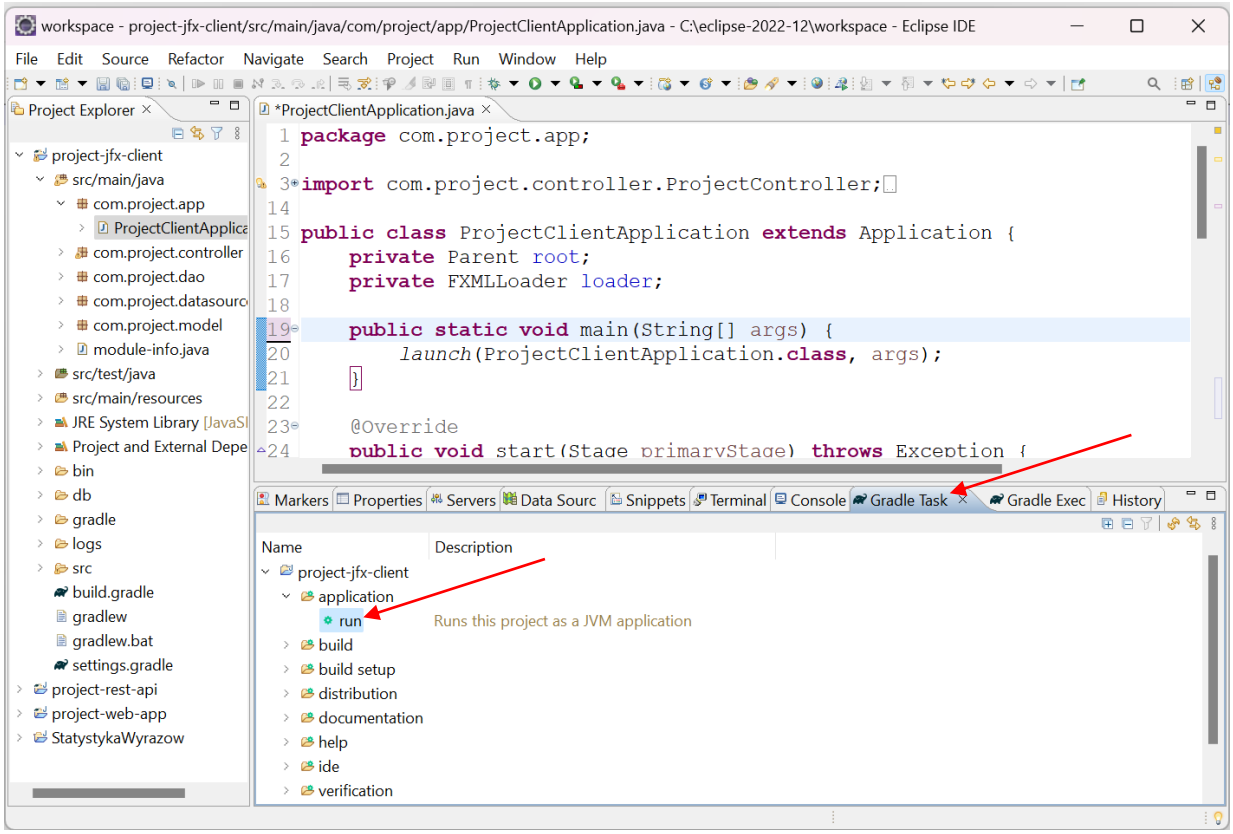


Edytuj plik *module-info.java*, który właśnie utworzyłeś i wprowadź do niego poniższą zawartość. Pamiętaj, że jeśli użyłeś innych nazw pakietów niż te podane w instrukcji, musisz zastąpić je odpowiednimi nazwami w tym pliku.

```
module project.jfx.client {
    exports com.project.datasource;
    exports com.project.dao;
    exports com.project.model;
    exports com.project.app;
    exports com.project.controller;

    requires javafx.base;
    requires javafx.fxml;
    requires javafx.controls;
    requires transitive javafx.graphics;
    requires com.zaxxer.hikari;
    requires transitive java.sql;
    requires org.hsqldb;
```

```
    requires org.slf4j;
    requires ch.qos.logback.classic;
    requires ch.qos.logback.core;

    opens com.project.app to javafx.graphics, javafx.fxml, javafx.base, javafx.controls;
    opens com.project.model to javafx.graphics, javafx.fxml, javafx.base, javafx.controls;
    opens com.project.controller to javafx.graphics, javafx.fxml, javafx.base, javafx.controls;
}
```

Aplikacja będzie odtąd uruchamiana za pomocą gradle'owskiego zadania *run*. Aby ją uruchomić przejdź do widoku *Gradle Task* (jeżeli widok nie jest wyświetlany to z menu wybierz *Window -> Show View -> Other*, a następnie zaznacz *Gradle Tasks*), rozwiń węzeł *application* i dwukrotnie kliknij lewym przyciskiem myszki na *run*.



6.6. Inicjalizacja kolumn tabeli oraz utworzenie w kontrolerze listy typu *ObservableList*, w której będą przechowywane pobrane z bazy dane. Zaletą użycia *ObservableCollections* z JavaFx jest fakt, że gdy zmodyfikujemy taką kolekcję to element GUI z nią powiązany zostanie automatycznie zaktualizowany.

```
package com.project.controller;
 ...
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.collections.ObservableList;
import javafx.collections.FXCollections;

public class ProjectController {
    ...

   private ObservableList<Projekt> projekty;

    ...

   @FXML
   public void initialize() {
       ...
      colId.setCellValueFactory(new PropertyValueFactory<Projekt, Integer>("projektId"));
      colNazwa.setCellValueFactory(new PropertyValueFactory<Projekt, String>("nazwa"));
      colOpis.setCellValueFactory(new PropertyValueFactory<Projekt, String>("opis"));
      colDataCzasUtworzenia.setCellValueFactory(new PropertyValueFactory<Projekt, LocalDateTime>
                                                                          ("dataCzasUtworzenia"));
      colDataOddania.setCellValueFactory(new PropertyValueFactory<Projekt, LocalDate>("dataOddania"));
```

```
        projekty = FXCollections.observableArrayList();

        //Powiązanie tabeli z listą typu ObservableList przechowującą projekty
        tblProjekt.setItems(projekty);
    }

    ...
}
```

6.7. Dodaj pulę wątków – obiekt klasy *ExecutorService*, który zarządza tworzeniem nowych oraz wykonuje „recykling" zakończonych wątków. Ponadto utwórz obiekt *DAO* zapewniający dostęp do danych, dodaj kolejny konstruktor uwzględniający ten obiekt oraz metodę *loadPage*, której zadaniem będzie pobieranie i przeładowywanie wskazanego podzbioru projektów. Trzeba również pamiętać, że przed zamknięciem aplikacji z obiektu klasy *ExecutorService* powinna być wywoływana jedna z metod - *shutdown* lub *shutdownNow*. W tym celu w kontrolerze dodajemy publiczną metodę również o nazwie *shutdown*.

```
package com.project.controller;
 ...
import javafx.application.Platform;
import com.project.dao.ProjektDAO;
import com.project.dao.ProjektDAOImpl;
import com.project.model.Projekt;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class ProjectController {
     ...
    private ExecutorService wykonawca;
    private ProjektDAO projektDAO;

     ...
    public ProjectController(ProjektDAO projektDAO) {
        this.projektDAO = projektDAO;
        wykonawca = Executors.newFixedThreadPool(1);// W naszej aplikacji wystarczy jeden wątek do pobierania
                                                    // danych. Przekazanie większej ilości takich zadań do puli
    }                                               // jednowątkowej powoduje ich kolejkowanie i sukcesywne
                                                    // wykonywanie.
    @FXML
    public void initialize() {
        ...

        wykonawca.execute(() -> loadPage(search4, pageNo, pageSize));

    }

    private void loadPage(String search4, Integer pageNo, Integer pageSize) {
        try {
              final List<Projekt> projektList = new ArrayList<>();
              if (search4 != null && !search4.isEmpty()) {

                /* TODO Tutaj można sprawdzać za pomocą wyrażeń regularnych, czy search4 jest nazwą,
                   datą czy też identyfikatorem i wtedy wywoływać odpowiednią metodę z projektDAO
                   tj. getProjektyWhereNazwaLike, getProjektyWhereDataOddaniaIs lub getProjekt, np.:

                    if (search4.matches("[0-9]+")) {
                       //IDENTYFIKATOR
                    }else if (search4.matches("^[0-9]{4}-(0[1-9]|1[0-2])-(0[1-9]|[12][0-9]|3[01])$")) {
                       //DATA
                    }else {
                       //NAZWA
                    }
                */
                   projektList.addAll(projektDAO.getProjektyWhereNazwaLike(search4, pageNo * pageSize, pageSize));
              }else {
                 projektList.addAll(projektDAO.getProjekty(pageNo * pageSize, pageSize));
              }
              Platform.runLater(() -> {
                 projekty.clear();
                 projekty.addAll(projektList);
              });
```

```
    } catch (RuntimeException e) {
          String errMsg = "Błąd podczas pobierania listy projektów.";
          logger.error(errMsg, e);
          String errDetails = e.getCause() != null ?
                    e.getMessage() + "\n" + e.getCause().getMessage()
                  : e.getMessage();
          Platform.runLater(() -> showError(errMsg, errDetails));
    }
}

/** Metoda pomocnicza do prezentowania użytkownikowi informacji o błędach */
private void showError(String header, String content) {
    Alert alert = new Alert(AlertType.ERROR);
    alert.setTitle("Błąd");
    alert.setHeaderText(header);
    alert.setContentText(content);
    alert.showAndWait();
}

public void shutdown() {
  // Wystarczyłoby tylko samo wywołanie metody wykonawca.shutdownNow(), ale można również, tak jak poniżej,
  // zaimplementować wersję z oczekiwaniem na zakończenie wszystkich zadań wykonywanych w puli wątków.
  if(wykonawca != null) {
      wykonawca.shutdown();
      try {
         if(!wykonawca.awaitTermination(5, TimeUnit.SECONDS))
            wykonawca.shutdownNow();
      } catch (InterruptedException e) {
         wykonawca.shutdownNow();
      }
    }
}
```

6.8. Modyfikujemy metodę *start* klasy *ProjectClientApplication*, aby przekazywać do kontrolera *ProjectController* (za pomocą jego konstruktora) utworzony obiekt *projektDAO*. Ponadto rejestrujemy automatyczne wywoływanie, podczas zamykania aplikacji, metody *shutdown* kontrolera.

```
package com.project.app;

import com.project.controller.ProjectController;
import com.project.dao.ProjektDAO;
import com.project.datasource.DbInitializer;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class ProjectClientApplication extends Application {
       private Parent root;
       private FXMLLoader loader;

       public static void main(String[] args) {
              DbInitializer.init();
              launch(ProjectClientApplication.class, args);
       }

       @Override
       public void start(Stage primaryStage) throws Exception {
              loader = new FXMLLoader();
              loader.setLocation(getClass().getResource("/fxml/ProjectFrame.fxml"));

              ProjektDAO projektDAO = new ProjektDAOImpl();
              loader.setControllerFactory(controllerClass ->  new ProjectController(projektDAO));
              root = loader.load();

              primaryStage.setTitle("Projekty");
              Scene scene = new Scene(root);
              scene.getStylesheets()
                     .add(getClass().getResource("/css/application.css")
                     .toExternalForm());
```

```
                ProjectController controller = loader.getController();
                primaryStage.setOnCloseRequest(event -> {
                        controller.shutdown();
                        Platform.exit();
                });

                primaryStage.setScene(scene);
                primaryStage.sizeToScene();
                primaryStage.show();
        }
}
```

6.9. Formatowanie daty i czasu.

```
package com.project.controller;
 ...
public class ProjectController {
    ...
  private static final DateTimeFormatter dateFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
  private static final DateTimeFormatter dateTimeFormater = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    ...


   @FXML
   public void initialize() {
       ...
      colDataCzasUtworzenia.setCellFactory(column -> new TableCell<Projekt, LocalDateTime>() {
         @Override
         protected void updateItem(LocalDateTime item, boolean empty) {
            super.updateItem(item, empty);
            if (item == null || empty) {
              setText(null);
            } else {
              setText(dateTimeFormater.format(item));
            }
         }
      });

   }

   ...
}
```

6.10. Tworzenie projektu. Do klasy kontrolera trzeba dodać przedstawioną poniżej metodę *edytujProjekt* oraz metodę pomocniczą *getRightLabel*. Następnie należy wstawić w *onActionBtnDodaj* wywołanie metody *edytujProjekt* przekazując w jej parametrze nowo utworzony, pusty projekt.

```
package com.project.controller;

...
import java.time.LocalDate;
import java.util.concurrent.ExecutorService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.project.dao.ProjektDAO;
import com.project.model.Projekt;
import java.util.Optional;
import java.time.format.DateTimeFormatter;
import java.time.format.DateTimeParseException;
import javafx.fxml.FXML;
import javafx.application.Platform;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.ButtonBar.ButtonData;
import javafx.scene.control.Button;
import javafx.scene.control.ButtonType;
```

```
import javafx.scene.control.DatePicker;
import javafx.scene.control.Dialog;
import javafx.scene.control.Label;
import javafx.scene.control.TableView;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.layout.GridPane;
import javafx.util.Callback;
import javafx.util.StringConverter;

public class ProjectController {
     ...

     @FXML
     public void onActionBtnDodaj(ActionEvent event) {
          edytujProjekt(new Projekt());
     }

     ...

     private void edytujProjekt(Projekt projekt) {
          Dialog<Projekt> dialog = new Dialog<>();
          dialog.setTitle("Edycja");
          if (projekt.getProjektId() != null) {
               dialog.setHeaderText("Edycja danych projektu");
          } else {
               dialog.setHeaderText("Dodawanie projektu");
          }
          dialog.setResizable(true);
          Label lblId = getRightLabel("Id: ");
          Label lblNazwa = getRightLabel("Nazwa: ");
          Label lblOpis = getRightLabel("Opis: ");
          Label lblDataCzasUtworzenia = getRightLabel("Data utworzenia: ");
          Label lblDataOddania = getRightLabel("Data oddania: ");
          Label txtId = new Label();
          if (projekt.getProjektId() != null)
               txtId.setText(projekt.getProjektId().toString());
          TextField txtNazwa = new TextField();
          if (projekt.getNazwa() != null)
               txtNazwa.setText(projekt.getNazwa());
          TextArea txtOpis = new TextArea();
          txtOpis.setPrefRowCount(6);
          txtOpis.setPrefColumnCount(40);
          txtOpis.setWrapText(true);
          if (projekt.getOpis() != null)
               txtOpis.setText(projekt.getOpis());
          Label txtDataUtworzenia = new Label();
          if (projekt.getDataCzasUtworzenia() != null)
               txtDataUtworzenia.setText(dateTimeFormater.format(projekt.getDataCzasUtworzenia()));

          DatePicker dtDataOddania = new DatePicker();
          dtDataOddania.setPromptText("RRRR-MM-DD");
          dtDataOddania.setConverter(new StringConverter<LocalDate>() {
               @Override
               public String toString(LocalDate date) {
                    return date != null ? dateFormatter.format(date) : null;
               }

               @Override
               public LocalDate fromString(String text) {
                    return text == null || text.trim().isEmpty() ? null : LocalDate.parse(text, dateFormatter);
               }
          });
          dtDataOddania.getEditor().focusedProperty().addListener((obsValue, oldFocus, newFocus) -> {
               if (!newFocus) {
                    try {
                         dtDataOddania.setValue(dtDataOddania.getConverter().fromString(
                                                                 dtDataOddania.getEditor().getText()));
                    } catch (DateTimeParseException e) {
                         dtDataOddania.getEditor().setText(dtDataOddania.getConverter()
                                                                  .toString(dtDataOddania.getValue()));
                    }
               }
          });
```

```
        if (projekt.getDataOddania() != null) {
            dtDataOddania.setValue(projekt.getDataOddania());
        }

        GridPane grid = new GridPane();
        grid.setHgap(10);
        grid.setVgap(10);
        grid.setPadding(new Insets(5, 5, 5, 5));
        grid.add(lblId, 0, 0);
        grid.add(txtId, 1, 0);
        grid.add(lblDataCzasUtworzenia, 0, 1);
        grid.add(txtDataUtworzenia, 1, 1);
        grid.add(lblNazwa, 0, 2);
        grid.add(txtNazwa, 1, 2);
        grid.add(lblOpis, 0, 3);
        grid.add(txtOpis, 1, 3);
        grid.add(lblDataOddania, 0, 4);
        grid.add(dtDataOddania, 1, 4);
        dialog.getDialogPane().setContent(grid);

        ButtonType buttonTypeOk = new ButtonType("Zapisz", ButtonData.OK_DONE);
        ButtonType buttonTypeCancel = new ButtonType("Anuluj", ButtonData.CANCEL_CLOSE);
        dialog.getDialogPane().getButtonTypes().add(buttonTypeOk);
        dialog.getDialogPane().getButtonTypes().add(buttonTypeCancel);
        dialog.setResultConverter(new Callback<ButtonType, Projekt>() {
            @Override
            public Projekt call(ButtonType butonType) {
                if (butonType == buttonTypeOk) {
                    projekt.setNazwa(txtNazwa.getText().trim());
                    projekt.setOpis(txtOpis.getText().trim());
                    projekt.setDataOddania(dtDataOddania.getValue());
                    return projekt;
                }
                return null;
            }
        });

        Optional<Projekt> result = dialog.showAndWait();
        if (result.isPresent()) {
            wykonawca.execute(() -> {
                try {
                    projektDAO.setProjekt(projekt);
                    Platform.runLater(() -> {
                        if (tblProjekt.getItems().contains(projekt)) {
                            tblProjekt.refresh();
                        } else {
                            tblProjekt.getItems().add(0, projekt);
                        }
                    });
                } catch (RuntimeException e) {
                    String errMsg = "Błąd podczas zapisywania danych projektu!";
                    Logger.error(errMsg, e);
                    String errDetails = e.getCause() != null ?
                                e.getMessage() + "\n" + e.getCause().getMessage()
                              : e.getMessage();
                    Platform.runLater(() -> showError(errMsg, errDetails));
                }
            });
        }
    }

    private Label getRightLabel(String text) {
        Label lbl = new Label(text);
        lbl.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
        lbl.setAlignment(Pos.CENTER_RIGHT);
        return lbl;
    }
```

6.11. Edycja projektu. W metodzie *initialize* klasy *ProjectController*, po fragmencie inicjalizującym kolumny tabeli, dodajemy nową kolumnę z przyciskami *Edytuj*, *Usuń* i *Zadania*. Można również ustawić względną szerokość poszczególnych kolumn za pomocą *setMaxWidth*.

```
@FXML
public void initialize() {
  ...

    colId.setCellValueFactory(new PropertyValueFactory<Projekt, Integer>("projektId"));
    colNazwa.setCellValueFactory(new PropertyValueFactory<Projekt, String>("nazwa"));
    colOpis.setCellValueFactory(new PropertyValueFactory<Projekt, String>("opis"));
    colDataCzasUtworzenia.setCellValueFactory(new PropertyValueFactory<Projekt,
                                                        LocalDateTime>("dataCzasUtworzenia"));
    colDataOddania.setCellValueFactory(new PropertyValueFactory<Projekt, LocalDate>("dataOddania"));

      //Utworzenie nowej kolumny
      TableColumn<Projekt, Void> colEdit = new TableColumn<>("Edycja");
      colEdit.setCellFactory(column -> new TableCell<Projekt, Void>() {
             private final GridPane pane;
             {       // Blok inicjalizujący w anonimowej klasie wewnętrznej
                    Button btnEdit = new Button("Edycja");
                    Button btnRemove = new Button("Usuń");
                    Button btnTask = new Button("Zadania");
                    btnEdit.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
                    btnRemove.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
                    btnTask.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
                    btnEdit.setOnAction(event -> {
                       edytujProjekt(getCurrentProjekt());
                    });
                    btnRemove.setOnAction(event -> {
                       //TODO wywoływać metodę usunProjekt(getCurrentProjekt()); po jej utworzeniu
                    });
                    btnTask.setOnAction(event -> {
                       //TODO wywoływać metodę openZadanieFrame(getCurrentProjekt()); po jej utworzeniu
                    });
                    pane = new GridPane();
                    pane.setAlignment(Pos.CENTER);
                    pane.setHgap(10);
                    pane.setVgap(10);
                    pane.setPadding(new Insets(5, 5, 5, 5));
                    pane.add(btnTask, 0, 0);
                    pane.add(btnEdit, 0, 1);
                    pane.add(btnRemove, 0, 2);
             }
             private Projekt getCurrentProjekt() {
                int index = this.getTableRow().getIndex();
                return this.getTableView().getItems().get(index);
             }
             @Override
             protected void updateItem(Void item, boolean empty) {
                super.updateItem(item, empty);
                setGraphic(empty ? null : pane);
             }
    });

    //Dodanie kolumny do tabeli
    tblProjekt.getColumns().add(colEdit);

    //Ustawienie względnej szerokości poszczególnych kolumn (liczą się proporcje)
    colId.setMaxWidth(5000);
    colNazwa.setMaxWidth(10000);
    colOpis.setMaxWidth(10000);
    colDataCzasUtworzenia.setMaxWidth(9000);
    colDataOddania.setMaxWidth(7000);
    colEdit.setMaxWidth(7000);

    ...
}
```

6.12. Usuwanie projektu.
W klasie *ProjectController* zaimplementuj metodę `private void usunProjekt(Projekt projekt)`. Podczas usuwania powinno pojawiać się okno dialogowe żądające potwierdzenia operacji. Wywołanie metody musi znaleźć się we fragmencie (zawartym w *initialize()*, p. 6.11) rejestrującym odbiorcę zdarzeń dla przycisku *btnRemove*
tj.

```
btnRemove.setOnAction(event -> {
   usunProjekt(getCurrentProjekt());
});
```

6.13. Stronicowanie i wyszukiwanie projektu.
Zaimplementuj wyszukiwanie projektu za pomocą słowa wpisywanego w polu tekstowym *txtSzukaj*. Mechanizm wyszukiwania powinien uwzględniać nazwę, identyfikator projektu oraz jego datę oddania. Do wyznaczania sposobu wyszukiwania można skorzystać z wyrażeń regularnych testujących zawartość pola tekstowego *txtSzukaj*. Dodaj również obsługę przycisków w metodach *onActionBtnDalej* i *onActionBtnWstecz*.

6.14. Zadania dodatkowe, tylko dla chętnych:

- implementacja metod *onActionBtnPierwsza* i *onActionBtnOstatnia*,
- zmiana rozmiaru strony,
- implementacja okna wyświetlającego listę zadań przypisaną do danego projektu, a także ekran edycji umożliwiający tworzenie i modyfikacje zadań. Poniższy listing prezentuje przykład, który ułatwi realizację tej funkcji oprogramowania.

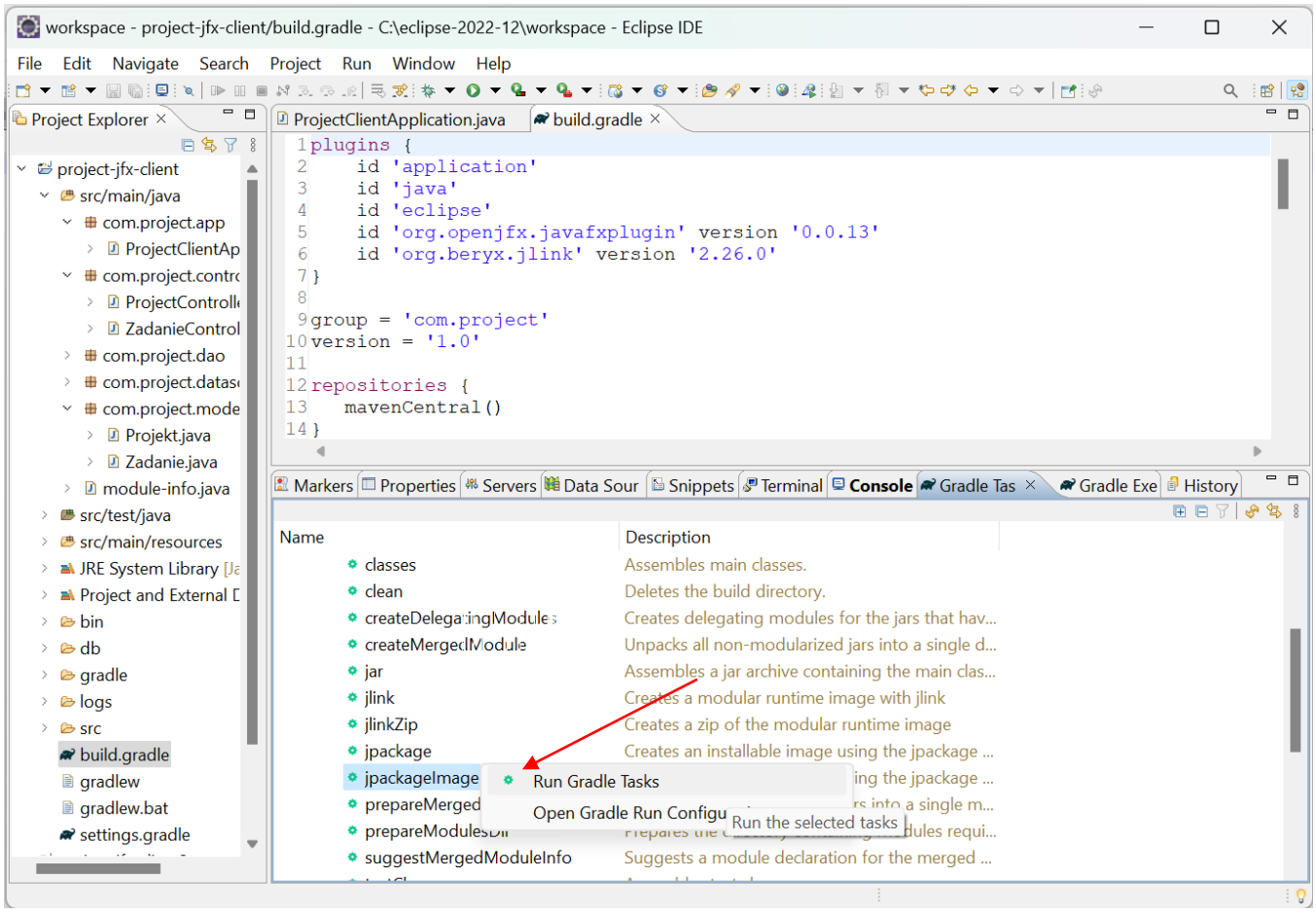
```
public class ProjectController {
    ...
    private ExecutorService wykonawca;
    private ProjektDAO projektDAO;
    private ZadanieDAOImpl zadanieDAO;
    ...
    private Stage openZadanieFrame(Projekt projekt) {
       try {
           FXMLLoader loader = new FXMLLoader(getClass().getResource("/fxml/ZadanieFrame.fxml"));
           loader.setControllerFactory(
                  controllerClass -> new ZadanieController(projekt, this.zadanieDAO, this.wykonawca));
           Stage stage = new Stage(StageStyle.DECORATED);
           stage.initModality(Modality.APPLICATION_MODAL);
           stage.setTitle("Zadania");
           Scene scene = new Scene(loader.load());
           scene.getStylesheets()
                  .add(getClass().getResource("/css/application.css").toExternalForm());
           stage.setScene(scene);
           stage.show();
           return stage;
       } catch (IOException e) {
           throw new RuntimeException(e);
       }
    }
    ...
}
```

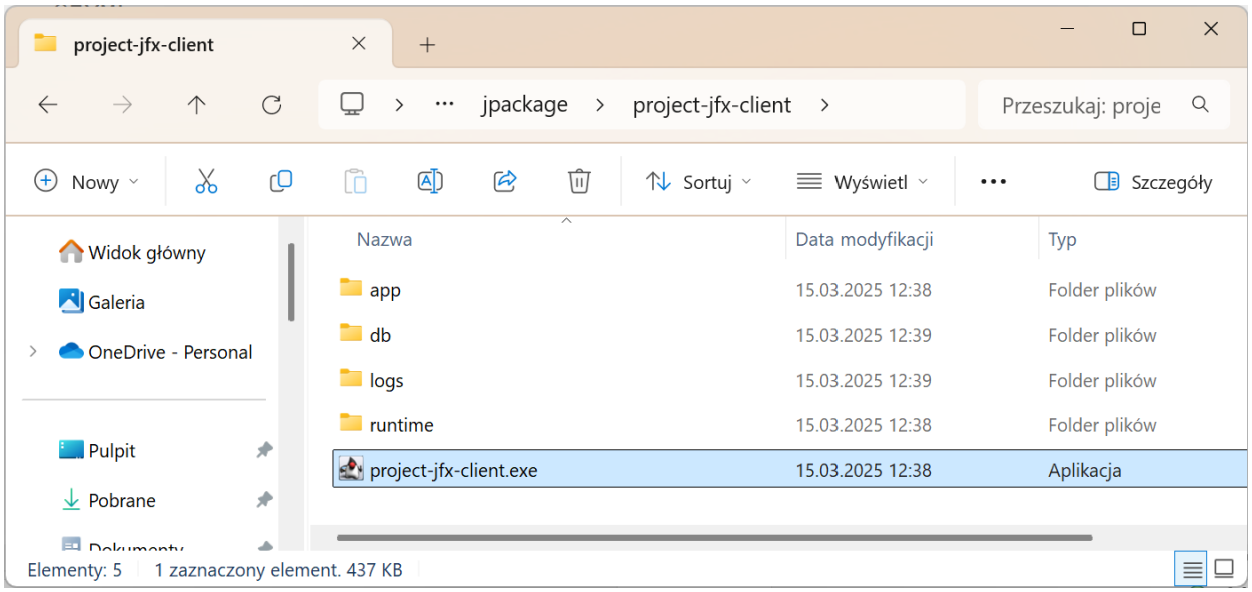
```
public class ZadanieController {
    ...
    @FXML
    private Button btnPowrot;
    private ExecutorService wykonawca;
    private ZadanieDAO zadanieDAO;
    private Projekt projekt;

    public ZadanieController(Projekt projekt, ZadanieDAO zadanieDAO, ExecutorService wykonawca) {
       this.projekt = projekt;
       this.zadanieDAO = zadanieDAO;
       this.wykonawca = wykonawca;
    }
```

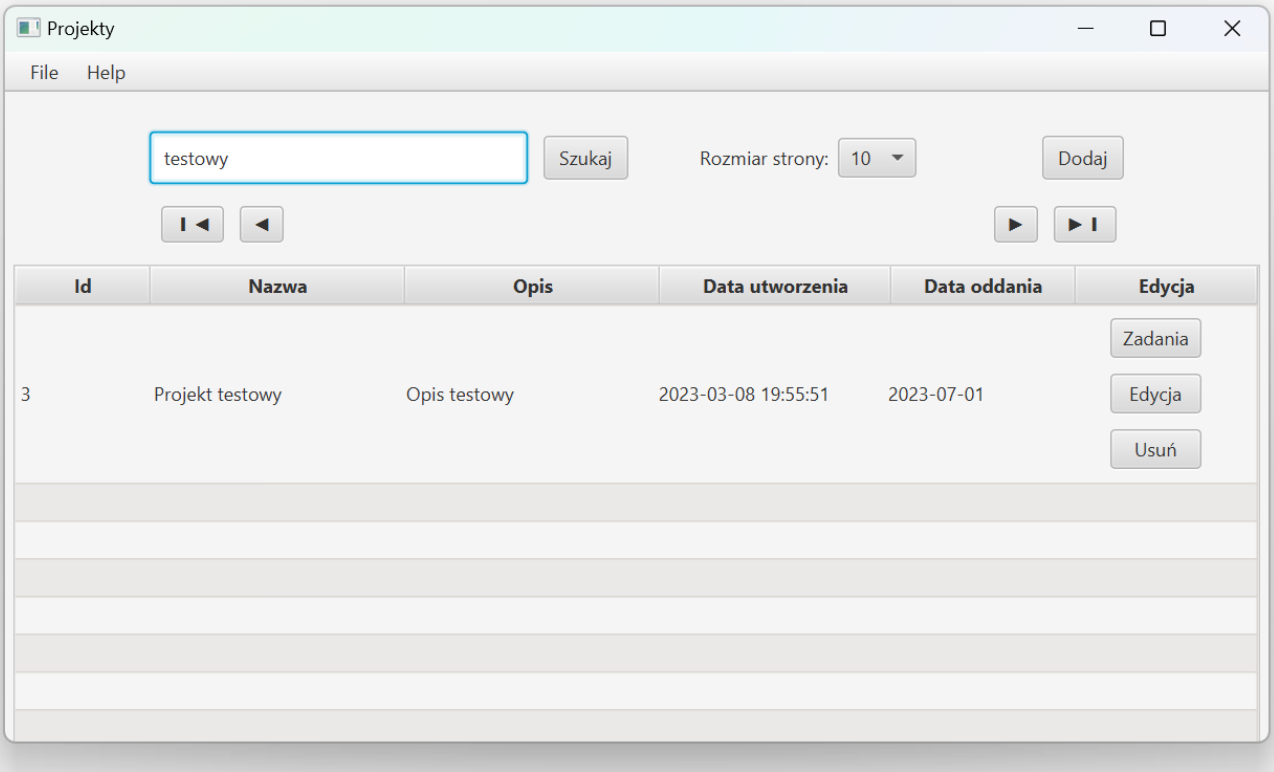
```
    ...
    @FXML
    private void onActionBtnPowrot(ActionEvent event) {
        Stage stage = (Stage) btnPowrot.getScene().getWindow();
        stage.fireEvent(new WindowEvent(stage, WindowEvent.WINDOW_CLOSE_REQUEST));
    }

    ...
}
```

7. UTWORZENIE WERSJI URUCHOMIENIOWEJ

W widoku *Gradle Tasks* (jeżeli widok nie jest wyświetlany to z menu wybierz *Window -> Show View -> Other*, a następnie zaznacz *Gradle Tasks*) kliknij prawym przyciskiem myszki na ***jpackageImage*** (z *project-jfx-client/build*) i wybierz *Run Gradle Tasks*. Utworzoną wersję uruchomieniową można znaleźć w katalogu *project-jfx-client\build\jpackage\project-jfx-client* (niewidocznym z poziomu Eclipse'a).




project-jfx-client
jpackage
project-jfx-client
Przeszukaj: proje
Nowy
Sortuj
Wyświetl
Szczegóły
Widok główny
Galeria
OneDrive - Personal
Pulpit
Pobrane
Nazwa
Data modyfikacji
Typ
app
15.03.2025 12:38
Folder plików
db
15.03.2025 12:39
Folder plików
logs
15.03.2025 12:39
Folder plików
runtime
15.03.2025 12:38
Folder plików
project-jfx-client.exe
15.03.2025 12:38
Aplikacja
Elementy: 5
1 zaznaczony element. 437 KB


Projekty
File
Help
testowy
Szukaj
Rozmiar strony:
10
Dodaj
Id
Nazwa
Opis
Data utworzenia
Data oddania
Edycja
3
Projekt testowy
Opis testowy
2023-03-08 19:55:51
2023-07-01
Zadania
Edycja
Usuń

PRZYDATNE SKRÓTY

CTRL + SHIFT + L – pokazuje wszystkie dostępne skróty

CTRL + SHIFT + F – formatowanie kodu

SHIFT + ALT + R – zmiana nazwy klasy, metody lub zmiennej itp., trzeba wcześniej ustawić kursor na nazwie

SHIFT + ALT + L – utworzenie zmiennej z zaznaczonego fragmentu kodu

SHIFT + ALT + M – utworzenie metody z zaznaczonego fragmentu kodu

CTRL + ALT + *STRZAŁKA W GÓRĘ* – skopiowanie linijki i wklejenie w bieżącym wierszu

CTRL + ALT + *STRZAŁKA W DÓŁ* – skopiowania bieżącej linijki i wklejenie poniżej

CTRL + SHIFT + O – automatyczne dodawanie i porządkowanie sekcji importów

CTRL + 1 – „zrób to co chcę zrobić”, m.in. sugestie rozwiązań bieżącego problemu

CTRL + Q – przejście do miejsca ostatniej modyfikacji

F11 – debugowanie aplikacji

CTRL + F11 – uruchomienie aplikacji

CTRL + M – powiększenie okna

Ustawienie kursora np. na wywołaniu metody, typie zmiennej, klasie importu itp. i wciśnięcie **F3** powoduje przejście do kodu źródłowego wywoływanej metody, klasy zmiennej, klasy importu itd.

Wpisanie ***sysout*** i naciśnięcie skrótu **CTRL + SPACJA** spowoduje wstawienie *System.out.println();*

Literatura

Herbert Schildt, "Java. Kompendium programisty". Wydawnictwo Helion, wydanie XII, 2022