**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Старший преподаватель департамента программной инженерии факультета компьютерных наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. А. Шершаков  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **Плагин для платформы IntelliJ для мониторинга процесса создания программы и формирования отчета**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.04.01-01 12 01-1-ЛУ** | | |
|  |  | |
| Исполнитель  студент группы \_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Т. В. Тибилов/  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. | |
|  | | |
|  | |  |

**Москва 2020**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| УТВЕРЖДЕН  RU.17701729.04.01-01 12 01-1-ЛУ |  | |  | |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **Плагин для платформы IntelliJ для мониторинга процесса создания программы и формирования отчета**  **Текст программы**  **RU.17701729.04.01-01 12 01-1**  **Листов 53** | | | | |
|  | |  | | |
|  | | |
|  | | | | |
|  | | | |  |

**Москва 2020**

ОГЛАВЛЕНИЕ

[**1. ТЕКСТ ПРОГРАММЫ 3**](#_Toc9372798)

# ТЕКСТ ПРОГРАММЫ

## Plugin.xml

<idea-plugin>  
 <id>org.taimuraztibilov.taskmanager</id>  
 <name>Task monitoring manager</name>  
 <vendor email="taimuraztibilov@yandex.ru" url="https://github.com/TaimurazTibilov">Taimuraz Tibilov</vendor>  
  
 <description><![CDATA[  
 <h2 class="code-line" data-line-start=0 data-line-end=1 >  
 <a id="Task\_Monitoring\_Manager\_Plugin\_0"></a>  
 Task Monitoring Manager Plugin  
 </h2>  
 <p class="has-line-data" data-line-start="1" data-line-end="2">  
 Plugin helps to manage your tasks on GitLab repository and create report about closed issues.  
 </p>  
 ]]></description>  
  
 *<!-- please see https://www.jetbrains.org/intellij/sdk/docs/basics/getting\_started/plugin\_compatibility.html  
 on how to target different products -->* <depends>com.intellij.modules.platform</depends>  
  
 <extensions defaultExtensionNs="com.intellij">  
 <applicationService serviceImplementation="org.taimuraztibilov.taskmanager.base.DataBaseManager"/>  
 <applicationService serviceImplementation="org.taimuraztibilov.taskmanager.base.ReportManager"/>  
 <applicationService serviceImplementation="org.taimuraztibilov.taskmanager.base.TimeManager"/>  
 <applicationService serviceImplementation="org.taimuraztibilov.taskmanager.base.PluginManagerService"/>  
 </extensions>  
  
 <actions>  
 <group id="org.taimuraztibilov.taskmanager.TaskManager" popup="true" text="Task Manager Tools">  
 <add-to-group group-id="ToolsMenu" anchor="last"/>  
 <group id="org.taimuraztibilov.taskmanager.Add" popup="true" text="Add New">  
 <action class="org.taimuraztibilov.taskmanager.action.AddProjectAction"  
 id="org.taimuraztibilov.taskmanager.action.AddProjectAction" text="Project"  
 description="Add new project to track and report">  
 <keyboard-shortcut first-keystroke="control alt P" keymap="$default"/>  
 </action>  
 <action class="org.taimuraztibilov.taskmanager.action.AddMilestoneAction"  
 id="org.taimuraztibilov.taskmanager.action.AddMilestoneAction" text="Milestone"  
 description="Add new milestone to track and report">  
 <keyboard-shortcut first-keystroke="control alt M" keymap="$default"/>  
 </action>  
 <action class="org.taimuraztibilov.taskmanager.action.AddTaskAction"  
 id="org.taimuraztibilov.taskmanager.action.AddTaskAction" text="Task"  
 description="Add new task to track and report">  
 <keyboard-shortcut first-keystroke="control alt T" keymap="$default"/>  
 </action>  
 <action class="org.taimuraztibilov.taskmanager.action.AddLabelAction"  
 id="org.taimuraztibilov.taskmanager.action.AddLabelAction" text="Label"  
 description="Add new label to pull">  
 <keyboard-shortcut first-keystroke="control alt L" keymap="$default"/>  
 </action>  
 </group>  
 <separator/>  
 <action class="org.taimuraztibilov.taskmanager.action.TrackProjectAction"  
 id="org.taimuraztibilov.taskmanager.action.TrackProjectAction" text="Choose Project To Track"  
 description="Shows all existing tasks to track">  
 <keyboard-shortcut first-keystroke="shift alt P" keymap="$default"/>  
 </action>  
 <action class="org.taimuraztibilov.taskmanager.action.TrackMilestoneAction"  
 id="org.taimuraztibilov.taskmanager.action.TrackMilestoneAction" text="Choose Milestone To Track"  
 description="Shows all existing milestones for tracked project to track">  
 <keyboard-shortcut first-keystroke="shift alt M" keymap="$default"/>  
 </action>  
 <action class="org.taimuraztibilov.taskmanager.action.TrackTaskAction"  
 id="org.taimuraztibilov.taskmanager.action.TrackTaskAction" text="Choose Task To Track"  
 description="Shows all existing tasks for tracked milestone to track">  
 <keyboard-shortcut first-keystroke="shift alt T" keymap="$default"/>  
 </action>  
 <action class="org.taimuraztibilov.taskmanager.action.StopTrackAction"  
 id="org.taimuraztibilov.taskmanager.action.StopTrackAction" text="Stop Tracking"  
 description="Stops tracking task and creates keypoint">  
 <keyboard-shortcut first-keystroke="shift alt S" keymap="$default"/>  
 </action>  
 <separator/>  
 <action class="org.taimuraztibilov.taskmanager.action.CreateReportAction"  
 id="org.taimuraztibilov.taskmanager.action.CreateReportAction" text="Generate Report"  
 description="Generates report for traking project from-to date">  
 <keyboard-shortcut first-keystroke="control alt G" keymap="$default"/>  
 </action>  
 </group>  
 </actions>  
</idea-plugin>

## DataEditor.java

package org.taimuraztibilov.taskmanager.base;  
  
import java.sql.SQLException;  
  
public interface DataEditor {  
 void editData(String table, String column, String value, int id) throws SQLException;  
 void addLabelToTask(int labelId, int taskId) throws SQLException;  
 void removeLabelFromTask(int labelId, int taskId) throws SQLException;  
}

## Project.java

package org.taimuraztibilov.taskmanager.base;  
  
import java.sql.SQLException;  
import java.util.ArrayList;  
  
public class Project {  
 private final int id;  
 private String title;  
 private String description;  
 private int state;  
 private ArrayList<Milestone> milestones;  
 private DataEditor listenerOnEdit;  
  
 public Project(int id, String title, String description, int state) {  
 this.id = id;  
 this.title = title;  
 this.description = description;  
 this.state = state;  
 this.milestones = new ArrayList<>();  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public int getState() {  
 return state;  
 }  
  
 public ArrayList<Milestone> getMilestones() {  
 return milestones;  
 }  
  
 public Project setListenerOnEdit(DataEditor listenerOnEdit) {  
 this.listenerOnEdit = listenerOnEdit;  
 return this;  
 }  
  
 public void setTitle(String title) throws SQLException {  
 this.title = title;  
 listenerOnEdit.editData("project", "title", title, id);  
 }  
  
 public void setDescription(String description) throws SQLException {  
 this.description = description;  
 listenerOnEdit.editData("project", "description", description, id);  
 }  
  
 public void setState(int state) throws SQLException {  
 this.state = state;  
 listenerOnEdit.editData("project", "state", String.*valueOf*(state), id);  
 }  
  
 public void setMilestones(ArrayList<Milestone> milestones) {  
 this.milestones = milestones;  
 }  
}

## Milestone.java

package org.taimuraztibilov.taskmanager.base;  
  
import java.sql.SQLException;  
import java.time.LocalDateTime;  
import java.util.ArrayList;  
  
public class Milestone {  
 private final int id;  
 private final int projectId;  
 private String title;  
 private String description;  
 private LocalDateTime deadline;  
 private int state;  
 private ArrayList<Task> tasks;  
 private DataEditor listenerOnEdit;  
  
 public Milestone(int id, int projectId, String title, String description, LocalDateTime deadline, int state) {  
 this.id = id;  
 this.projectId = projectId;  
 this.title = title;  
 this.description = description;  
 this.deadline = deadline;  
 this.state = state;  
 this.tasks = new ArrayList<>();  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public int getProjectId() {  
 return projectId;  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public LocalDateTime getDeadline() {  
 return deadline;  
 }  
  
 public int getState() {  
 return state;  
 }  
  
 public ArrayList<Task> getTasks() {  
 return tasks;  
 }  
  
 public Milestone setListenerOnEdit(DataEditor listenerOnEdit) {  
 this.listenerOnEdit = listenerOnEdit;  
 return this;  
 }  
  
 public void setTitle(String title) throws SQLException {  
 this.title = title;  
 listenerOnEdit.editData("milestone", "title", title, id);  
 }  
  
 public void setDescription(String description) throws SQLException {  
 this.description = description;  
 listenerOnEdit.editData("milestone", "description", description, id);  
 }  
  
 public void setDeadline(LocalDateTime deadline) throws SQLException {  
 this.deadline = deadline;  
 listenerOnEdit.editData("milestone", "deadline", deadline.toString(), id);  
 }  
  
 public void setState(int state) throws SQLException {  
 this.state = state;  
 listenerOnEdit.editData("milestone", "state", String.*valueOf*(state), id);  
 }  
  
 public void setTasks(ArrayList<Task> tasks) {  
 this.tasks = tasks;  
 }  
}

## Task.java

package org.taimuraztibilov.taskmanager.base;  
  
import java.sql.SQLException;  
import java.time.LocalDateTime;  
import java.time.LocalTime;  
import java.util.ArrayList;  
  
public class Task {  
 private final int id;  
 private int milestoneId;  
 private String title;  
 private String description;  
 private final LocalDateTime createdOn;  
 private LocalDateTime deadline;  
 private LocalTime timeEstimated;  
 private LocalTime timeSpent;  
 private int state;  
 private ArrayList<Label> labels;  
 private ArrayList<KeyPoint> keyPoints;  
 private DataEditor listenerOnEdit;  
  
 public Task(int id, int milestoneId, String title, String description, LocalDateTime createdOn,  
 LocalDateTime deadline, LocalTime timeEstimated, LocalTime timeSpent, int state) {  
 this.id = id;  
 this.milestoneId = milestoneId;  
 this.title = title;  
 this.description = description;  
 this.createdOn = createdOn;  
 this.deadline = deadline;  
 this.timeEstimated = timeEstimated;  
 this.timeSpent = timeSpent;  
 this.state = state;  
 this.labels = new ArrayList<>();  
 this.keyPoints = new ArrayList<>();  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public int getMilestoneId() {  
 return milestoneId;  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public LocalDateTime getCreatedOn() {  
 return createdOn;  
 }  
  
 public LocalDateTime getDeadline() {  
 return deadline;  
 }  
  
 public LocalTime getTimeEstimated() {  
 return timeEstimated;  
 }  
  
 public LocalTime getTimeSpent() {  
 return timeSpent;  
 }  
  
 public int getState() {  
 return state;  
 }  
  
 public ArrayList<Label> getLabels() {  
 return labels;  
 }  
  
 public ArrayList<KeyPoint> getKeyPoints() {  
 return keyPoints;  
 }  
  
 public Task setListenerOnEdit(DataEditor listenerOnEdit) {  
 this.listenerOnEdit = listenerOnEdit;  
 return this;  
 }  
  
 public void setMilestoneId(int milestoneId) throws SQLException {  
 this.milestoneId = milestoneId;  
 listenerOnEdit.editData("task", "milestone\_id", String.*valueOf*(milestoneId), id);  
 }  
  
 public void setTitle(String title) throws SQLException {  
 this.title = title;  
 listenerOnEdit.editData("task", "title", title, id);  
 }  
  
 public void setDescription(String description) throws SQLException {  
 this.description = description;  
 listenerOnEdit.editData("task", "description", description, id);  
 }  
  
 public void setDeadline(LocalDateTime deadline) throws SQLException {  
 this.deadline = deadline;  
 listenerOnEdit.editData("task", "deadline", deadline.toString(), id);  
 }  
  
 public void setTimeEstimated(LocalTime timeEstimated) throws SQLException {  
 this.timeEstimated = timeEstimated;  
 listenerOnEdit.editData("task", "time\_estimated", timeEstimated.toString(), id);  
 }  
  
 public void setTimeSpent(LocalTime timeSpent) throws SQLException {  
 this.timeSpent = timeSpent;  
 listenerOnEdit.editData("task", "time\_spent", timeSpent.toString(), id);  
 }  
  
 public void setState(int state) throws SQLException {  
 this.state = state;  
 listenerOnEdit.editData("task", "state", String.*valueOf*(state), id);  
 }  
  
 public void setLabels(ArrayList<Label> labels) {  
 this.labels = labels;  
 }  
  
 public void setKeyPoints(ArrayList<KeyPoint> keyPoints) {  
 this.keyPoints = keyPoints;  
 }  
  
 public void addLabel(Label label) throws SQLException {  
 if (labels.contains(label))  
 return;  
 listenerOnEdit.addLabelToTask(label.getId(), id);  
 labels.add(label);  
 }  
  
 public void removeLabel(Label label) throws SQLException {  
 if (!labels.contains(label))  
 return;  
 listenerOnEdit.removeLabelFromTask(label.getId(), id);  
 labels.remove(label);  
 }  
  
 public void addKeyPoint(KeyPoint keyPoint) {  
 keyPoints.add(keyPoint);  
 }  
  
 public void removeKeyPoint(KeyPoint keyPoint) {  
 keyPoints.remove(keyPoint);  
 }  
}

## KeyPoint.java

package org.taimuraztibilov.taskmanager.base;  
  
import java.sql.SQLException;  
import java.time.LocalDate;  
import java.time.LocalTime;  
  
public class KeyPoint {  
 private final int id;  
 private final int taskId;  
 private String solution;  
 private LocalDate date;  
 private LocalTime timeSpent;  
 private DataEditor listenerOnEdit;  
  
 public KeyPoint(int id, int taskId, String solution, LocalDate date, LocalTime timeSpent) {  
 this.id = id;  
 this.taskId = taskId;  
 this.solution = solution;  
 this.date = date;  
 this.timeSpent = timeSpent;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public int getTaskId() {  
 return taskId;  
 }  
  
 public String getSolution() {  
 return solution;  
 }  
  
 public LocalDate getDate() {  
 return date;  
 }  
  
 public LocalTime getTimeSpent() {  
 return timeSpent;  
 }  
  
 public KeyPoint setListenerOnEdit(DataEditor listenerOnEdit) {  
 this.listenerOnEdit = listenerOnEdit;  
 return this;  
 }  
  
 public void setSolution(String solution) throws SQLException {  
 this.solution = solution;  
 listenerOnEdit.editData("keypoint", "solution", solution, id);  
 }  
  
 public void setDate(LocalDate date) throws SQLException {  
 this.date = date;  
 listenerOnEdit.editData("keypoint", "date\_closed", date.toString(), id);  
 }  
  
 public void setTimeSpent(LocalTime timeSpent) throws SQLException {  
 this.timeSpent = timeSpent;  
 listenerOnEdit.editData("keypoint", "time\_spent", timeSpent.toString(), id);  
 }  
}

## Label.java

package org.taimuraztibilov.taskmanager.base;  
  
import java.sql.SQLException;  
  
public class Label {  
 private final int id;  
 private String color;  
 private String title;  
 private DataEditor listenerOnEdit;  
  
 public Label(int id, String color, String title) {  
 this.id = id;  
 this.color = color;  
 this.title = title;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getColor() {  
 return color;  
 }  
  
 public String getTitle() {  
 return title;  
 }  
  
 public Label setListenerOnEdit(DataEditor listenerOnEdit) {  
 this.listenerOnEdit = listenerOnEdit;  
 return this;  
 }  
  
 public void setColor(String color) throws SQLException {  
 this.color = color;  
 listenerOnEdit.editData("label", "color", color, id);  
 }  
  
 public void setTitle(String title) throws SQLException {  
 this.title = title;  
 listenerOnEdit.editData("label", "title", title, id);  
 }  
}

## DataBaseManager.java

package org.taimuraztibilov.taskmanager.base;  
  
import com.intellij.openapi.components.Service;  
import org.sqlite.JDBC;  
  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Paths;  
import java.sql.\*;  
import java.time.LocalDate;  
import java.time.LocalDateTime;  
import java.time.LocalTime;  
import java.util.ArrayList;  
  
@Service  
public final class DataBaseManager implements DataEditor {  
 private static DataBaseManager *instance*;  
 private String connectionPath;  
 private Connection connection;  
  
 private DataBaseManager() throws SQLException {  
 this.connectionPath = "taskmanagerdb.sqlite"; *// this.connectionPath = "taskmanagerbase.sqlite";* DriverManager.*registerDriver*(new JDBC());  
 this.connection = DriverManager.*getConnection*("jdbc:sqlite:" + this.connectionPath);  
 connection.createStatement().executeUpdate("pragma foreign\_keys = on");  
 createProjectTable();  
 createMilestoneTable();  
 createTaskTable();  
 createKeyPointTable();  
 createLabelTable();  
 }  
  
 private DataBaseManager(String connectionPath) throws SQLException {  
 this.connectionPath = connectionPath;  
 DriverManager.*registerDriver*(new JDBC());  
 this.connection = DriverManager.*getConnection*("jdbc:sqlite:" + this.connectionPath);  
 connection.createStatement().executeUpdate("pragma foreign\_keys = on");  
 createProjectTable();  
 createMilestoneTable();  
 createTaskTable();  
 createKeyPointTable();  
 createLabelTable();  
 }  
  
 @Override  
 public void editData(String table, String column, String value, int id) throws SQLException {  
 boolean isInt = true;  
 try {  
 Integer.*parseInt*(value);  
 } catch (Exception e) {  
 isInt = false;  
 }  
 if (isInt) {  
 connection.createStatement().executeUpdate("update " + table + " set " +  
 column + " = " + value + " where id = " + id);  
 return;  
 }  
 connection.createStatement().executeUpdate("update " + table + " set " +  
 column + " = " + '\"' + value + '\"' + " where id = " + id);  
 *// TODO: 10.05.2020 переписать метод. если тип вносимых данных число, то пишем значение без кавычек, иначе в кавычках* }  
  
 public static synchronized DataBaseManager getInstance() throws SQLException {  
 if (*instance* == null)  
 *instance* = new DataBaseManager("taskmanagerdb.sqlite");  
 return *instance*;  
 }  
  
 public synchronized void changeConnection(String newPath) throws IOException, SQLException {  
 Files.*copy*(Paths.*get*(connectionPath), Paths.*get*(newPath));  
 connectionPath = newPath;  
 connection = DriverManager.*getConnection*("jdbc:sqlite:" + connectionPath);  
 }  
  
 private void createProjectTable() throws SQLException {  
 Statement statement = connection.createStatement();  
 statement.setQueryTimeout(10);  
  
 statement.executeUpdate("create table if not exists project (" +  
 "id integer primary key autoincrement not null," +  
 "title text not null," +  
 "description text," +  
 "state int not null)");  
 }  
  
 private void createMilestoneTable() throws SQLException {  
 Statement statement = connection.createStatement();  
 statement.setQueryTimeout(10);  
  
 statement.executeUpdate("create table if not exists milestone (" +  
 "id integer primary key autoincrement not null," +  
 "project\_id integer not null," +  
 "title text not null," +  
 "description text," +  
 "deadline text not null," +  
 "state int not null," +  
 "foreign key (project\_id) references project(id))");  
 }  
  
 private void createTaskTable() throws SQLException {  
 Statement statement = connection.createStatement();  
 statement.setQueryTimeout(10);  
  
 statement.executeUpdate("create table if not exists task (" +  
 "id integer primary key autoincrement not null," +  
 "milestone\_id integer not null," +  
 "title text not null," +  
 "description text," +  
 "created\_on text not null," +  
 "deadline text not null," +  
 "time\_estimated text," +  
 "time\_spent text," +  
 "state int not null," +  
 "foreign key (milestone\_id) references milestone(id))");  
 }  
  
 private void createKeyPointTable() throws SQLException {  
 Statement statement = connection.createStatement();  
 statement.setQueryTimeout(10);  
  
 statement.executeUpdate("create table if not exists keypoint (" +  
 "id integer primary key autoincrement not null," +  
 "task\_id integer not null," +  
 "solution text," +  
 "date\_closed text," +  
 "time\_spent text," +  
 "foreign key (task\_id) references task(id))");  
 }  
  
 private void createLabelTable() throws SQLException {  
 Statement statement = connection.createStatement();  
 statement.setQueryTimeout(20);  
  
 statement.executeUpdate("create table if not exists label (" +  
 "id integer primary key autoincrement not null," +  
 "title text not null," +  
 "color text not null)");  
 statement.executeUpdate("create table if not exists label\_task (" +  
 "label\_id integer not null," +  
 "task\_id integer not null," +  
 "foreign key (label\_id) references label(id)," +  
 "foreign key (task\_id) references task(id))");  
 }  
  
 public synchronized Project addProject(String title, String description, int state) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("insert into " +  
 "project (title, description, state) values (?, ?, ?)");  
 statement.setObject(1, title);  
 statement.setObject(2, description);  
 statement.setObject(3, state);  
 statement.executeUpdate();  
 ResultSet newId = connection.createStatement().executeQuery("select *max*(id) from project");  
 return new Project(newId.getInt(1), title, description, state).setListenerOnEdit(this);  
 }  
  
 public synchronized void deleteProject(int id) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("select id from milestone where project\_id = ?");  
 statement.setObject(1, id);  
 ResultSet result = statement.executeQuery();  
 while (result.next())  
 deleteMilestone(result.getInt(1));  
 statement = connection.prepareStatement("delete from project where id = ?");  
 statement.setObject(1, id);  
 statement.executeUpdate();  
 }  
  
 public synchronized Project getProject(int id) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("select *\** from project where id = ?");  
 statement.setObject(1, id);  
 ResultSet result = statement.executeQuery();  
 return new Project(  
 result.getInt("id"),  
 result.getString("title"),  
 result.getString("description"),  
 result.getInt("state")).setListenerOnEdit(this);  
 }  
  
 public synchronized ArrayList<Project> getProjects() throws SQLException {  
 ArrayList<Project> projects = new ArrayList<>();  
 Statement statement = connection.createStatement();  
 ResultSet result = statement.executeQuery("select id from project");  
 while (result.next())  
 projects.add(getProject(result.getInt("id")));  
  
 return projects;  
 }  
  
 public synchronized Milestone addMilestone(int projectId, String title, String description,  
 LocalDateTime deadline, int state)  
 throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("insert into milestone " +  
 "(project\_id, title, description, deadline, state) values (?, ?, ?, ?, ?)");  
 statement.setObject(1, projectId);  
 statement.setObject(2, title);  
 statement.setObject(3, description);  
 statement.setObject(4, deadline.toString());  
 statement.setObject(5, state);  
 statement.executeUpdate();  
 ResultSet newId = connection.createStatement().executeQuery("select *max*(id) from milestone");  
 return new Milestone(newId.getInt(1), projectId, title,  
 description, deadline, state).setListenerOnEdit(this);  
 }  
  
 public synchronized void deleteMilestone(int id) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("select id from task where milestone\_id = ?");  
 statement.setObject(1, id);  
 ResultSet result = statement.executeQuery();  
 while (result.next())  
 deleteTask(result.getInt(1));  
 statement = connection.prepareStatement("delete from milestone where id = ?");  
 statement.setObject(1, id);  
 statement.executeUpdate();  
 }  
  
 public synchronized Milestone getMilestone(int id) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("select *\** from milestone where id = ?");  
 statement.setObject(1, id);  
 ResultSet result = statement.executeQuery();  
 return new Milestone(  
 result.getInt("id"),  
 result.getInt("project\_id"),  
 result.getString("title"),  
 result.getString("description"),  
 LocalDateTime.*parse*(result.getString("deadline")),  
 result.getInt("state")).setListenerOnEdit(this);  
 }  
  
 public synchronized ArrayList<Milestone> getMilestones(int projectId) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("select id from milestone where project\_id = ?");  
 statement.setObject(1, projectId);  
 ResultSet result = statement.executeQuery();  
 ArrayList<Milestone> milestones = new ArrayList<>();  
 while (result.next()) {  
 milestones.add(getMilestone(result.getInt("id")));  
 }  
 return milestones;  
 }  
  
 public synchronized Label addLabel(String color, String title) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("insert into label (title, color) values (?, ?)");  
 statement.setObject(1, title);  
 statement.setObject(2, color);  
 statement.executeUpdate();  
 ResultSet newId = connection.createStatement().executeQuery("select *max*(id) from label");  
 return new Label(newId.getInt(1), color, title).setListenerOnEdit(this);  
 }  
  
 public synchronized void deleteLabel(int id) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("delete from label\_task where label\_id = ?");  
 statement.setObject(1, id);  
 statement.executeUpdate();  
 statement = connection.prepareStatement("delete from label where id = ?");  
 statement.setObject(1, id);  
 statement.executeUpdate();  
 }  
  
 public synchronized Label getLabel(int id) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("select *\** from label where id = ?");  
 statement.setObject(1, id);  
 ResultSet result = statement.executeQuery();  
 return new Label(  
 result.getInt("id"),  
 result.getString("color"),  
 result.getString("title")).setListenerOnEdit(this);  
 }  
  
 public synchronized ArrayList<Label> getLabels() throws SQLException {  
 ResultSet result = connection.createStatement().executeQuery("select id from label");  
 ArrayList<Label> labels = new ArrayList<>();  
 while (result.next()) {  
 labels.add(getLabel(result.getInt(1)));  
 }  
 return labels;  
 }  
  
 @Deprecated  
 public synchronized ArrayList<Label> getTaskLabels(int taskId, ArrayList<Label> labels) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("select label\_id from label\_task " +  
 "where task\_id = ?");  
 statement.setObject(1, taskId);  
 ResultSet result = statement.executeQuery();  
 ArrayList<Label> taskLabels = new ArrayList<>();  
 while (result.next()) {  
 int id = result.getInt(1);  
 labels.forEach(x -> {  
 if (id == x.getId())  
 taskLabels.add(x);  
 });  
 }  
 return taskLabels;  
 }  
  
 public synchronized Task addTask(int milestoneId, String title, String description, LocalDateTime deadline,  
 LocalTime timeEstimated, int state, ArrayList<Label> labels)  
 throws SQLException {  
 LocalDateTime createdOn = LocalDateTime.*now*();  
 PreparedStatement statement = connection.prepareStatement("insert into task " +  
 "(milestone\_id, title, description, created\_on, deadline, time\_estimated, time\_spent, state) " +  
 "values (?, ?, ?, ?, ?, ?, ?, ?)");  
 statement.setObject(1, milestoneId);  
 statement.setObject(2, title);  
 statement.setObject(3, description);  
 statement.setObject(4, createdOn.toString());  
 statement.setObject(5, deadline.toString());  
 statement.setObject(6, timeEstimated == null ? "" : timeEstimated.toString());  
 statement.setObject(7, LocalTime.*MIN*.toString());  
 statement.setObject(8, state);  
 statement.executeUpdate();  
 ResultSet newId = connection.createStatement().executeQuery("select *max*(id) from task");  
 Task newTask = new Task(newId.getInt(1), milestoneId, title, description, createdOn,  
 deadline, timeEstimated, LocalTime.*MIN*, state).setListenerOnEdit(this);  
 for (Label label : labels) {  
 statement = connection.prepareStatement("insert into label\_task (label\_id, task\_id) values (?, ?)");  
 statement.setObject(1, label.getId());  
 statement.setObject(2, newTask.getId());  
 statement.executeUpdate();  
 newTask.addLabel(label);  
 }  
 return newTask;  
 }  
  
 public synchronized void deleteTask(int id) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("delete from keypoint where task\_id = ?");  
 statement.setObject(1, id);  
 statement.executeUpdate();  
 statement = connection.prepareStatement("delete from label\_task where task\_id = ?");  
 statement.setObject(1, id);  
 statement.executeUpdate();  
 statement = connection.prepareStatement("delete from task where id = ?");  
 statement.setObject(1, id);  
 statement.executeUpdate();  
 }  
  
 public synchronized Task getTask(int id) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("select *\** from task where id = ?");  
 statement.setObject(1, id);  
 ResultSet result = statement.executeQuery();  
 LocalTime estimated = result.getString("time\_estimated") == null ||  
 result.getString("time\_estimated").isBlank() ? LocalTime.*MIN* :  
 LocalTime.*parse*(result.getString("time\_estimated"));  
 LocalTime spent = result.getString("time\_spent") == null ||  
 result.getString("time\_spent").isBlank() ? LocalTime.*MIN* :  
 LocalTime.*parse*(result.getString("time\_spent"));  
 Task task = new Task(id,  
 result.getInt("milestone\_id"),  
 result.getString("title"),  
 result.getString("description"),  
 LocalDateTime.*parse*(result.getString("created\_on")),  
 LocalDateTime.*parse*(result.getString("deadline")),  
 estimated,  
 spent,  
 result.getInt("state")).setListenerOnEdit(this);  
 statement = connection.prepareStatement("select label\_id from label\_task where task\_id = ?");  
 statement.setObject(1, task.getId());  
 result = statement.executeQuery();  
 if (result.next())  
 task.addLabel(getLabel(result.getInt(1)));  
 return task;  
 }  
  
 public synchronized ArrayList<Task> getTasks(int milestoneId) throws SQLException {  
 ArrayList<Task> tasks = new ArrayList<>();  
 PreparedStatement statement = connection.prepareStatement("select id from task where milestone\_id = ?");  
 statement.setObject(1, milestoneId);  
 ResultSet result = statement.executeQuery();  
 while (result.next())  
 tasks.add(getTask(result.getInt("id")));  
 return tasks;  
 }  
  
 public synchronized KeyPoint addKeyPoint(int taskId, String solution, LocalDate date) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("insert into keypoint " +  
 "(task\_id, solution, date\_closed, time\_spent) values (?, ?, ?, ?)");  
 statement.setObject(1, taskId);  
 statement.setObject(2, solution);  
 statement.setObject(3, date.toString());  
 statement.setObject(4, LocalTime.*MIN*.toString());  
 statement.executeUpdate();  
 ResultSet result = connection.createStatement().executeQuery("select *max*(id) from keypoint");  
 KeyPoint keyPoint = new KeyPoint(  
 result.getInt(1),  
 taskId,  
 solution,  
 date,  
 LocalTime.*MIN*).setListenerOnEdit(this);  
 return keyPoint;  
 }  
  
 public synchronized void deleteKeyPoint(int id) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("delete from keypoint where id = ?");  
 statement.setObject(1, id);  
 statement.executeUpdate();  
 }  
  
 public synchronized KeyPoint getKeyPoint(int id) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("select *\** from keypoint where id = ?");  
 statement.setObject(1, id);  
 ResultSet result = statement.executeQuery();  
 LocalDate date = LocalDate.*parse*(result.getString("date"));  
 LocalTime spent = result.getString("time\_spent") == null ||  
 result.getString("time\_spent").isBlank() ? LocalTime.*MIN* :  
 LocalTime.*parse*(result.getString("time\_spent"));  
 return new KeyPoint(id,  
 result.getInt("task\_id"),  
 result.getString("solution"),  
 date,  
 spent).setListenerOnEdit(this);  
 }  
  
 public synchronized ArrayList<KeyPoint> getKeyPoints(int taskId) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("select id from keypoint where task\_id = ?");  
 statement.setObject(1, taskId);  
 ResultSet result = statement.executeQuery();  
 ArrayList<KeyPoint> keyPoints = new ArrayList<>();  
 while (result.next())  
 keyPoints.add(getKeyPoint(result.getInt(1)));  
 return keyPoints;  
 }  
  
 public synchronized void addLabelToTask(int labelId, int taskId) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("insert into label\_task (label\_id, task\_id) " +  
 "values (?, ?)");  
 statement.setObject(1, labelId);  
 statement.setObject(2, taskId);  
 statement.executeUpdate();  
 }  
  
 public synchronized void removeLabelFromTask(int labelId, int taskId) throws SQLException {  
 PreparedStatement statement = connection.prepareStatement("delete from label\_task " +  
 "where (label\_id = ?, task\_id = ?)");  
 statement.setObject(1, labelId);  
 statement.setObject(2, taskId);  
 statement.executeUpdate();  
 }  
  
 public synchronized ResultSet getReportData(LocalDate from, LocalDate to, int projectId) throws SQLException {  
 return connection.createStatement().executeQuery(  
 "select t.title\_m, t.title\_t, k.date\_closed, k.solution, k.time\_spent " +  
 "from keypoint k " +  
 "inner join " +  
 "(select ta.id, ta.title as title\_t, m.title as title\_m " +  
 "from task ta " +  
 "inner join milestone m on ta.milestone\_id = m.id and m.project\_id = " + projectId + ") t " +  
 "on t.id = k.task\_id " +  
 "where *date*(k.date\_closed) >= *date*(" + '\"' + from.toString() + '\"' + ") and " +  
 "*date*(k.date\_closed) <= *date*(" + '\"' + to.toString() + '\"' + ") " +  
 "order by 1, 2, 3");  
 }  
}

## TimeManager.java

package org.taimuraztibilov.taskmanager.base;  
  
import com.intellij.openapi.components.Service;  
  
import java.sql.SQLException;  
import java.time.Duration;  
import java.time.LocalDateTime;  
import java.time.LocalTime;  
  
@Service  
public class TimeManager {  
 private static TimeManager *instance*;  
 private int taskId;  
 private LocalDateTime start;  
  
 private TimeManager() {  
 taskId = -1;  
 }  
  
 public static synchronized TimeManager getInstance() {  
 if (*instance* == null)  
 *instance* = new TimeManager();  
 return *instance*;  
 }  
  
 public synchronized boolean trackKeyPoint(int id) throws SQLException {  
 if (taskId != -1)  
 return false;  
 taskId = id;  
 DataBaseManager.*getInstance*().editData("task", "state", Integer.*toString*(States.*IN\_PROGRESS*), id);  
 start = LocalDateTime.*now*();  
 return true;  
 }  
  
 public synchronized LocalTime stopTracking() throws SQLException {  
 DataBaseManager.*getInstance*().editData("task", "state", Integer.*toString*(States.*OPEN*), taskId);  
 if (taskId == -1)  
 return LocalTime.*MIN*;  
 LocalTime result = LocalTime.*MIN*.plus(Duration.*between*(start, LocalDateTime.*now*()));  
 start = LocalDateTime.*now*();  
 taskId = -1;  
 return result;  
 }  
  
 public int getTaskId() {  
 return taskId;  
 }  
}

## ReportManager.java

package org.taimuraztibilov.taskmanager.base;  
  
import com.intellij.openapi.components.Service;  
import com.opencsv.CSVWriter;  
  
import java.io.File;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.time.LocalDate;  
import java.time.LocalDateTime;  
import java.time.LocalTime;  
import java.time.format.DateTimeFormatter;  
  
@Service  
public final class ReportManager {  
 private static ReportManager *instance*;  
  
 private ReportManager() {  
 }  
  
 public static synchronized ReportManager getInstance() {  
 if (*instance* == null)  
 *instance* = new ReportManager();  
 return *instance*;  
 }  
  
 public String generateReport(String projectPath, LocalDate from, LocalDate to, int projectId,  
 String organisation, String developerName) throws SQLException {  
 try {  
 String path = projectPath;  
 if (projectPath.charAt(projectPath.length() - 1) != '\\' && projectPath.charAt(projectPath.length() - 1) != '/')  
 path += "/";  
 path += "Report\_on\_" + LocalDate.*now*() + '-' +  
 LocalTime.*now*().getHour() + '-' +  
 LocalTime.*now*().getMinute() + '-' +  
 LocalTime.*now*().getSecond() + ".csv";  
 new File(path).createNewFile();  
 CSVWriter writer = new CSVWriter(new FileWriter(path));  
 writer.writeNext(new String[]{"Отчет по проекту с " + from.toString() + " по " + to.toString(), "", "", "", ""});  
 writer.writeNext(new String[]{  
 "Наименование проекта: ", DataBaseManager.*getInstance*().getProject(projectId).getTitle(), "", "", ""  
 });  
 writer.writeNext(new String[]{  
 "Разработчик: ", developerName, "", "", ""  
 });  
 writer.writeNext(new String[]{  
 "Организация: ", organisation, "", "", ""  
 });  
 writer.writeNext(new String[]{"", "", "", "", ""});  
 writer.writeNext(new String[]{  
 "Веха", "Задача", "Дата", "Комментарий", "Затраченное время"  
 });  
 ResultSet data = DataBaseManager.*getInstance*().getReportData(from, to, projectId);  
 while (data.next())  
 writer.writeNext(new String[]{  
 data.getString(1),  
 data.getString(2),  
 data.getString(3),  
 data.getString(4),  
 data.getString(5)  
 });  
 writer.close();  
 return "Отчет был успешно создан. Путь: " + path;  
 } catch (IOException e) {  
 return e.getLocalizedMessage();  
 }  
 }  
}

## PluginManagerService.java

package org.taimuraztibilov.taskmanager.base;  
  
import com.intellij.openapi.components.Service;  
  
@Service  
public final class PluginManagerService {  
 private int trackingProject;  
 private int trackingMilestone;  
 private int trackingTask;  
 private static PluginManagerService *instance*;  
  
 private PluginManagerService() {  
 trackingMilestone = -1;  
 trackingProject = -1;  
 trackingTask = -1;  
 }  
  
 public static synchronized PluginManagerService getInstance() {  
 if (*instance* == null)  
 *instance* = new PluginManagerService();  
 return *instance*;  
 }  
  
 public synchronized int getTrackingProject() {  
 return trackingProject;  
 }  
  
 public synchronized void setTrackingProject(int trackingProject) {  
 this.trackingProject = trackingProject;  
 }  
  
 public synchronized int getTrackingMilestone() {  
 return trackingMilestone;  
 }  
  
 public synchronized void setTrackingMilestone(int trackingMilestone) {  
 this.trackingMilestone = trackingMilestone;  
 }  
  
 public synchronized int getTrackingTask() {  
 return trackingTask;  
 }  
  
 public synchronized void setTrackingTask(int trackingTask) {  
 this.trackingTask = trackingTask;  
 }  
}

## AddDataFormBuilder.java

package org.taimuraztibilov.taskmanager.ui;  
  
import com.intellij.openapi.project.Project;  
import com.intellij.openapi.ui.ComboBox;  
import com.intellij.ui.JBColor;  
import com.intellij.ui.components.\*;  
import com.intellij.uiDesigner.core.GridConstraints;  
import com.intellij.uiDesigner.core.GridLayoutManager;  
import com.intellij.util.ui.JBUI;  
import com.intellij.util.ui.UIUtil;  
import org.jdesktop.swingx.JXDatePicker;  
import org.taimuraztibilov.taskmanager.base.\*;  
import org.taimuraztibilov.taskmanager.base.Label;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.sql.Date;  
import java.sql.SQLException;  
import java.time.LocalDate;  
import java.time.LocalDateTime;  
import java.time.LocalTime;  
import java.util.ArrayList;  
import java.util.TimeZone;  
  
public class AddDataFormBuilder {  
  
 public static synchronized void addProjectByUser(Project project) {  
 JFrame createProject = new JFrame("Add new project");  
 GridConstraints constraints = new GridConstraints();  
 JBPanel form = new JBPanel();  
 form.setLayout(new GridLayoutManager(4, 1,  
 JBUI.*insets*(12, 20), 20, 20));  
  
 JBPanel title = new JBPanel(new BorderLayout());  
 JBLabel label = new JBLabel("Title: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 title.add(label);  
 JBTextField getTitle = new JBTextField();  
 getTitle.setFont(label.getFont());  
 getTitle.setColumns(50);  
 title.add(getTitle, BorderLayout.*EAST*);  
 form.add(title, constraints);  
 constraints.setRow(1);  
  
 JBPanel description = new JBPanel(new BorderLayout());  
 label = new JBLabel("Description: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 description.add(label);  
 JBTextArea getDescription = new JBTextArea(6, 50);  
 getDescription.setLineWrap(true);  
 getDescription.setWrapStyleWord(true);  
 getDescription.setFont(label.getFont());  
 description.add(new JBScrollPane(getDescription), BorderLayout.*EAST*);  
 form.add(description, constraints);  
 constraints.setRow(2);  
  
 JBPanel getState = new JBPanel(new BorderLayout());  
 label = new JBLabel("State: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 getState.add(label);  
 DefaultComboBoxModel<String> comboBoxModel = new DefaultComboBoxModel<>(States.*getArray*());  
 ComboBox<String> comboBox = new ComboBox<>(comboBoxModel);  
 comboBox.setFont(label.getFont());  
 comboBox.setSelectedIndex(1);  
 getState.add(comboBox, BorderLayout.*EAST*);  
 form.add(getState, constraints);  
 constraints.setRow(0);  
  
 JBPanel buttons = new JBPanel(new GridLayoutManager(1, 2,  
 JBUI.*insets*(10, 10), 0, 0));  
 JButton createButton = new JButton("Create");  
 JButton cancelButton = new JButton("Cancel");  
 cancelButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 createProject.dispose();  
 }  
 });  
 createButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 String title = getTitle.getText();  
 String description = getDescription.getText();  
 int state = States.*toInt*((String) comboBox.getSelectedItem());  
 PluginManagerService.*getInstance*().setTrackingProject(  
 DataBaseManager.*getInstance*().addProject(title, description, state).getId());  
 *// TODO: 08.05.2020 Notification that project was added correctly* createProject.dispose();  
 } catch (SQLException throwables) {  
 *// TODO: 08.05.2020 Show notification about exception* }  
 }  
 });  
 buttons.add(createButton, constraints);  
 constraints.setColumn(1);  
 buttons.add(cancelButton, constraints);  
 constraints.setColumn(0);  
 constraints.setRow(3);  
 form.add(buttons, constraints);  
 createProject.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 createProject.setContentPane(form);  
 createProject.pack();  
 createProject.setVisible(true);  
 }  
  
 public static synchronized void addMilestoneByUser(int projectId) {  
 JFrame createProject = new JFrame("Add new milestone");  
 GridConstraints constraints = new GridConstraints();  
 JBPanel form = new JBPanel();  
 form.setLayout(new GridLayoutManager(5, 1,  
 JBUI.*insets*(12, 20), 20, 20));  
  
 JBPanel title = new JBPanel(new BorderLayout());  
 JBLabel label = new JBLabel("Title: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 title.add(label);  
 JBTextField getTitle = new JBTextField("");  
 getTitle.setFont(label.getFont());  
 getTitle.setColumns(50);  
 title.add(getTitle, BorderLayout.*EAST*);  
 form.add(title, constraints);  
 constraints.setRow(1);  
  
 JBPanel description = new JBPanel(new BorderLayout());  
 label = new JBLabel("Description: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 description.add(label);  
 JBTextArea getDescription = new JBTextArea(6, 50);  
 getDescription.setLineWrap(true);  
 getDescription.setWrapStyleWord(true);  
 getDescription.setFont(label.getFont());  
 description.add(new JBScrollPane(getDescription), BorderLayout.*EAST*);  
 form.add(description, constraints);  
 constraints.setRow(2);  
  
 JBPanel deadline = new JBPanel(new BorderLayout());  
 label = new JBLabel("Deadline: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 deadline.add(label);  
 JXDatePicker getDeadline = new JXDatePicker();  
 getDeadline.setTimeZone(TimeZone.*getDefault*());  
 getDeadline.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if (Date.*valueOf*(LocalDate.*now*()).after(getDeadline.getDate()))  
 getDeadline.setDate(Date.*valueOf*(LocalDate.*now*()));  
 }  
 });  
 deadline.add(getDeadline, BorderLayout.*EAST*);  
 form.add(deadline, constraints);  
 constraints.setRow(3);  
  
 JBPanel getState = new JBPanel(new BorderLayout());  
 label = new JBLabel("State: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 getState.add(label);  
 DefaultComboBoxModel<String> comboBoxModel = new DefaultComboBoxModel<>(States.*getArray*());  
 ComboBox<String> comboBox = new ComboBox<>(comboBoxModel);  
 comboBox.setFont(label.getFont());  
 comboBox.setSelectedIndex(1);  
 getState.add(comboBox, BorderLayout.*EAST*);  
 form.add(getState, constraints);  
 constraints.setRow(0);  
  
 JBPanel buttons = new JBPanel(new GridLayoutManager(1, 2,  
 JBUI.*insets*(10, 10), 0, 0));  
 JButton createButton = new JButton("Create");  
 JButton cancelButton = new JButton("Cancel");  
 cancelButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 createProject.dispose();  
 }  
 });  
 createButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 if (getDeadline.getDate() == null) {  
 *// TODO: 10.05.2020 show notification* return;  
 }  
 String title = getTitle.getText();  
 String description = getDescription.getText();  
 LocalDateTime deadline = LocalDateTime.*parse*(getDeadline.getDate().toInstant().toString()  
 .substring(0, getDeadline.getDate().toInstant().toString().length() - 1));  
 int state = States.*toInt*((String) comboBox.getSelectedItem());  
 PluginManagerService.*getInstance*().setTrackingMilestone(  
 DataBaseManager.*getInstance*()  
 .addMilestone(projectId, title, description, deadline, state).getId());  
 *// TODO: 08.05.2020 Notification that project was added correctly* createProject.dispose();  
 } catch (SQLException throwables) {  
 *// TODO: 08.05.2020 Show notification about exception* }  
 }  
 });  
 buttons.add(createButton, constraints);  
 constraints.setColumn(1);  
 buttons.add(cancelButton, constraints);  
 constraints.setColumn(0);  
 constraints.setRow(4);  
 form.add(buttons, constraints);  
 createProject.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 createProject.setContentPane(form);  
 createProject.pack();  
 createProject.setVisible(true);  
 }  
  
 public static synchronized void addTaskByUser(int milestoneId) {  
 JFrame createProject = new JFrame("Add new task");  
 GridConstraints constraints = new GridConstraints();  
 JBPanel form = new JBPanel();  
 form.setLayout(new GridLayoutManager(6, 1,  
 JBUI.*insets*(12, 20), 20, 20));  
  
 JBPanel title = new JBPanel(new BorderLayout());  
 JBLabel label = new JBLabel("Title: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 title.add(label);  
 JBTextField getTitle = new JBTextField("");  
 getTitle.setFont(label.getFont());  
 getTitle.setColumns(50);  
 title.add(getTitle, BorderLayout.*EAST*);  
 form.add(title, constraints);  
 constraints.setRow(1);  
  
 JBPanel description = new JBPanel(new BorderLayout());  
 label = new JBLabel("Description: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 description.add(label);  
 JBTextArea getDescription = new JBTextArea(6, 50);  
 getDescription.setLineWrap(true);  
 getDescription.setWrapStyleWord(true);  
 getDescription.setFont(label.getFont());  
 description.add(new JBScrollPane(getDescription), BorderLayout.*EAST*);  
 form.add(description, constraints);  
 constraints.setRow(2);  
 ;  
  
 JBPanel deadline = new JBPanel(new BorderLayout());  
 label = new JBLabel("Deadline: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 deadline.add(label);  
 JXDatePicker getDeadline = new JXDatePicker();  
 getDeadline.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if (Date.*valueOf*(LocalDate.*now*()).after(getDeadline.getDate()))  
 getDeadline.setDate(Date.*valueOf*(LocalDate.*now*()));  
 }  
 });  
 deadline.add(getDeadline, BorderLayout.*EAST*);  
 form.add(deadline, constraints);  
 constraints.setRow(3);  
  
 JBPanel getState = new JBPanel(new BorderLayout());  
 label = new JBLabel("State: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 getState.add(label);  
 DefaultComboBoxModel<String> comboBoxModel = new DefaultComboBoxModel<>(States.*getArray*());  
 ComboBox<String> comboBox = new ComboBox<>(comboBoxModel);  
 comboBox.setFont(label.getFont());  
 comboBox.setSelectedIndex(1);  
 getState.add(comboBox, BorderLayout.*EAST*);  
 form.add(getState, constraints);  
 constraints.setRow(4);  
  
 JBPanel getLabels = new JBPanel(new BorderLayout());  
 label = new JBLabel("Label: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 getState.add(label);  
 ComboBox<String> labels = new ComboBox<>();  
 ArrayList<Label> arrayList = new ArrayList<>();  
 try {  
 arrayList = DataBaseManager.*getInstance*().getLabels();  
 } catch (SQLException throwables) {  
 *// TODO: 08.05.2020 Show notification about exception* }  
 for (Label dataLabel : arrayList) {  
 labels.addItem(dataLabel.getTitle());  
 }  
 labels.addItem("None");  
 labels.setSelectedIndex(0);  
 getLabels.add(labels, BorderLayout.*EAST*);  
 form.add(getLabels, constraints);  
 constraints.setRow(0);  
  
 JBPanel buttons = new JBPanel(new GridLayoutManager(1, 2,  
 JBUI.*insets*(10, 10), 0, 0));  
 JButton createButton = new JButton("Create");  
 JButton cancelButton = new JButton("Cancel");  
 cancelButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 createProject.dispose();  
 }  
 });  
 ArrayList<Label> finalArrayList = arrayList;  
 createButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 if (getDeadline.getDate() == null) {  
 *// TODO: 10.05.2020 show notification* return;  
 }  
 String title = getTitle.getText();  
 String description = getDescription.getText();  
 LocalDateTime deadline = LocalDateTime.*parse*(getDeadline.getDate().toInstant().toString()  
 .substring(0, getDeadline.getDate().toInstant().toString().length() - 1));  
 int state = States.*toInt*((String) comboBox.getSelectedItem());  
 Label label1 = null;  
 if (labels.getSelectedItem() != "None")  
 label1 = finalArrayList.get(labels.getSelectedIndex());  
 ArrayList<Label> labels1 = new ArrayList<>();  
 if (label1 != null)  
 labels1.add(label1);  
 PluginManagerService.*getInstance*().setTrackingTask(  
 DataBaseManager.*getInstance*()  
 .addTask(milestoneId, title, description, deadline, LocalTime.*MIN*, state, labels1)  
 .getId());  
 *// TODO: 08.05.2020 Notification that project was added correctly* createProject.dispose();  
 } catch (SQLException throwables) {  
 *// TODO: 08.05.2020 Show notification about exception* }  
 }  
 });  
 buttons.add(createButton, constraints);  
 constraints.setColumn(1);  
 buttons.add(cancelButton, constraints);  
 constraints.setColumn(0);  
 constraints.setRow(5);  
 form.add(buttons, constraints);  
 createProject.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 createProject.setContentPane(form);  
 createProject.pack();  
 createProject.setVisible(true);  
 }  
  
 public static synchronized void addLabelByUser() {  
 JFrame createProject = new JFrame("Add new label");  
 GridConstraints constraints = new GridConstraints();  
 JBPanel form = new JBPanel();  
 form.setLayout(new GridLayoutManager(2, 1,  
 JBUI.*insets*(12, 20), 20, 20));  
  
 JBPanel title = new JBPanel(new BorderLayout());  
 JBLabel label = new JBLabel("Title: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 title.add(label);  
 JBTextField getTitle = new JBTextField("");  
 getTitle.setFont(label.getFont());  
 getTitle.setColumns(50);  
 title.add(getTitle, BorderLayout.*EAST*);  
 form.add(title, constraints);  
  
 JBPanel buttons = new JBPanel(new GridLayoutManager(1, 2,  
 JBUI.*insets*(10, 10), 0, 0));  
 JButton createButton = new JButton("Create");  
 JButton cancelButton = new JButton("Cancel");  
 cancelButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 createProject.dispose();  
 }  
 });  
 createButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 String title = getTitle.getText();  
 String color = String.*valueOf*(JBColor.*WHITE*.getRGB());  
 DataBaseManager.*getInstance*().addLabel(color, title);  
 *// TODO: 08.05.2020 Notification that project was added correctly* createProject.dispose();  
 } catch (SQLException throwables) {  
 *// TODO: 08.05.2020 Show notification about exception* }  
 }  
 });  
 buttons.add(createButton, constraints);  
 constraints.setColumn(1);  
 buttons.add(cancelButton, constraints);  
 constraints.setColumn(0);  
 constraints.setRow(1);  
 form.add(buttons, constraints);  
 createProject.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 createProject.setContentPane(form);  
 createProject.pack();  
 createProject.setVisible(true);  
 }  
  
 public static synchronized void addKeyPointByUser(LocalTime timeSpent, int taskId) {  
 JFrame createProject = new JFrame("Add new keypoint");  
 GridConstraints constraints = new GridConstraints();  
 JBPanel form = new JBPanel();  
 form.setLayout(new GridLayoutManager(2, 1,  
 JBUI.*insets*(12, 20), 20, 20));  
  
 JBPanel description = new JBPanel(new BorderLayout());  
 JBLabel label = new JBLabel("Description of solution: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 description.add(label);  
 JBTextArea getDescription = new JBTextArea(6, 50);  
 getDescription.setLineWrap(true);  
 getDescription.setWrapStyleWord(true);  
 getDescription.setFont(label.getFont());  
 description.add(new JBScrollPane(getDescription), BorderLayout.*EAST*);  
 form.add(description, constraints);  
  
 JBPanel buttons = new JBPanel(new GridLayoutManager(1, 2,  
 JBUI.*insets*(10, 10), 0, 0));  
 JButton createButton = new JButton("Create");  
 JButton cancelButton = new JButton("Cancel");  
 cancelButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 createProject.dispose();  
 }  
 });  
 createButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 String title = getDescription.getText();  
 DataBaseManager.*getInstance*().addKeyPoint(taskId, title, LocalDate.*now*()).setTimeSpent(timeSpent);  
 *// TODO: 08.05.2020 Notification that project was added correctly* createProject.dispose();  
 } catch (SQLException throwables) {  
 *// TODO: 08.05.2020 Show notification about exception* }  
 }  
 });  
 buttons.add(createButton, constraints);  
 constraints.setColumn(1);  
 buttons.add(cancelButton, constraints);  
 constraints.setColumn(0);  
 constraints.setRow(1);  
 form.add(buttons, constraints);  
 createProject.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 createProject.setContentPane(form);  
 createProject.pack();  
 createProject.setVisible(true);  
 }  
  
 public static synchronized void generateReport(String projectPath) {  
 JFrame createProject = new JFrame("Create new report");  
 GridConstraints constraints = new GridConstraints();  
 JBPanel form = new JBPanel();  
 form.setLayout(new GridLayoutManager(5, 1,  
 JBUI.*insets*(12, 20), 20, 20));  
  
 JBPanel title = new JBPanel(new BorderLayout());  
 JBLabel label = new JBLabel("Full name: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 title.add(label);  
 JBTextField getTitle = new JBTextField("");  
 getTitle.setFont(label.getFont());  
 getTitle.setColumns(50);  
 title.add(getTitle, BorderLayout.*EAST*);  
 form.add(title, constraints);  
 constraints.setRow(1);  
  
 JBPanel organ = new JBPanel(new BorderLayout());  
 label = new JBLabel("Organization: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 organ.add(label);  
 JBTextField getOrg = new JBTextField("");  
 getOrg.setFont(label.getFont());  
 getOrg.setColumns(50);  
 organ.add(getOrg, BorderLayout.*EAST*);  
 form.add(organ, constraints);  
 constraints.setRow(2);  
  
 JBPanel deadline = new JBPanel(new BorderLayout());  
 label = new JBLabel("Date from: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 deadline.add(label);  
 JXDatePicker getDeadline = new JXDatePicker();  
 deadline.add(getDeadline, BorderLayout.*EAST*);  
 form.add(deadline, constraints);  
 constraints.setRow(3);  
  
 JBPanel dateTo = new JBPanel(new BorderLayout());  
 label = new JBLabel("Date to: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 dateTo.add(label);  
 JXDatePicker datePicker = new JXDatePicker();  
 dateTo.add(datePicker, BorderLayout.*EAST*);  
 form.add(dateTo, constraints);  
 constraints.setRow(0);  
  
 JBPanel buttons = new JBPanel(new GridLayoutManager(1, 2,  
 JBUI.*insets*(10, 10), 0, 0));  
 JButton createButton = new JButton("Create");  
 JButton cancelButton = new JButton("Cancel");  
 cancelButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 createProject.dispose();  
 }  
 });  
 createButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 if (getDeadline.getDate() == null || datePicker.getDate() == null) {  
 *// TODO: 10.05.2020 show notification* return;  
 }  
 String name = getTitle.getText();  
 String organ = getOrg.getText();  
 LocalDateTime from = LocalDateTime.*parse*(getDeadline.getDate().toInstant().toString()  
 .substring(0, getDeadline.getDate().toInstant().toString().length() - 1));  
 LocalDateTime to = LocalDateTime.*parse*(datePicker.getDate().toInstant().toString()  
 .substring(0, datePicker.getDate().toInstant().toString().length() - 1));  
 ReportManager.*getInstance*().generateReport(projectPath, from.toLocalDate(), to.toLocalDate(),  
 PluginManagerService.*getInstance*().getTrackingProject(), organ, name);  
 *// TODO: 08.05.2020 Notification that project was added correctly* createProject.dispose();  
 } catch (SQLException throwables) {  
 *// TODO: 08.05.2020 Show notification about exception* }  
 }  
 });  
 buttons.add(createButton, constraints);  
 constraints.setColumn(1);  
 buttons.add(cancelButton, constraints);  
 constraints.setColumn(0);  
 constraints.setRow(4);  
 form.add(buttons, constraints);  
 createProject.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 createProject.setContentPane(form);  
 createProject.pack();  
 createProject.setVisible(true);  
 }  
}

## ShowDataFormBuilder.java

package org.taimuraztibilov.taskmanager.ui;  
  
import com.intellij.ui.components.\*;  
import com.intellij.uiDesigner.core.GridConstraints;  
import com.intellij.uiDesigner.core.GridLayoutManager;  
import com.intellij.util.ui.JBUI;  
import com.intellij.util.ui.UIUtil;  
import org.taimuraztibilov.taskmanager.base.\*;  
  
import javax.swing.\*;  
import javax.swing.event.ListSelectionEvent;  
import javax.swing.event.ListSelectionListener;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;  
import java.sql.SQLException;  
import java.time.LocalDateTime;  
import java.time.LocalTime;  
import java.util.ArrayList;  
  
public class ShowDataFormBuilder {  
 private static ArrayList<Project> *projects*;  
 private static ArrayList<Milestone> *milestones*;  
 private static ArrayList<Task> *tasks*;  
 private static JBList<String> *list*;  
 private static int *lastSelectedList* = -1;  
  
 private static JBList<String> getProjectList() throws SQLException {  
 *projects* = DataBaseManager.*getInstance*().getProjects();  
 DefaultListModel<String> model = new DefaultListModel<>();  
 for (int i = 0; i < *projects*.size(); i++)  
 model.add(i, *projects*.get(i).getTitle());  
 JBList<String> projectList = new JBList<>(model);  
 projectList.setSelectionMode(ListSelectionModel.*SINGLE\_SELECTION*);  
 return projectList;  
 }  
  
 private static JBList<String> getMilestoneList(int projectId) throws SQLException {  
 *milestones* = DataBaseManager.*getInstance*().getMilestones(projectId);  
 DefaultListModel<String> model = new DefaultListModel<>();  
 for (int i = 0; i < *milestones*.size(); i++)  
 model.add(i, *milestones*.get(i).getTitle());  
 JBList<String> milestoneList = new JBList<>(model);  
 milestoneList.setSelectionMode(ListSelectionModel.*SINGLE\_SELECTION*);  
 return milestoneList;  
 }  
  
 private static JBList<String> getTaskList(int milestoneId) throws SQLException {  
 *tasks* = DataBaseManager.*getInstance*().getTasks(milestoneId);  
 DefaultListModel<String> model = new DefaultListModel<>();  
 for (int i = 0; i < *tasks*.size(); i++)  
 model.add(i, *tasks*.get(i).getTitle());  
 JBList<String> tasksList = new JBList<>(model);  
 tasksList.setSelectionMode(ListSelectionModel.*SINGLE\_SELECTION*);  
 return tasksList;  
 }  
  
*// public static synchronized void showTasksInformation() {  
// JFrame showData = new JFrame("Choose task to track");  
// GridConstraints constraints = new GridConstraints();  
// JBPanel form = new JBPanel();  
// form.setLayout(new GridLayoutManager(3, 3,  
// JBUI.insets(20, 8), 8, 20));  
// JBLabel label = new JBLabel("Project",  
// UIUtil.ComponentStyle.REGULAR, UIUtil.FontColor.NORMAL);  
// label.setHorizontalAlignment(SwingConstants.LEFT);  
// form.add(label, constraints);  
// constraints.setColumn(1);  
// label = new JBLabel("Milestone",  
// UIUtil.ComponentStyle.REGULAR, UIUtil.FontColor.NORMAL);  
// label.setHorizontalAlignment(SwingConstants.LEFT);  
// form.add(label, constraints);  
// constraints.setColumn(2);  
// label = new JBLabel("Task",  
// UIUtil.ComponentStyle.REGULAR, UIUtil.FontColor.NORMAL);  
// label.setHorizontalAlignment(SwingConstants.LEFT);  
// form.add(label, constraints);  
// constraints.setColumn(0);  
// constraints.setRow(1);  
//  
// JBList<String> component = new JBList<>();  
// try {  
// component = getProjectList();  
// } catch (SQLException throwables) {  
// // TODO: 09.05.2020 Notification  
// }  
// component.addListSelectionListener(new ListSelectionListener() {  
// @Override  
// public void valueChanged(ListSelectionEvent e) {  
// int selected = ((JBList<?>) e.getSource()).getSelectedIndex();  
// PluginManagerService.getInstance().setTrackingProject(selected);  
// PluginManagerService.getInstance().setTrackingTask(-1);  
// PluginManagerService.getInstance().setTrackingMilestone(-1);  
// GridConstraints constraints = new GridConstraints();  
// constraints.setRow(1);  
// constraints.setColumn(1);  
// JBList<String> component = new JBList<>();  
// try {  
// component = getMilestoneList(selected);  
// } catch (SQLException throwables) {  
// // TODO: 09.05.2020 Notification  
// }  
// component.addListSelectionListener(new ListSelectionListener() {  
// @Override  
// public void valueChanged(ListSelectionEvent e) {  
// int selected = ((JBList<?>) e.getSource()).getSelectedIndex();  
// PluginManagerService.getInstance().setTrackingMilestone(selected);  
// PluginManagerService.getInstance().setTrackingTask(-1);  
// GridConstraints constraints = new GridConstraints();  
// constraints.setRow(1);  
// constraints.setColumn(2);  
// list = new JBList<>();  
// try {  
// list = getTaskList(selected);  
// } catch (SQLException throwables) {  
// // TODO: 09.05.2020 Notification  
// }  
// list.addMouseListener(new MouseAdapter() {  
// @Override  
// public void mouseClicked(MouseEvent e) {  
// if (e.getClickCount() == 2) {  
// int selected = ((JBList<?>) e.getSource()).locationToIndex(e.getPoint());  
// showTaskData(tasks.get(selected));  
// }  
// }  
// });  
// form.add(list, constraints);  
// }  
// });  
// component.addMouseListener(new MouseAdapter() {  
// @Override  
// public void mouseClicked(MouseEvent e) {  
// if (e.getClickCount() == 2) {  
// int selected = ((JBList<?>) e.getSource()).locationToIndex(e.getPoint());  
// showMilestoneData(milestones.get(selected));  
// }  
// }  
// });  
// form.add(component, constraints);  
// }  
// });  
// component.addMouseListener(new MouseAdapter() {  
// @Override  
// public void mouseClicked(MouseEvent e) {  
// if (e.getClickCount() == 2) {  
// int selected = ((JBList<?>) e.getSource()).locationToIndex(e.getPoint());  
// showProjectData(projects.get(selected));  
// }  
// }  
// });  
// form.add(component, constraints);  
// constraints.setColumn(0);  
// constraints.setRow(0);  
//  
// JBPanel buttons = new JBPanel(new GridLayoutManager(1, 2,  
// JBUI.insets(10, 10), 0, 0));  
// JButton createButton = new JButton("Track");  
// JButton cancelButton = new JButton("Cancel");  
// cancelButton.addActionListener(new ActionListener() {  
// @Override  
// public void actionPerformed(ActionEvent e) {  
// showData.dispose();  
// }  
// });  
// createButton.addActionListener(new ActionListener() {  
// @Override  
// public void actionPerformed(ActionEvent e) {  
// try {  
// if (PluginManagerService.getInstance().getTrackingTask() == -1)  
// return;  
// int onTrack = tasks.get(list.getSelectedIndex()).getId();  
// PluginManagerService.getInstance().setTrackingTask(onTrack);  
// LocalTime tracked = TimeManager.getInstance().stopTracking();  
// int trackedId = TimeManager.getInstance().getTaskId();  
// TimeManager.getInstance().trackKeyPoint(onTrack);  
// showData.dispose();  
// if (trackedId == -1) {  
// AddDataFormBuilder.addKeyPointByUser(tracked, trackedId);  
// }  
// } catch (SQLException throwables) {  
// // TODO: 08.05.2020 Show notification about exception  
// }  
// }  
// });  
// buttons.add(createButton, constraints);  
// constraints.setColumn(1);  
// buttons.add(cancelButton, constraints);  
// constraints.setColumn(2);  
// constraints.setRow(2);  
// form.add(buttons, constraints);  
// showData.setDefaultCloseOperation(WindowConstants.DISPOSE\_ON\_CLOSE);  
// showData.setContentPane(form);  
// showData.pack();  
// showData.setVisible(true);  
// }* public static synchronized void showProjects() {  
 JFrame showData = new JFrame("Choose project to track");  
 GridConstraints constraints = new GridConstraints();  
 JBPanel form = new JBPanel();  
 form.setLayout(new GridLayoutManager(3, 1,  
 JBUI.*insets*(20, 8), 8, 20));  
 JBLabel label = new JBLabel("Project",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 form.add(label, constraints);  
 constraints.setRow(1);  
 JBList<String> component = new JBList<>();  
 try {  
 component = *getProjectList*();  
 } catch (SQLException throwables) {  
 *// TODO: 09.05.2020 Notification* }  
 component.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 if (*projects*.size() == 0)  
 return;  
 if (e.getClickCount() == 2) {  
 int selected = ((JBList<?>) e.getSource()).locationToIndex(e.getPoint());  
 *showProjectData*(*projects*.get(selected));  
 }  
 }  
 });  
 component.addListSelectionListener(new ListSelectionListener() {  
 @Override  
 public void valueChanged(ListSelectionEvent e) {  
 int selected = ((JBList<?>) e.getSource()).getSelectedIndex();  
 PluginManagerService.*getInstance*().setTrackingProject(*projects*.get(selected).getId());  
 PluginManagerService.*getInstance*().setTrackingTask(-1);  
 PluginManagerService.*getInstance*().setTrackingMilestone(-1);  
 }  
 });  
 form.add(component, constraints);  
 constraints.setRow(2);  
 JButton track = new JButton("Track");  
 track.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 showData.dispose();  
 }  
 });  
 form.add(track, constraints);  
 showData.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 showData.setContentPane(form);  
 showData.pack();  
 showData.setVisible(true);  
 }  
  
 public static synchronized void showMilestones() {  
 JFrame showData = new JFrame("Choose milestone to track");  
 GridConstraints constraints = new GridConstraints();  
 JBPanel form = new JBPanel();  
 form.setLayout(new GridLayoutManager(3, 1,  
 JBUI.*insets*(20, 8), 8, 20));  
 JBLabel label = new JBLabel("Milestone",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 form.add(label, constraints);  
 constraints.setRow(1);  
 JBList<String> component = new JBList<>();  
 try {  
 component = *getMilestoneList*(PluginManagerService.*getInstance*().getTrackingProject());  
 } catch (SQLException throwables) {  
 *// TODO: 09.05.2020 Notification* }  
 component.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 if (*milestones*.size() == 0)  
 return;  
 if (e.getClickCount() == 2) {  
 int selected = ((JBList<?>) e.getSource()).locationToIndex(e.getPoint());  
 *showMilestoneData*(*milestones*.get(selected));  
 }  
 }  
 });  
 component.addListSelectionListener(new ListSelectionListener() {  
 @Override  
 public void valueChanged(ListSelectionEvent e) {  
 int selected = ((JBList<?>) e.getSource()).getSelectedIndex();  
 PluginManagerService.*getInstance*().setTrackingTask(-1);  
 PluginManagerService.*getInstance*().setTrackingMilestone(*milestones*.get(selected).getId());  
 }  
 });  
 form.add(component, constraints);  
 constraints.setRow(2);  
 JButton track = new JButton("Track");  
 track.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 showData.dispose();  
 }  
 });  
 form.add(track, constraints);  
 showData.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 showData.setContentPane(form);  
 showData.pack();  
 showData.setVisible(true);  
 }  
  
 public static synchronized void showTasks() {  
 JFrame showData = new JFrame("Choose task to track");  
 GridConstraints constraints = new GridConstraints();  
 JBPanel form = new JBPanel();  
 form.setLayout(new GridLayoutManager(3, 1,  
 JBUI.*insets*(20, 8), 8, 20));  
 JBLabel label = new JBLabel("Task",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 form.add(label, constraints);  
 constraints.setRow(1);  
 JBList<String> component = new JBList<>();  
 try {  
 component = *getTaskList*(PluginManagerService.*getInstance*().getTrackingMilestone());  
 } catch (SQLException throwables) {  
 *// TODO: 09.05.2020 Notification* }  
 component.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 if (*tasks*.size() == 0)  
 return;  
 if (e.getClickCount() == 2) {  
 int selected = ((JBList<?>) e.getSource()).locationToIndex(e.getPoint());  
 *showTaskData*(*tasks*.get(selected));  
 }  
 }  
 });  
 component.addListSelectionListener(new ListSelectionListener() {  
 @Override  
 public void valueChanged(ListSelectionEvent e) {  
 int selected = ((JBList<?>) e.getSource()).getSelectedIndex();  
 PluginManagerService.*getInstance*().setTrackingTask(*tasks*.get(selected).getId()); *// TODO: 10.05.2020* }  
 });  
 form.add(component, constraints);  
 *list* = component;  
 constraints.setRow(2);  
 JButton track = new JButton("Track");  
 track.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 if (PluginManagerService.*getInstance*().getTrackingTask() == -1 || *tasks*.size() == 0)  
 return;  
 int onTrack = *tasks*.get(*list*.getSelectedIndex()).getId();  
 PluginManagerService.*getInstance*().setTrackingTask(onTrack);  
 int trackedId = TimeManager.*getInstance*().getTaskId();  
 LocalTime tracked = TimeManager.*getInstance*().stopTracking();  
 TimeManager.*getInstance*().trackKeyPoint(onTrack);  
 showData.dispose();  
 if (trackedId != -1) {  
 AddDataFormBuilder.*addKeyPointByUser*(tracked, trackedId);  
 }  
 } catch (SQLException throwables) {  
 *// TODO: 08.05.2020 Show notification about exception* }  
 }  
 });  
 form.add(track, constraints);  
 showData.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 showData.setContentPane(form);  
 showData.pack();  
 showData.setVisible(true);  
 }  
  
 private static synchronized void showProjectData(Project project) {  
 JFrame createProject = new JFrame(project.getTitle());  
 GridConstraints constraints = new GridConstraints();  
 JBPanel form = new JBPanel();  
 form.setLayout(new GridLayoutManager(3, 1,  
 JBUI.*insets*(12, 20), 20, 20));  
  
 JBPanel title = new JBPanel(new BorderLayout());  
 JBLabel label = new JBLabel("Title: " + project.getTitle(),  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 title.add(label);  
 form.add(title, constraints);  
 constraints.setRow(1);  
  
 JBPanel description = new JBPanel(new BorderLayout());  
 label = new JBLabel("Description: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 description.add(label);  
 JBTextArea getDescription = new JBTextArea(project.getDescription(), 6, 50);  
 getDescription.setLineWrap(true);  
 getDescription.setEditable(false);  
 getDescription.setWrapStyleWord(true);  
 getDescription.setFont(label.getFont());  
 description.add(new JBScrollPane(getDescription), BorderLayout.*EAST*);  
 form.add(description, constraints);  
 constraints.setRow(2);  
  
 JBPanel getState = new JBPanel(new BorderLayout());  
 label = new JBLabel("State: " + States.*toString*(project.getState()),  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 getState.add(label);  
 form.add(getState, constraints);  
 createProject.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 createProject.setContentPane(form);  
 createProject.pack();  
 createProject.setVisible(true);  
 }  
  
 private static synchronized void showMilestoneData(Milestone milestone) {  
 JFrame createProject = new JFrame(milestone.getTitle());  
 GridConstraints constraints = new GridConstraints();  
 JBPanel form = new JBPanel();  
 form.setLayout(new GridLayoutManager(4, 1,  
 JBUI.*insets*(12, 20), 20, 20));  
  
 JBPanel title = new JBPanel(new BorderLayout());  
 JBLabel label = new JBLabel("Title: " + milestone.getTitle(),  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 title.add(label);  
 form.add(title, constraints);  
 constraints.setRow(1);  
  
 JBPanel description = new JBPanel(new BorderLayout());  
 label = new JBLabel("Description: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 description.add(label);  
 JBTextArea getDescription = new JBTextArea(milestone.getDescription(), 6, 50);  
 getDescription.setLineWrap(true);  
 getDescription.setEditable(false);  
 getDescription.setWrapStyleWord(true);  
 getDescription.setFont(label.getFont());  
 description.add(new JBScrollPane(getDescription), BorderLayout.*EAST*);  
 form.add(description, constraints);  
 constraints.setRow(2);  
  
 JBPanel deadline = new JBPanel(new BorderLayout());  
 label = new JBLabel("Deadline: " + milestone.getDeadline().toString(),  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 deadline.add(label);  
 form.add(deadline, constraints);  
 constraints.setRow(3);  
  
 JBPanel getState = new JBPanel(new BorderLayout());  
 label = new JBLabel("State: " + States.*toString*(milestone.getState()),  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 getState.add(label);  
 form.add(getState, constraints);  
 createProject.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 createProject.setContentPane(form);  
 createProject.pack();  
 createProject.setVisible(true);  
 }  
  
 private static synchronized void showTaskData(Task task) {  
 JFrame createProject = new JFrame(task.getTitle());  
 GridConstraints constraints = new GridConstraints();  
 JBPanel form = new JBPanel();  
 form.setLayout(new GridLayoutManager(5, 1,  
 JBUI.*insets*(12, 20), 20, 20));  
  
 JBPanel title = new JBPanel(new BorderLayout());  
 JBLabel label = new JBLabel("Title: " + task.getTitle(),  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 title.add(label);  
 form.add(title, constraints);  
 constraints.setRow(1);  
  
 JBPanel description = new JBPanel(new BorderLayout());  
 label = new JBLabel("Description: ",  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 description.add(label);  
 JBTextArea getDescription = new JBTextArea(task.getDescription(), 6, 50);  
 getDescription.setLineWrap(true);  
 getDescription.setEditable(false);  
 getDescription.setWrapStyleWord(true);  
 getDescription.setFont(label.getFont());  
 description.add(new JBScrollPane(getDescription), BorderLayout.*EAST*);  
 form.add(description, constraints);  
 constraints.setRow(2);  
  
 JBPanel deadline = new JBPanel(new BorderLayout());  
 label = new JBLabel("Deadline: " + task.getDeadline().toString(),  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 deadline.add(label);  
 form.add(deadline, constraints);  
 constraints.setRow(3);  
  
 JBPanel panel = new JBPanel(new BorderLayout());  
 label = new JBLabel("Label: " +  
 (task.getLabels().size() == 0 ? "None" : task.getLabels().get(0).getTitle()),  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 panel.add(label);  
 form.add(panel, constraints);  
 constraints.setRow(4);  
  
 JBPanel getState = new JBPanel(new BorderLayout());  
 label = new JBLabel("State: " + States.*toString*(task.getState()),  
 UIUtil.ComponentStyle.*REGULAR*, UIUtil.FontColor.*NORMAL*);  
 label.setHorizontalAlignment(SwingConstants.*LEFT*);  
 getState.add(label);  
 form.add(getState, constraints);  
 createProject.setDefaultCloseOperation(WindowConstants.*DISPOSE\_ON\_CLOSE*);  
 createProject.setContentPane(form);  
 createProject.pack();  
 createProject.setVisible(true);  
 }  
}

## AddLabelAction.java

package org.taimuraztibilov.taskmanager.action;  
  
import com.intellij.openapi.actionSystem.AnAction;  
import com.intellij.openapi.actionSystem.AnActionEvent;  
import com.intellij.openapi.project.Project;  
import org.jetbrains.annotations.NotNull;  
import org.taimuraztibilov.taskmanager.ui.AddDataFormBuilder;  
  
public class AddLabelAction extends AnAction {  
 @Override  
 public void update(@NotNull AnActionEvent e) {  
 Project project = e.getProject();  
 e.getPresentation().setEnabled(project != null);  
 }  
  
 @Override  
 public void actionPerformed(@NotNull AnActionEvent e) {  
 AddDataFormBuilder.*addLabelByUser*();  
 }  
}

## AddProjectAction.java

package org.taimuraztibilov.taskmanager.action;  
  
import com.intellij.openapi.actionSystem.AnAction;  
import com.intellij.openapi.actionSystem.AnActionEvent;  
import com.intellij.openapi.project.Project;  
import org.jetbrains.annotations.NotNull;  
import org.taimuraztibilov.taskmanager.ui.AddDataFormBuilder;  
  
public class AddProjectAction extends AnAction {  
 @Override  
 public void update(@NotNull AnActionEvent e) {  
 Project project = e.getProject();  
 e.getPresentation().setEnabledAndVisible(project != null);  
 }  
  
 @Override  
 public void actionPerformed(@NotNull AnActionEvent e) {  
 AddDataFormBuilder.*addProjectByUser*(e.getProject());  
 }  
}

## AddMilestoneAction.java

package org.taimuraztibilov.taskmanager.action;  
  
import com.intellij.openapi.actionSystem.AnAction;  
import com.intellij.openapi.actionSystem.AnActionEvent;  
import com.intellij.openapi.project.Project;  
import org.jetbrains.annotations.NotNull;  
import org.taimuraztibilov.taskmanager.base.PluginManagerService;  
import org.taimuraztibilov.taskmanager.ui.AddDataFormBuilder;  
  
public class AddMilestoneAction extends AnAction {  
 @Override  
 public void update(@NotNull AnActionEvent e) {  
 Project project = e.getProject();  
 e.getPresentation().setVisible(project != null);  
 e.getPresentation().setEnabled(PluginManagerService.*getInstance*().getTrackingProject() != -1);  
 }  
  
 @Override  
 public void actionPerformed(@NotNull AnActionEvent e) {  
 AddDataFormBuilder.*addMilestoneByUser*(PluginManagerService.*getInstance*().getTrackingProject());  
 }  
}

## AddTaskAction

package org.taimuraztibilov.taskmanager.action;  
  
import com.intellij.openapi.actionSystem.AnAction;  
import com.intellij.openapi.actionSystem.AnActionEvent;  
import com.intellij.openapi.project.Project;  
import org.jetbrains.annotations.NotNull;  
import org.taimuraztibilov.taskmanager.base.PluginManagerService;  
import org.taimuraztibilov.taskmanager.ui.AddDataFormBuilder;  
  
public class AddTaskAction extends AnAction {  
 @Override  
 public void update(@NotNull AnActionEvent e) {  
 Project project = e.getProject();  
 e.getPresentation().setVisible(project != null);  
 e.getPresentation().setEnabled(PluginManagerService.*getInstance*().getTrackingMilestone() != -1);  
 }  
  
 @Override  
 public void actionPerformed(@NotNull AnActionEvent e) {  
 AddDataFormBuilder.*addTaskByUser*(PluginManagerService.*getInstance*().getTrackingMilestone());  
 }  
}

## CreateReportAction.java

package org.taimuraztibilov.taskmanager.action;  
  
import com.intellij.openapi.actionSystem.AnAction;  
import com.intellij.openapi.actionSystem.AnActionEvent;  
import com.intellij.openapi.project.Project;  
import org.jetbrains.annotations.NotNull;  
import org.taimuraztibilov.taskmanager.base.PluginManagerService;  
import org.taimuraztibilov.taskmanager.ui.AddDataFormBuilder;  
  
public class CreateReportAction extends AnAction {  
 @Override  
 public void update(@NotNull AnActionEvent e) {  
 Project project = e.getProject();  
 e.getPresentation().setVisible(project != null);  
 e.getPresentation().setEnabled(PluginManagerService.*getInstance*().getTrackingProject() != -1);  
 }  
  
 @Override  
 public void actionPerformed(@NotNull AnActionEvent e) {  
 if (e.getProject() != null)  
 AddDataFormBuilder.*generateReport*(e.getProject().getBasePath());  
 }  
}

## StopTrackAction.java

package org.taimuraztibilov.taskmanager.action;  
  
import com.intellij.openapi.actionSystem.AnAction;  
import com.intellij.openapi.actionSystem.AnActionEvent;  
import com.intellij.openapi.project.Project;  
import org.jetbrains.annotations.NotNull;  
import org.taimuraztibilov.taskmanager.base.TimeManager;  
import org.taimuraztibilov.taskmanager.ui.AddDataFormBuilder;  
  
import java.sql.SQLException;  
import java.time.LocalTime;  
  
public class StopTrackAction extends AnAction {  
 @Override  
 public void update(@NotNull AnActionEvent e) {  
 Project project = e.getProject();  
 e.getPresentation().setVisible(project != null);  
 e.getPresentation().setEnabled(TimeManager.*getInstance*().getTaskId() != -1);  
 }  
  
 @Override  
 public void actionPerformed(@NotNull AnActionEvent e) {  
 try {  
 int trackedId = TimeManager.*getInstance*().getTaskId();  
 LocalTime time = TimeManager.*getInstance*().stopTracking();  
 AddDataFormBuilder.*addKeyPointByUser*(time, trackedId);  
 } catch (SQLException throwables) {  
 *// TODO: 10.05.2020 notification* }  
 }  
}

## TrackProjectAction.java

package org.taimuraztibilov.taskmanager.action;  
  
import com.intellij.openapi.actionSystem.AnAction;  
import com.intellij.openapi.actionSystem.AnActionEvent;  
import com.intellij.openapi.project.Project;  
import org.jetbrains.annotations.NotNull;  
import org.taimuraztibilov.taskmanager.ui.ShowDataFormBuilder;  
  
public class TrackProjectAction extends AnAction {  
 @Override  
 public void update(@NotNull AnActionEvent e) {  
 Project project = e.getProject();  
 e.getPresentation().setEnabledAndVisible(project != null);  
 }  
  
 @Override  
 public void actionPerformed(@NotNull AnActionEvent e) {  
 ShowDataFormBuilder.*showProjects*();  
 }  
}

## TrackMilestoneAction

package org.taimuraztibilov.taskmanager.action;  
  
import com.intellij.openapi.actionSystem.AnAction;  
import com.intellij.openapi.actionSystem.AnActionEvent;  
import com.intellij.openapi.project.Project;  
import org.jetbrains.annotations.NotNull;  
import org.taimuraztibilov.taskmanager.base.PluginManagerService;  
import org.taimuraztibilov.taskmanager.ui.ShowDataFormBuilder;  
  
public class TrackMilestoneAction extends AnAction {  
 @Override  
 public void update(@NotNull AnActionEvent e) {  
 Project project = e.getProject();  
 e.getPresentation().setVisible(project != null);  
 e.getPresentation().setEnabled(PluginManagerService.*getInstance*().getTrackingProject() != -1);  
 }  
  
 @Override  
 public void actionPerformed(@NotNull AnActionEvent e) {  
 ShowDataFormBuilder.*showMilestones*();  
 }  
}

## TrackTaskAction.java

package org.taimuraztibilov.taskmanager.action;  
  
import com.intellij.openapi.actionSystem.AnAction;  
import com.intellij.openapi.actionSystem.AnActionEvent;  
import com.intellij.openapi.project.Project;  
import org.jetbrains.annotations.NotNull;  
import org.taimuraztibilov.taskmanager.base.PluginManagerService;  
import org.taimuraztibilov.taskmanager.ui.ShowDataFormBuilder;  
  
public class TrackTaskAction extends AnAction {  
 @Override  
 public void update(@NotNull AnActionEvent e) {  
 Project project = e.getProject();  
 e.getPresentation().setVisible(project != null);  
 e.getPresentation().setEnabled(PluginManagerService.*getInstance*().getTrackingMilestone() != -1);  
 }  
  
 @Override  
 public void actionPerformed(@NotNull AnActionEvent e) {  
 ShowDataFormBuilder.*showTasks*();  
 }  
}

## Build.gradle

plugins {  
 id 'java'  
 id 'org.jetbrains.intellij' version '0.4.18'  
}  
  
group 'org.taimuraztibilov'  
version '1.0'  
  
repositories {  
 mavenCentral()  
}  
  
dependencies {  
 testCompile group: 'junit', name: 'junit', version: '4.12'  
 compile 'com.opencsv:opencsv:5.1'  
 compile group:'org.xerial', name:'sqlite-jdbc', version:'3.30.1'  
}  
  
*// See https://github.com/JetBrains/gradle-intellij-plugin/*intellij {  
 version '2020.1'  
}  
patchPluginXml {  
 changeNotes """  
 Add change notes here.<br>  
 <em>most HTML tags may be used</em>"""  
}

## PluginManagersTest

package org.taimuraztibilov.taskmanager.base;  
  
import org.junit.Test;  
  
import java.sql.SQLException;  
import java.time.LocalDateTime;  
import java.time.LocalTime;  
import java.util.ArrayList;  
import java.util.Objects;  
  
import static org.junit.Assert.\*;  
  
public class PluginManagersTest {  
  
 @Test  
 public void getInstance() {  
 try {  
 DataBaseManager.*getInstance*();  
 } catch (Exception e) {  
 throw new AssertionError();  
 }  
 }  
  
  
 @Test  
 public void editData() {  
 Project proj = null;  
 try {  
 proj = DataBaseManager.*getInstance*().addProject("title", "description", States.*OPEN*);  
 proj.setTitle("new title");  
 proj.setState(States.*IN\_PROGRESS*);  
 } catch (SQLException throwables) {  
 throw new AssertionError();  
 }  
 assert proj.getTitle().equals("new title");  
 assert proj.getState() == States.*IN\_PROGRESS*;  
 }  
  
 @Test  
 public void addProject() {  
 Project proj = null;  
 try {  
 proj = DataBaseManager.*getInstance*().addProject("title", "description", States.*OPEN*);  
 } catch (SQLException throwables) {  
 throw new AssertionError();  
 }  
 assert proj.getTitle().equals("title");  
 assert proj.getState() == States.*OPEN*;  
 }  
  
 @Test  
 public void deleteProject() {  
 try {  
 DataBaseManager.*getInstance*().addProject("title", "description", States.*OPEN*);  
 ArrayList<Project> projects = DataBaseManager.*getInstance*().getProjects();  
 DataBaseManager.*getInstance*().deleteProject(projects.get(0).getId());  
 if (projects.size() - 1 != DataBaseManager.*getInstance*().getProjects().size()) throw new AssertionError();  
 } catch (Exception e) {  
 throw new AssertionError();  
 }  
 }  
  
 @Test  
 public void getProject() {  
 Project proj = null;  
 try {  
 proj = DataBaseManager.*getInstance*().addProject("title", "description", States.*OPEN*);  
 assert Objects.*equals*(proj.getId(), DataBaseManager.*getInstance*().getProject(proj.getId()).getId());  
 } catch (SQLException throwables) {  
 throw new AssertionError();  
 }  
 }  
  
 @Test  
 public void addMilestone() {  
 Milestone milestone = null;  
 try {  
 milestone = DataBaseManager.*getInstance*().addMilestone(  
 DataBaseManager.*getInstance*().addProject("title", "description", States.*OPEN*).getId(),  
 "title", "description", LocalDateTime.*now*(), States.*OPEN*);  
 } catch (SQLException throwables) {  
 throw new AssertionError();  
 }  
 assert milestone.getTitle().equals("title");  
 assert milestone.getState() == States.*OPEN*;  
 }  
  
 @Test  
 public void getMilestone() {  
  
 try {  
 Project p = DataBaseManager.*getInstance*().addProject("title", "description", States.*OPEN*);  
 Milestone m = DataBaseManager.*getInstance*().addMilestone(  
 p.getId(), "title", "description", LocalDateTime.*now*(), States.*OPEN*);  
 assert m.getId() == DataBaseManager.*getInstance*().getMilestone(m.getId()).getId();  
 } catch (SQLException e) {  
 throw new AssertionError();  
 }  
 }  
  
 @Test  
 public void addLabel() {  
 try {  
 DataBaseManager.*getInstance*().addLabel("0xFFFFFF", "title");  
  
 } catch (SQLException e) {  
 throw new AssertionError();  
 }  
 }  
  
 @Test  
 public void deleteLabel() {  
  
 try {  
 int id = DataBaseManager.*getInstance*().addLabel("0xFFFFFF", "title").getId();  
 var labels = DataBaseManager.*getInstance*().getLabels();  
 DataBaseManager.*getInstance*().deleteLabel(id);  
 assert labels.size() - 1 == DataBaseManager.*getInstance*().getLabels().size();  
 } catch (SQLException e) {  
 throw new AssertionError();  
 }  
 }  
  
 @Test  
 public void getLabel() {  
  
 try {  
 var l = DataBaseManager.*getInstance*().addLabel("0xFFFFFF", "title");  
 assert l.getId() == DataBaseManager.*getInstance*().getLabel(l.getId()).getId();  
 } catch (SQLException e) {  
 throw new AssertionError();  
 }  
 }  
  
 @Test  
 public void addTask() {  
  
 try {  
 DataBaseManager.*getInstance*().addMilestone(  
 DataBaseManager.*getInstance*().getProjects().get(0).getId(),  
 "title", "description", LocalDateTime.*now*(), States.*OPEN*);  
 DataBaseManager.*getInstance*().addTask(DataBaseManager.*getInstance*().getMilestones(  
 DataBaseManager.*getInstance*().getProjects().get(0).getId()  
 ).get(0).getId(), "t", "d", LocalDateTime.*now*(),  
 LocalTime.*MIN*, 1, new ArrayList<>());  
 } catch (SQLException e) {  
 throw new AssertionError();  
 }  
 }  
  
 @Test  
 public void getTask() {  
  
 try {  
 DataBaseManager.*getInstance*().addMilestone(  
 DataBaseManager.*getInstance*().getProjects().get(0).getId(),  
 "title", "description", LocalDateTime.*now*(), States.*OPEN*);  
 var t = DataBaseManager.*getInstance*().addTask(DataBaseManager.*getInstance*().getMilestones(  
 DataBaseManager.*getInstance*().getProjects().get(0).getId()  
 ).get(0).getId(), "t", "d", LocalDateTime.*now*(),  
 LocalTime.*MIN*, 1, new ArrayList<>());  
 assert t.getId() == DataBaseManager.*getInstance*().getTask(t.getId()).getId();  
 } catch (SQLException e) {  
 throw new AssertionError();  
 }  
 }  
}

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |