

Project Management Application – dokumentacja

Spis treści

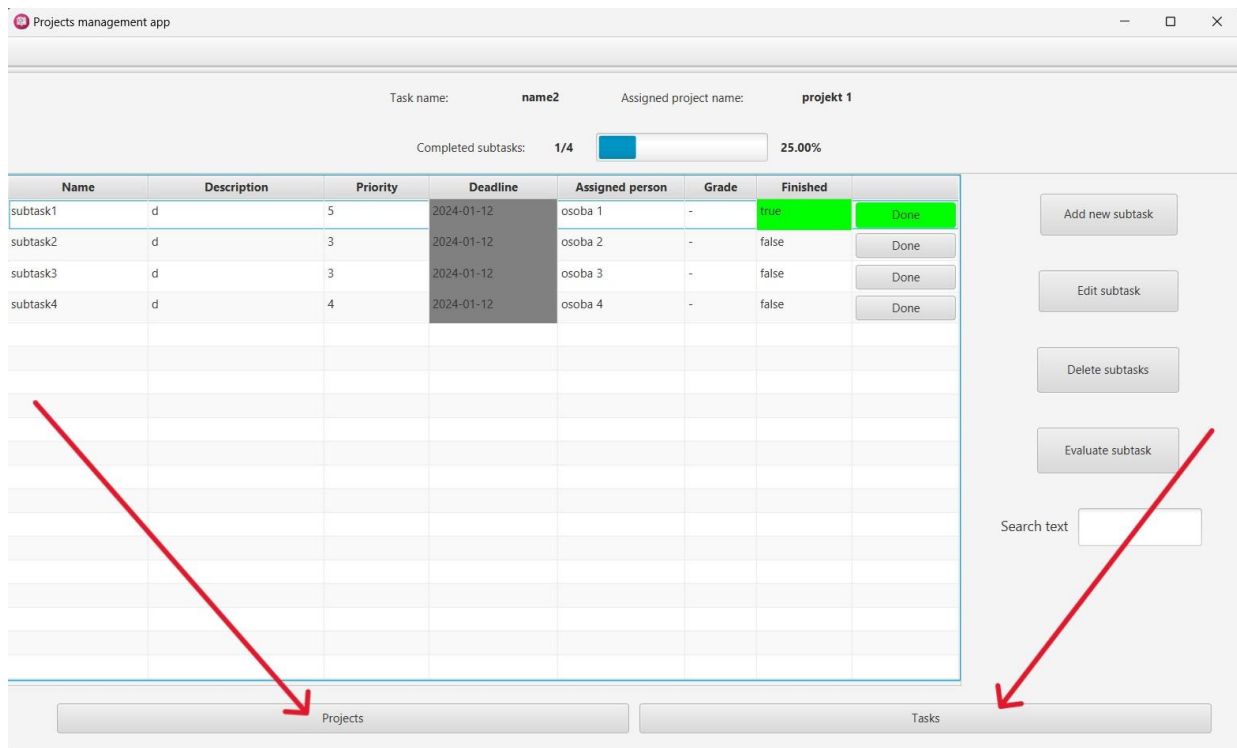
Funkcjonalności:	2
Przejdźcie do trybu zadań/projektów	2
Przejdźcie do trybu podzadań	2
Funkcjonalności wspólne dla trybów	3
Funkcjonalności w trybie zadań	8
Funkcjonalności w trybie projektów	9
Funkcjonalności w trybie podzadań	10
Warunki ukończenia projektu/zadania oraz kolor komórek kolumny „deadline”	11
Struktura projektu – diagram klas	12
Zastosowane wzorce projektowe	13
Strategia	13
Obserwator	15
Fasada i Singleton	17

Aplikacja została zaprojektowana z myślą o ułatwieniu efektywnego zarządzania zadaniami i projektami. Oferuje trzy główne tryby: tryb zadań, tryb projektów oraz tryb podzadań.

Funkcjonalności:

Przejsięcie do trybu zadań/projektów

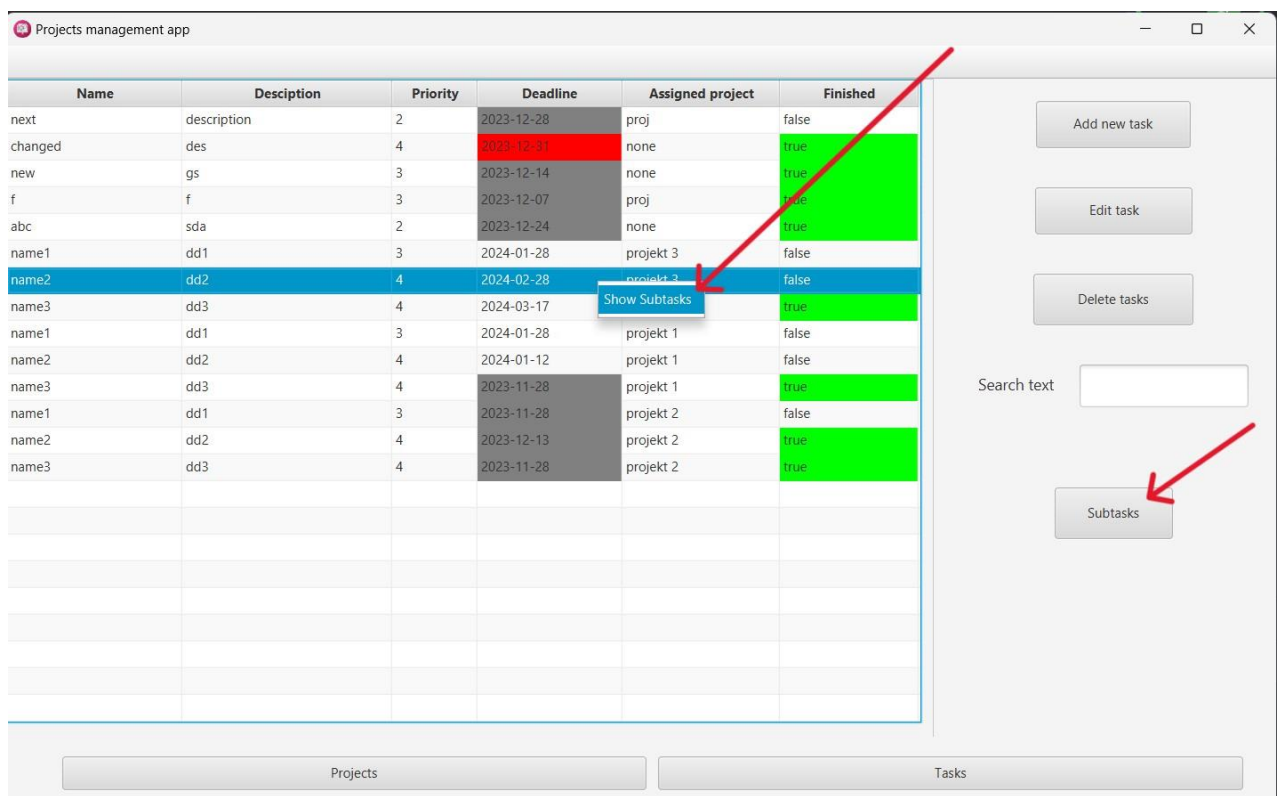
W każdym z trzech trybów w dolnym menu aplikacji mamy przyciski, które przekierowują nas do trybu zadań/projektów.



Przejsięcie do trybu podzadań

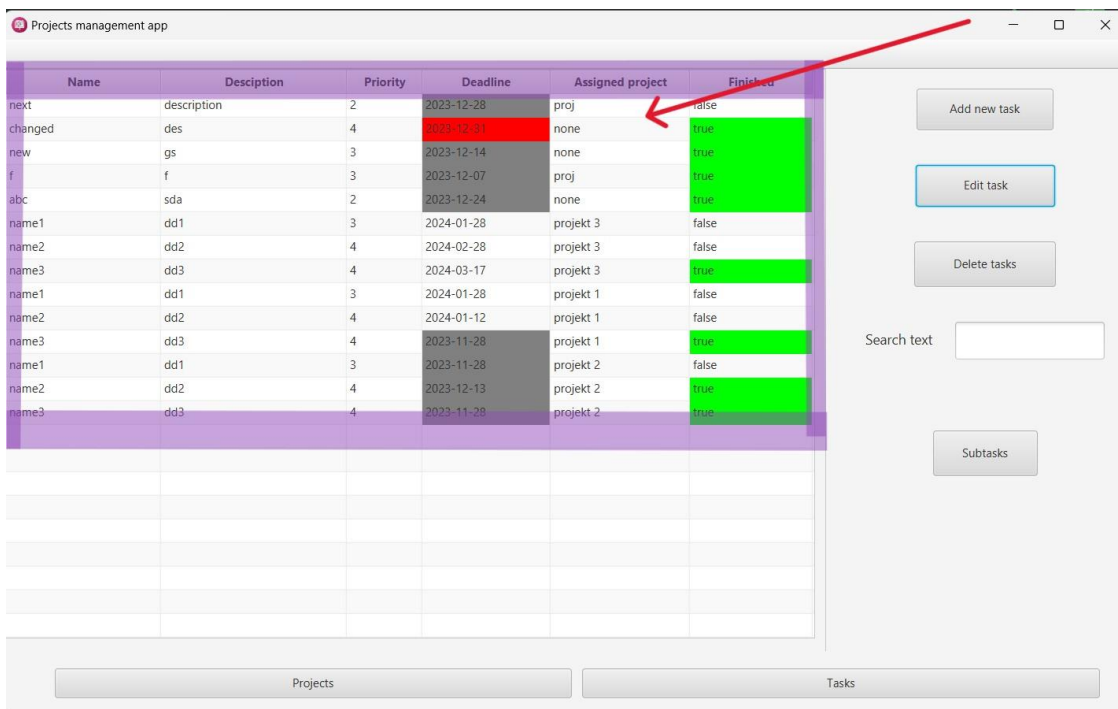
Przejsięcie do trybu podzadań jest możliwe tylko z trybu zadań po:

- 1) wybraniu zadania (klikając lewym przyciskiem myszy na pozycję w tabeli)
- 2) kliknięciu przycisku „Subtask” z bocznego menu albo po kliknięciu lewym przyciskiem na tabelę i wybraniu opcji „Show subtasks” z ukazanego menu tabeli



- ✓ Widok tabeli z obecnymi projektami/zadaniami/podzadaniami

Przejrzysta tabela prezentująca wszystkie bieżące projekty/zadania/podzadania z istotnymi informacjami.



Name	Description	Priority	Deadline	Assigned project	Finished
next	description	2	2023-12-28	proj	false
changed	des	4	2023-12-31	none	true
new	gs	3	2023-12-14	none	true
f	f	3	2023-12-07	proj	true
abc	sda	2	2023-12-24	none	true
name1	dd1	3	2024-01-28	projekt 3	false
name2	dd2	4	2024-02-28	projekt 3	false
name3	dd3	4	2024-03-17	projekt 3	true
name1	dd1	3	2024-01-28	projekt 1	false
name2	dd2	4	2024-01-12	projekt 1	false
name3	dd3	4	2023-11-28	projekt 1	true
name1	dd1	3	2023-11-28	projekt 2	false
name2	dd2	4	2023-12-13	projekt 2	true
name3	dd3	4	2023-11-28	projekt 2	true

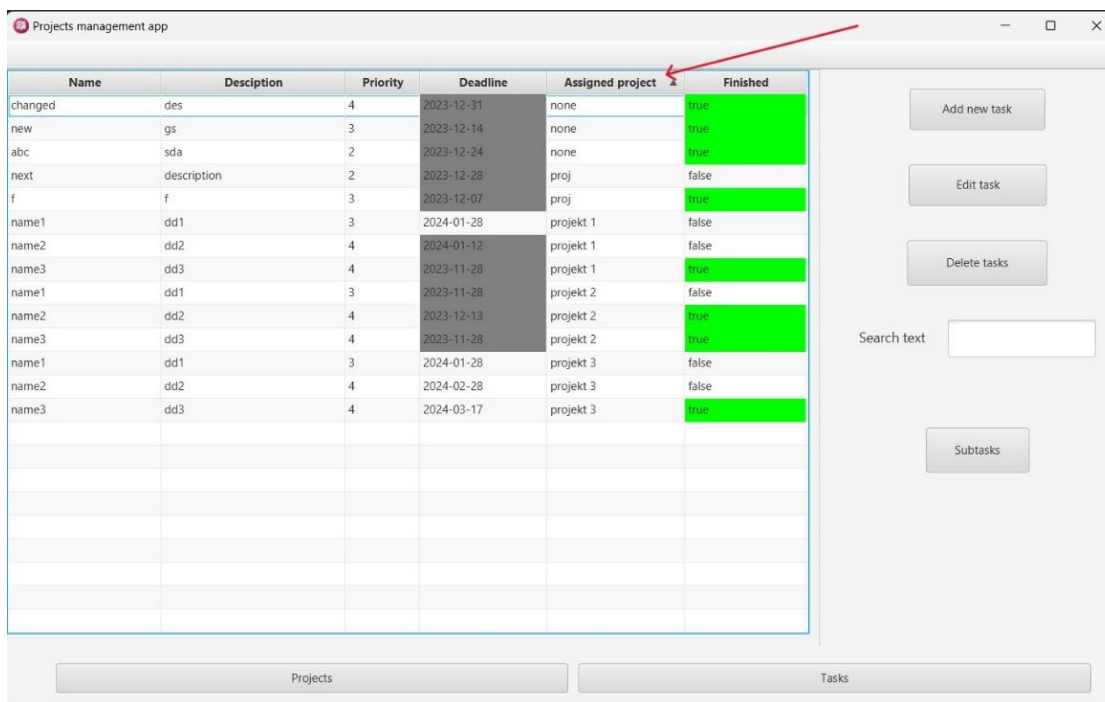
Buttons: Add new task, Edit task, Delete tasks, Search text, Subtasks.

Footer: Projects, Tasks

- ✓ Sortowanie tabeli po kolumnach

Możliwość sortowania elementów według różnych kryteriów dla lepszej organizacji. Mamy możliwość sortowania według kolumn z tabeli. Chcąc posortować dane, klikamy lewym przyciskiem na nazwę kolumny - wtedy pojawi się odpowiednia strzałka, mówiąca o kolejności sortowania. Aby zmienić kolejność albo wyłączyć sortowania, klikamy jeszcze raz na nazwę kolumny.

Chcąc posortować dane według kilku kolumn po posortowaniu ich według jednej kolumny, musimy przytrzymać klawisz „Shift” i, trzymając go, nacisnąć na odpowiednią nazwę kolumny.



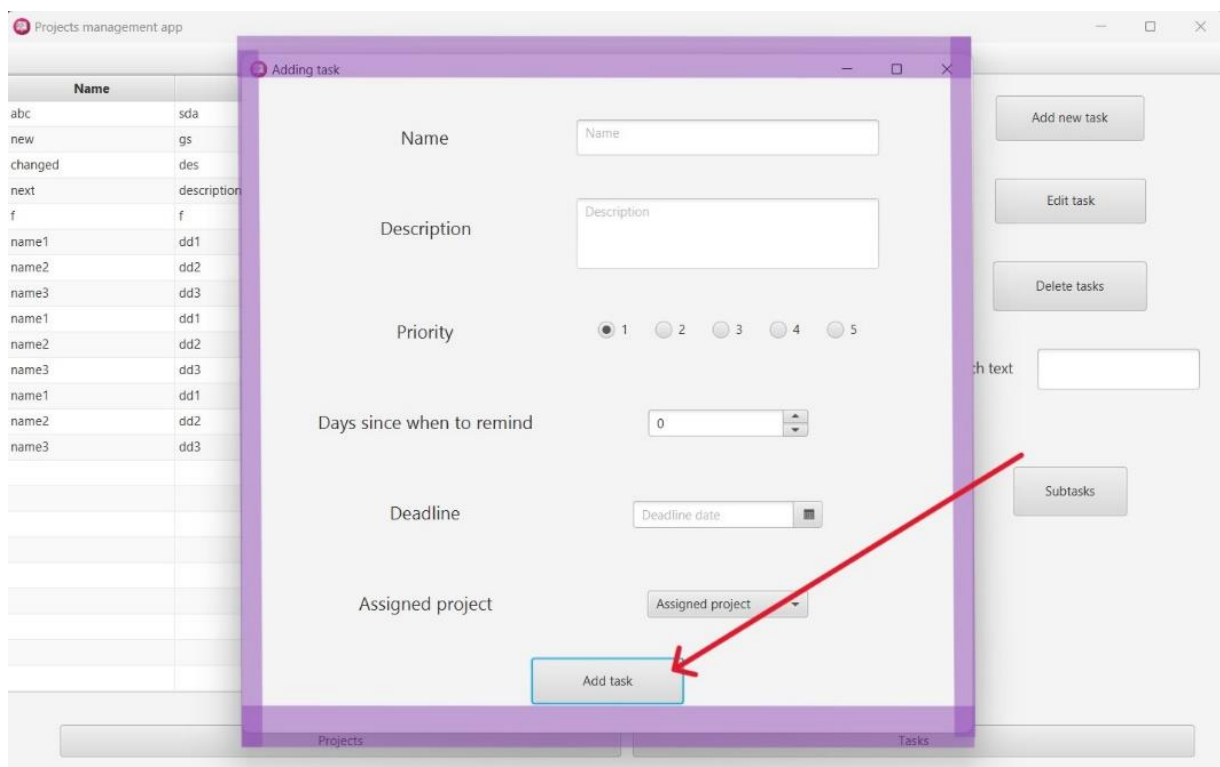
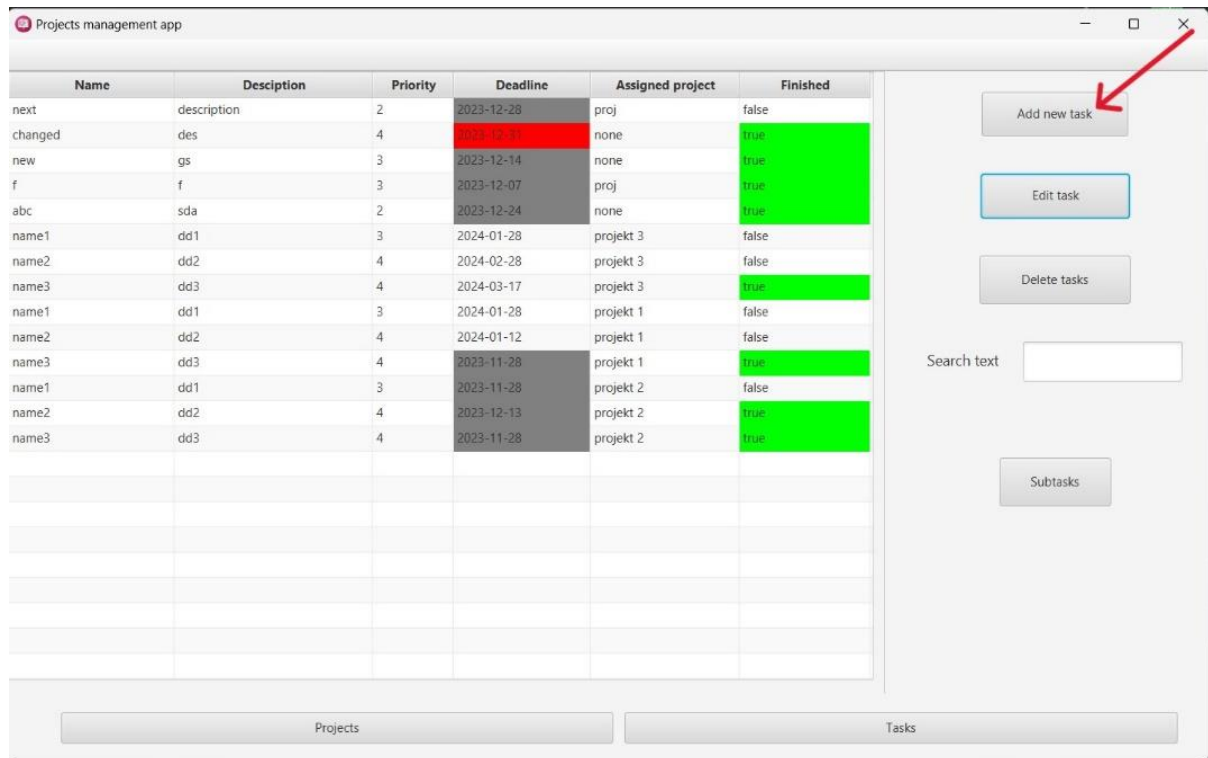
Name	Description	Priority	Deadline	Assigned project	Finished
changed	des	4	2023-12-31	none	true
new	gs	3	2023-12-14	none	true
abc	sda	2	2023-12-24	none	true
next	description	2	2023-12-28	proj	false
f	f	3	2023-12-07	proj	true
name1	dd1	3	2024-01-28	projekt 1	false
name2	dd2	4	2024-01-12	projekt 1	false
name3	dd3	4	2023-11-28	projekt 1	true
name1	dd1	3	2023-11-28	projekt 2	false
name2	dd2	4	2023-12-13	projekt 2	true
name3	dd3	4	2023-11-28	projekt 2	true
name1	dd1	3	2024-01-28	projekt 3	false
name2	dd2	4	2024-02-28	projekt 3	false
name3	dd3	4	2024-03-17	projekt 3	true

Buttons: Add new task, Edit task, Delete tasks, Search text, Subtasks.

Footer: Projects, Tasks

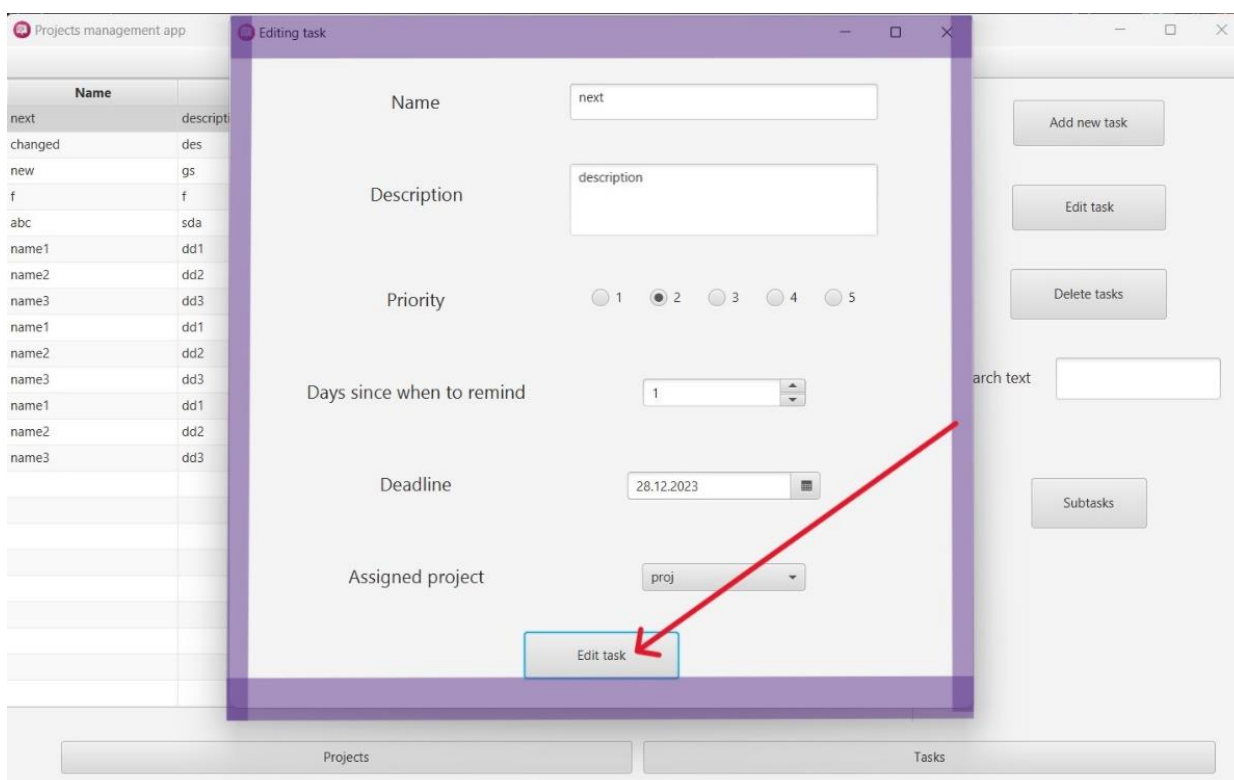
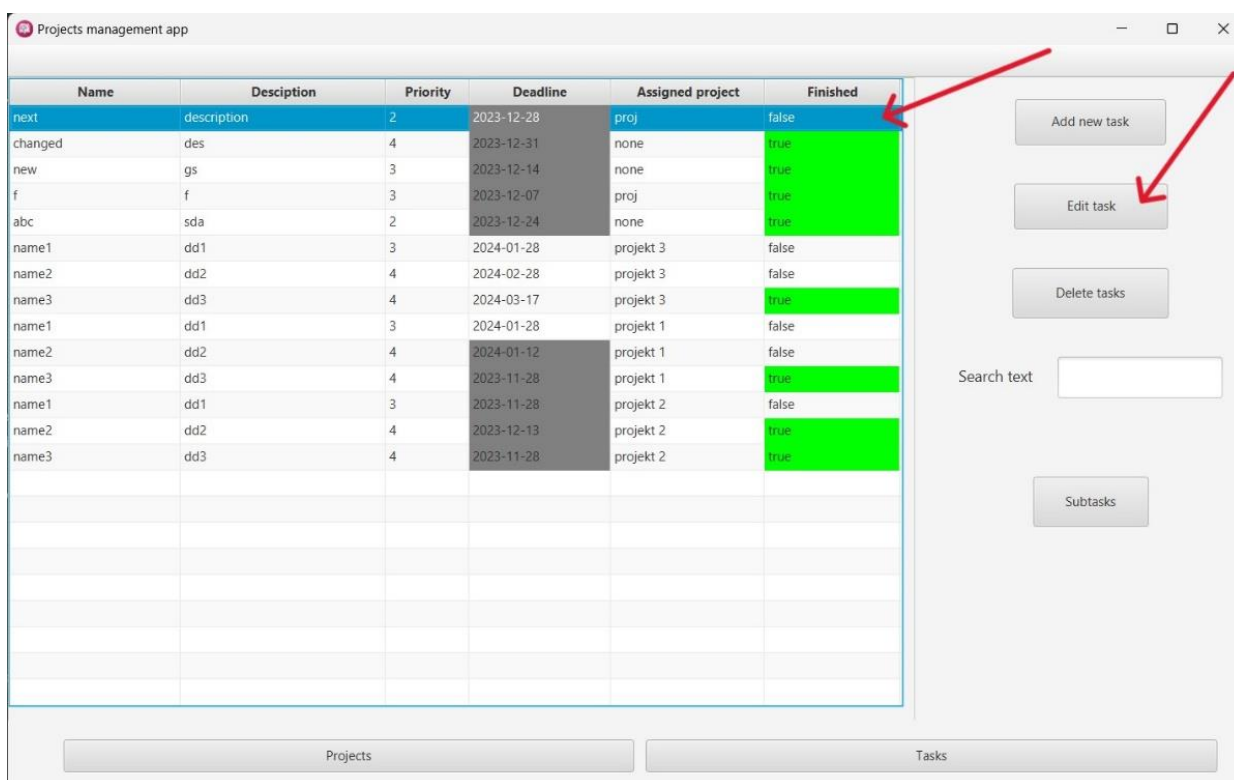
✓ Dodanie projektu/zadania/podzadania

Możliwość dodania elementu, klikając przycisk „Add new project”/„Add new task”/„Add new subtask” z bocznego menu. Otworzy nam się intuicyjny interfejs, do którego wprowadzamy parametry. Potem klikamy przycisk „Add project”/„Add task”/„Add subtask”, znajdujący się na samym dole. Zostaniemy poinformowani komunikatem o poprawnym dodaniu nowego projektu/zadania/podzadania, bądź o jakiś błędach.



✓ Edycja projektu/zadania/podzadania

Możliwość dokonywania zmian w istniejących elementach w zależności od potrzeb. Aby dokonać edycji, najpierw musimy wybrać element z tabeli, klikając lewym przyciskiem myszy na wybraną pozycję, a następnie kliknąć przycisk „Edit project”/„Edit task”/„Edit subtask” z bocznego menu. Otworzy nam się intuicyjny interfejs, w którym będą widnieć obecne parametry zadania. Po edycji dowolnych parametrów, aby zatwierdzić zmiany klikamy przycisk „Edit project”/„Edit task”/„Edit subtask” znajdujący się na samym dole. Zostaniemy poinformowani komunikatem o poprawnej edycji, bądź o jakiś błędach. Jeśli nie chcemy wprowadzać żadnych zmian, wychodzimy z trybu zamykając okienko.

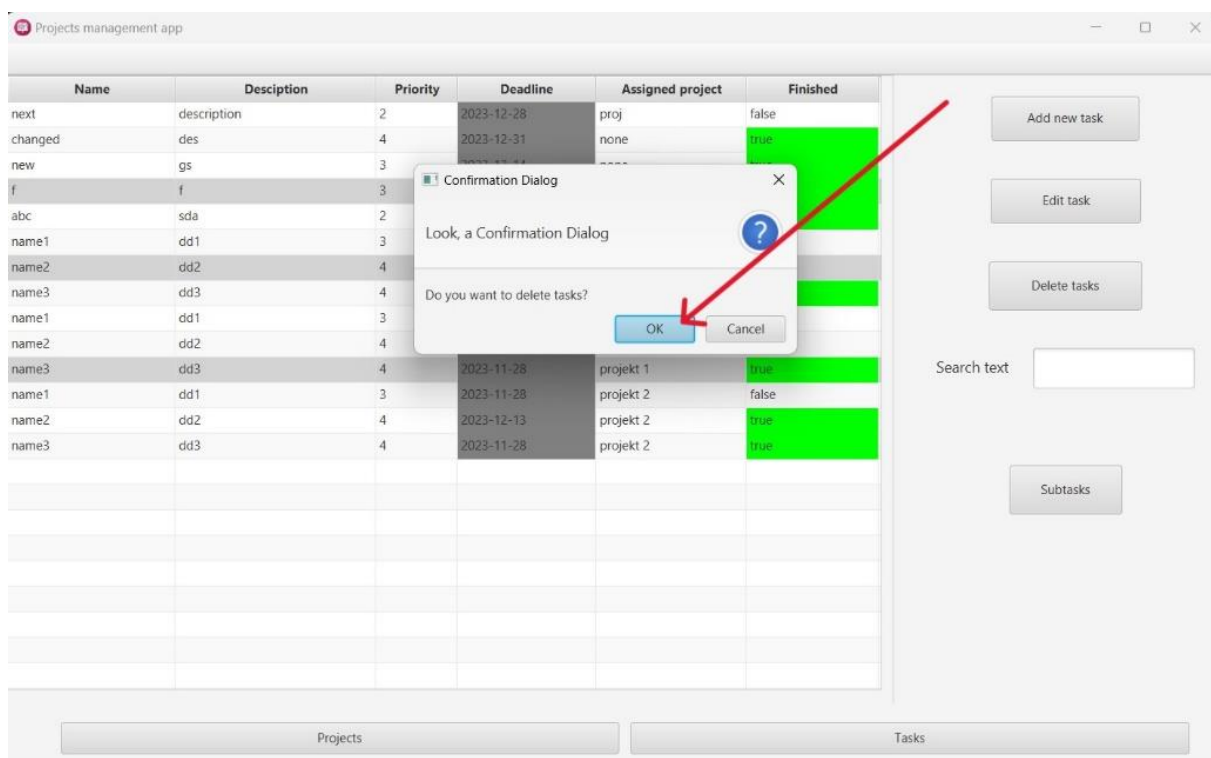
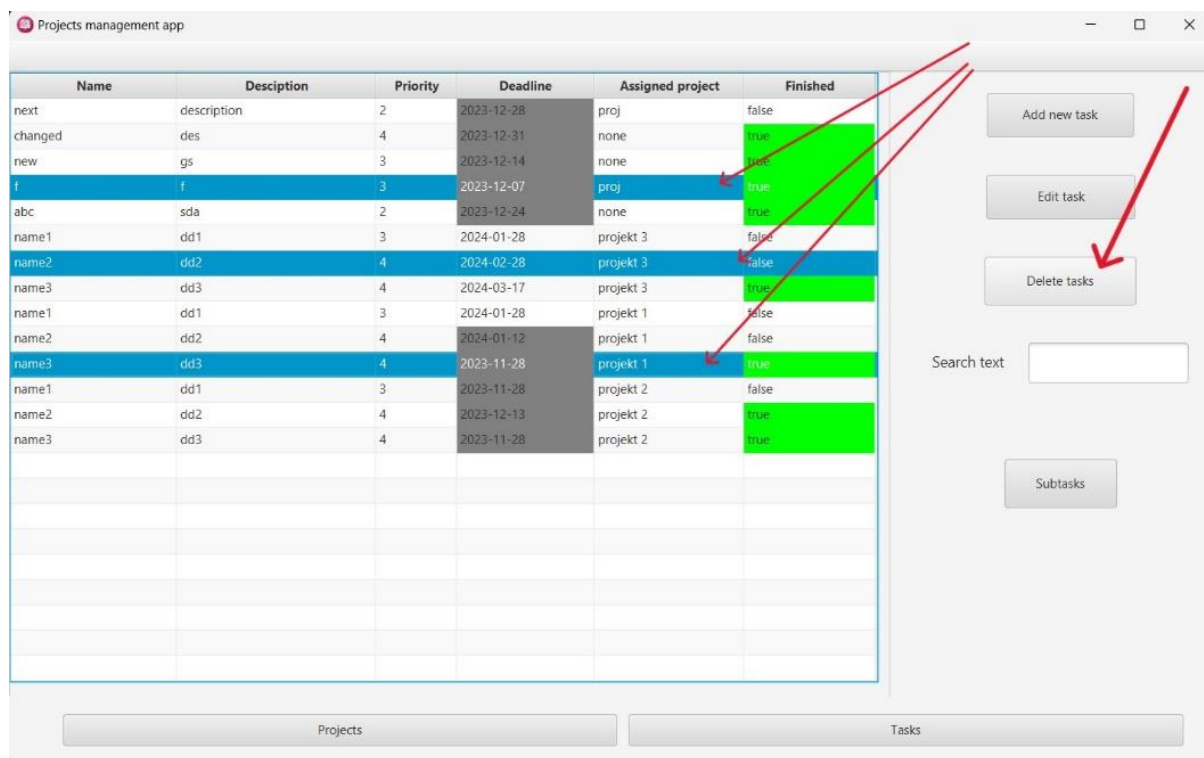


✓ Usunięcie projektu/zadania/podzadania

Możliwość usunięcia elementu. Aby tego dokonać, należy wybrać element bądź kilka elementów z tabeli, klikając lewym przyciskiem myszy na wybraną pozycję (by wybrać kilka, należy przed kliknięciem kolejnej pozycji przytrzymać przycisk Ctrl), a następnie kliknąć przycisk „Delete project”/„Delete task”/„Delete subtask” z bocznego menu oraz zaakceptować decyzję w ukazanym oknie. Zostaniemy poinformowani komunikatem o poprawnym usunięciu, bądź o jakiś błędach.

Usunięcie projektu: Usuwa wszystkie zadania przypisane do tego projektu.

Usunięcie zadania: Usuwa wszystkie podzadania przypisane do tego zadania.



✓ Dynamiczne filtrowanie

Elastyczne filtrowanie elementów dla szybkiego dostępu do istotnych informacji.

Wpisując dowolny tekst w pole tekstowe, zostaną pokazane jedynie elementy, które w swoich parametrach posiadają wpisany tekst.

The screenshot shows the 'Projects management app' interface. It features a table with columns: Name, Description, Priority, Deadline, Assigned project, and Finished. The table contains 15 rows of data. To the right of the table is a sidebar with buttons: 'Add new task', 'Edit task', 'Delete tasks', 'Search text', and 'Subtasks'. A red arrow points to the 'Search text' input field.

Name	Description	Priority	Deadline	Assigned project	Finished
next	description	2	2023-12-28	proj	false
changed	des	4	2023-12-31	none	true
new	gs	3	2023-12-14	none	true
f	f	3	2023-12-07	proj	true
abc	sda	2	2023-12-24	none	true
name1	dd1	3	2024-01-28	projekt 3	false
name2	dd2	4	2024-02-28	projekt 3	false
name3	dd3	4	2024-03-17	projekt 3	true
name1	dd1	3	2024-01-28	projekt 1	false
name2	dd2	4	2024-01-12	projekt 1	false
name3	dd3	4	2023-11-28	projekt 1	true
name1	dd1	3	2023-11-28	projekt 2	false
name2	dd2	4	2023-12-13	projekt 2	true
name3	dd3	4	2023-11-28	projekt 2	true

The screenshot shows the 'Projects management app' interface with the search bar containing the text 'true'. The table is filtered to show only rows where the 'Finished' status is 'true'.

Name	Description	Priority	Deadline	Assigned project	Finished
changed	des	4	2023-12-31	none	true
new	gs	3	2023-12-14	none	true
f	f	3	2023-12-07	proj	true
abc	sda	2	2023-12-24	none	true
name3	dd3	4	2024-03-17	projekt 3	true
name3	dd3	4	2023-11-28	projekt 1	true
name2	dd2	4	2023-12-13	projekt 2	true
name3	dd3	4	2023-11-28	projekt 2	true

Funkcjonalności w trybie zadań

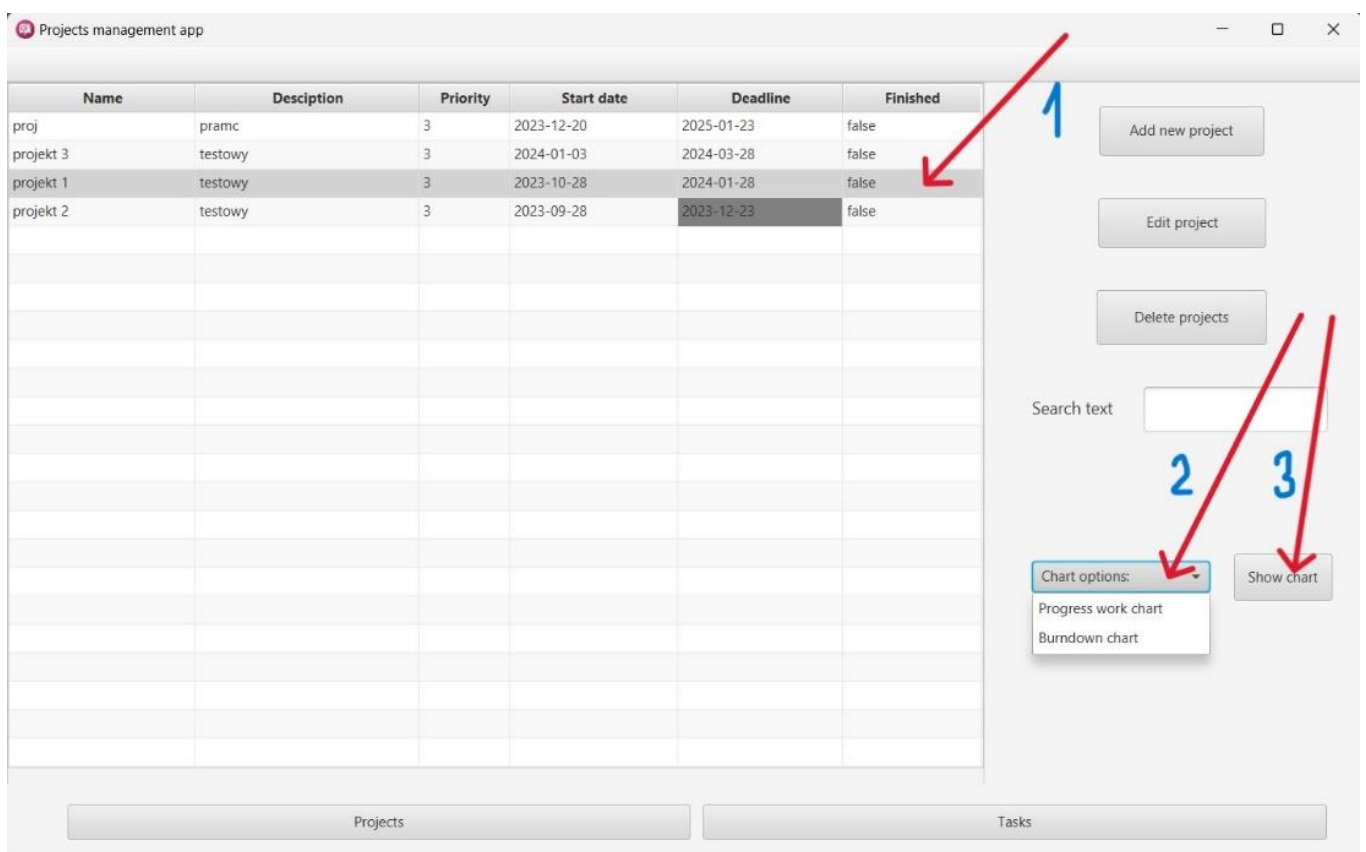
✓ Przejście do trybu podzadań

- ✓ Pokazanie wykresu

Mamy do wyboru dwa rodzaje wykresów:

- 1) Burndown chart – prezentuje wykres wypalania burndown. Pokazuje postęp ukończenia projektu w kontekście czasu. Oś X – to czas; Oś Y - Liczba Zadań; Linia Celu (Right line): reprezentuje oczekiwany przebieg postępu projektu w ciągu czasu. Umożliwia porównanie planu do rzeczywistego postępu; Linia Rzeczywistego Postępu: odzwierciedla faktyczną liczbę zadań, które zostały zrealizowane w danym punkcie czasowym.
- 2) Progress work chart – pokazuje procent ukończenia poszczególnych zadań.

- 1) Wybrać projekt, klikając lewym przyciskiem myszy na wybraną pozycję z tabeli
- 2) Wybrać rodzaj wykresu z rozwijanej listy wyboru w bocznym menu
- 3) Kliknąć przycisk „Show chart” z bocznego menu

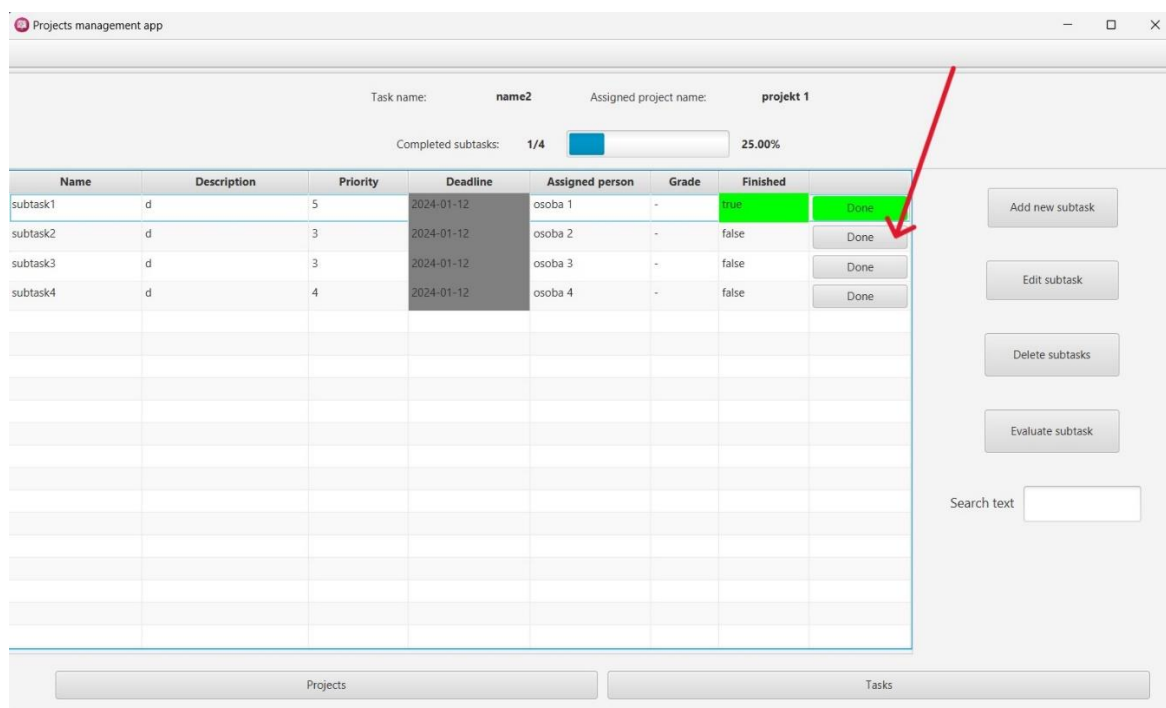


Funkcjonalności w trybie podzadań

✓ Done/undone

Oznaczanie podzadań jako wykonane/niewykonane dla kontroli postępu.

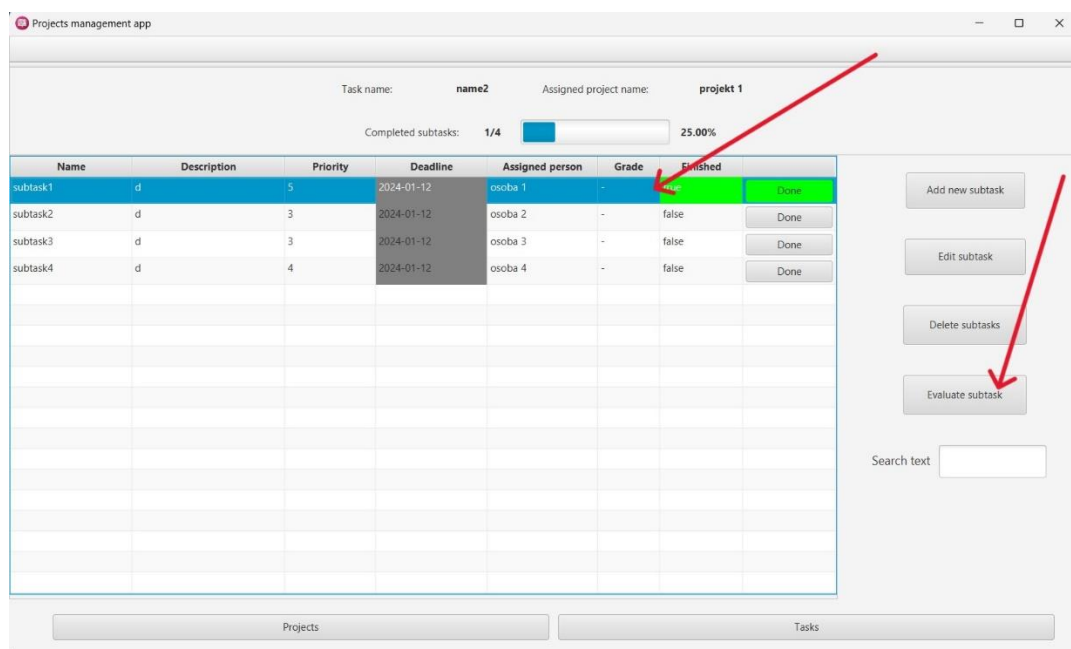
By oznaczyć podzadanie jako wykonane/niewykonane, musimy kliknąć na przycisk znajdujący się w tabeli w ostatniej kolumnie obok kolumny „finished” w wybranym wierszu – podzadaniu.

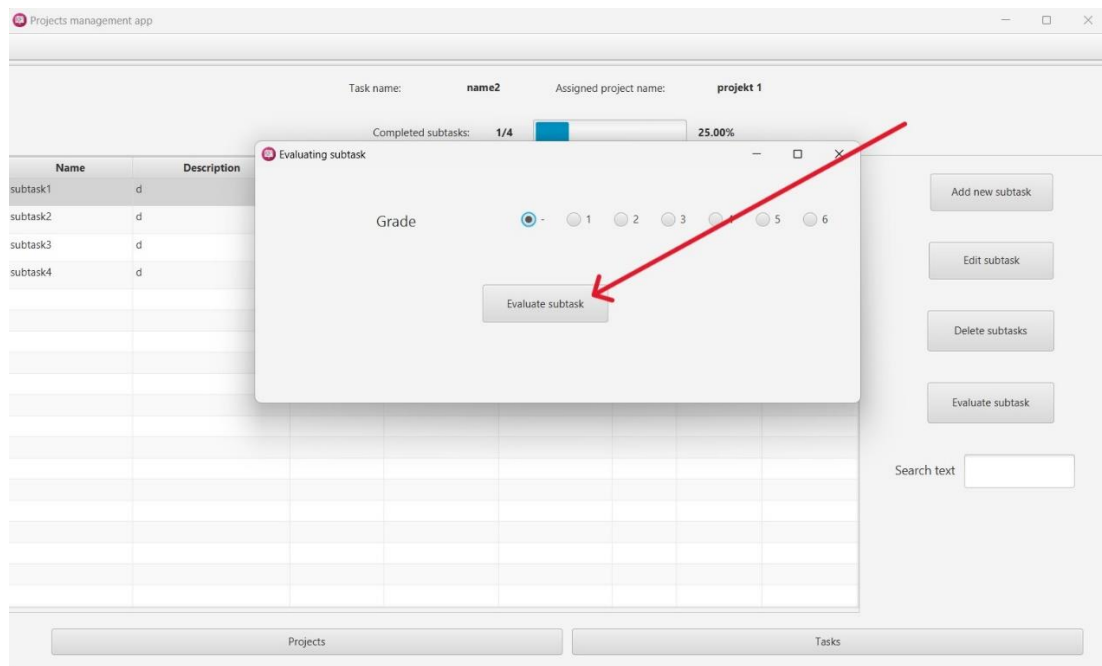


✓ Ocena ukończonego podzadania

Możliwość przypisywania ocen ukończonym podzadaniom.

Jeśli podzadanie jest ukończone, możemy je ocenić; w przeciwnym przypadku dostaniemy komunikat o błędzie. Aby tego dokonać, należy wybrać podzadanie, klikając lewym przyciskiem myszy na pozycję w tabeli, a następnie kliknąć przycisk „Evaluate subtask” z bocznego menu. Otworzy nam się okno z możliwością wyboru oceny (1-6, „-” oznacza brak oceny). Po dokonaniu wyboru, należy kliknąć przycisk „Evaluate subtask” znajdujący się w wyświetlonym oknie z wyborem ocen. Zostaniemy poinformowani komunikatem o poprawnej ocenie podzadania, bądź o jakiś błędach.





Warunki ukończenia projektu/zadania oraz kolor komórek kolumny „deadline”

- Projekt jest wykonany, gdy wszystkie jego zadania są wykonane; zadanie jest wykonane, gdy wszystkie jego podzadania są wykonane.
- Czerwony kolor komórki w tabeli w kolumnie „deadline” oznacza, że czas wykonania zbliża się; szary kolor komórki w tabeli w kolumnie „deadline” oznacza, że czas wykonania minął.

Struktura projektu – diagram klas



Zastosowane wzorce projektowe

Strategia

Wzorzec strategii został zastosowany w celu umożliwienia dynamicznego wyboru i zamiany algorytmów lub strategii w trakcie działania programu.

W projekcie wykorzystano wzorzec strategii do obsługi różnych strategii prezentacji postępu projektów (wykresów). Zaimplementowane zostały różne algorytmy generowania wykresów. Obecnie do wyboru są dwa rodzaje wykresów: burndown chart oraz progress work chart.

Interfejs ChartStrategy zawiera trzy metody: pierwsza ustawia aktualnie wybrany projekt, druga generuje odpowiedni wykres, a trzecia zwraca informacje powstałe podczas generacji wykresu.

```
2 implementations
public interface ChartStrategy {
    1 usage 2 implementations
    void setOperatedProject(Project operatedProject);
    1 usage 2 implementations
    Chart generateChart();

    1 usage 2 implementations
    ArrayList<String> getInfo();
}
```

Dwie klasy implementujące interfejs, reprezentujące rodzaj prezentacji danych.

```
public class ProgressWorkChartStrategy implements ChartStrategy {
    2 usages
    private ProgressWorkChartData progressWorkChartData;

    1 usage
    @Override
    public void setOperatedProject(Project operatedProject) {
        progressWorkChartData = new ProgressWorkChartData(operatedProject);
    }

    1 usage
    @Override
    public Chart generateChart() {

public class BurndownChartStrategy implements ChartStrategy {
    3 usages
    private BurndownChartData burndownChartData;
    1 usage
    @Override
    ~ public void setOperatedProject(Project operatedProject) {
        burndownChartData = new BurndownChartData(operatedProject);
    }

    1 usage
    @Override
    ~ public Chart generateChart() {
```

Klasa kontrolera, który obsługuje widok wykresu. W programie, przed otwarciem okna z widokiem, należy ustawić odpowiednią strategię. Wtedy, w momencie otwarcia widoku przez użytkownika, zostanie wywołana metoda `initialize()`, która przekaże do widoku odpowiedni wykres.

```
public class ChartController {
    3 usages
    private static ChartStrategy chartStrategy;

    @FXML
    protected Label projectNameField;
    @FXML
    protected Label completedTasksField;
    @FXML
    protected Label percentValueField;
    @FXML
    protected ProgressBar progressBar;
    @FXML
    protected VBox bottomVBox;
    @FXML
    protected BorderPane pane;

    1 usage
    public static void setChartStrategy(ChartStrategy chartStrategy) { ChartController.chartStrategy = chartStrategy; }

    @FXML
    public void initialize() {
        createTopPanel();
        pane.setCenter(chartStrategy.generateChart());
        createBottomPanel();
    }
}
```

Klasa kontrolera, który obsługuje widok trybu projektów. Jako składowe posiada m.in. mapę przechowującą strategię, obiekt typu `ChartStrategy` przechowujący wybraną strategię przez użytkownika.

```
public class ProjectsViewController extends MainController {
    1 usage
    public static final String PROJECT_MODIFICATION_MODE_FXML = "project-modification-mode.fxml";

    3 usages
    private final Map<String, ChartStrategy> chartStrategies = new HashMap<>();
    4 usages
    private ChartStrategy selectedChartStrategy;

    @FXML
    protected ComboBox<String> chartOptionsBox;
}
```

W metodzie inicjalizującej wypełniana jest mapa strategiami, a także rozwijana lista wyboru. Następnie zaimplementowana jest opcja polegająca na tym, że w momencie wykonanej akcji na liście wyboru ustawiana jest wybrana strategia.

```
chartStrategies.put("Progress work chart", new ProgressWorkChartStrategy());
chartStrategies.put("Burndown chart", new BurndownChartStrategy());

chartOptionsBox.setItems(FXCollections.observableArrayList("Progress work chart", "Burndown chart"));
chartOptionsBox.setOnAction(actionEvent -> {
    selectedChartStrategy = chartStrategies.get(chartOptionsBox.getValue());
});
```


Metoda, która jest wywoływana w momencie kliknięcia przycisku „Show Chart” przez użytkownika. Jeśli wszystko zostało wybrane poprawnie, najpierw ustawia ona w odpowiedniej klasie strategii wybrany z tabeli przez użytkownika projekt, a następnie w kontrolerze widoku wykresu ustawiana jest odpowiednia strategia i otwierane jest okno widoku.

```
@FXML
protected void showChart() throws IOException {
    if (isIncorrectSelectionOfTableRow(ManagerFacade.getInstance().getProjectStorage().getList())) {
        return;
    }

    if (selectedChartStrategy != null) {
        selectedChartStrategy.setOperatedProject(ManagerFacade.getInstance().getProjectStorage().getList().get(getSelectedRowsOfTable().get(0)));
        ChartController.setChartStrategy(selectedChartStrategy);
        openNewWindow( pathToViewFile: "chart.fxml", windowTitle: "Chart", isWindowCloseApp: false);
    } else {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Error Dialog");
        alert.setHeaderText("Look, an Error Dialog");
        alert.setContentText("Chart option is not selected");

        alert.showAndWait();
    }
}
```

Obserwator

Wzorzec obserwatora został zastosowany w celu umożliwienia efektywnego monitorowania i reagowania na zmiany w czasie rzeczywistym w różnych komponentach systemu. Dzięki temu projekt może dynamicznie informować inne części o swoim stanie i umożliwiać im podjęcie odpowiednich działań w zależności od zachodzących zdarzeń.

Oprócz obserwatora, który jest zastosowany do obsługi wszystkich zdarzeń (kliknięcia przycisków, kliknięcia tabeli), zastosowano go także do monitorowania aktualnego czasu (porównywanie go z deadlineem projektów/zadań/podzadań i odpowiednia reakcja), a także został zastosowany do monitorowania list z projektami, zadaniami i podzadaniami. W momencie, gdy element został dodany/edytowany/usunięty itd. z listy, tabela jest automatycznie aktualizowana.

Interfejsy Observable oraz Observer. Pierwszy z nich implementują klasy, które są obserwowane i powiadamiają, gdy zmieni się ich stan. Natomiast drugi klasy, które obserwują.

```
2 implementations
public interface Observable {
    3 usages 2 implementations
    void addObserver(Observer observer);
    no usages 2 implementations
    void removeObserver(Observer observer);
    8 usages 2 implementations
    void notifyObservers();
}
```

```
4 implementations
public interface Observer extends Serializable {
    6 usages 2 implementations
    void update();
}
```


Klasa CurrentTime, która co godzinę informuje o zmianie godziny.

```
public class CurrentTime implements Observable {
    3 usages
    private final ArrayList<Observer> observers = new ArrayList<>();
    55 usages
    > public static LocalDate getCurrentSystemTime() { return LocalDate.now(); }

    1 usage
    public void startTimer() {
        > Timer timer = new Timer();
        TimerTask timerTask = () -> { notifyObservers(); };
        timer.scheduleAtFixedRate(timerTask, delay: 0, period: 3600000); //1s - 1000ms 1min - 60 000ms 1h - 3 600 000 ms
    }

    3 usages
    @Override
    > public void addObserver(Observer observer) { observers.add(observer); }

    no usages
    @Override
    > public void removeObserver(Observer observer) { observers.remove(observer); }

    8 usages
    @Override
    > public void notifyObservers() { observers.forEach(Observer::update); }
}
```

Klasa ListStorage, która informuje obserwatorów po każdej operacji (dodaniu, ustawieniu listy itp.).

```
public class ListStorage<TaskElement> implements Serializable, Observable {
    5 usages
    private ArrayList<TaskElement> list = new ArrayList<>();
    4 usages
    private transient ArrayList<Observer> observersList = new ArrayList<>();

    > public ArrayList<TaskElement> getList() { return list; }

    ~
    public void add(TaskElement element) {
        list.add(element);
        notifyObservers();
    }

    3 usages
    ~
    public void edit(int index, TaskElement element) {
        list.set(index, element);
        notifyObservers();
    }

    4 usages
    ~
    public void remove(int rowIndex) {
        list.remove(rowIndex);
        notifyObservers();
    }

    2 usages
    ~
    public void setList(ArrayList<TaskElement> list) {
        this.list = list;
        notifyObservers();
    }

    3 usages
    > @Override
    public void addObserver(Observer observer) { observersList.add(observer); }

    no usages
    > @Override
    public void removeObserver(Observer observer) { observersList.remove(observer); }

    8 usages
    > @Override
    public void notifyObservers() { observersList.forEach(Observer::update); }
```

Metoda `update()` w klasie, która obserwuje `ListStorage`, jest wywoływana w momencie zmiany stanu klasy obserwowanej.

```
MainController.java x
6 usages 1 override
94 @Override
95 public void update() {
96     ManagerFacade.getInstance().updateStateOfElementIfCompletionStatusHasChanged(listDisplayingInTable.getList());
97     loadTable();
98     table.refresh();
99 }
```

Fasada i Singleton

Wzorzec fasady został zastosowany w celu zapewnienia jednolitego i uproszczonego interfejsu. Dzięki temu projekt może ograniczyć zależności pomiędzy komponentami poprzez udostępnienie jednego punktu dostępu do różnych funkcji systemu.

W projekcie zastosowano wzorzec fasady w klasie `ManagerFacade`; dostarcza ona proste funkcje i jednocześnie ukrywa szczegóły implementacyjne tych funkcji. Cała warstwa modelu (logiki) jest schowana za fasadą, tzn. klasa `ManagerFacade` udostępnia odpowiednie funkcje, które jeśli wywołamy, wywołują odpowiednie metody z warstwy logiki. Dzięki temu klasy kontrolerów nie muszą bezpośrednio integrować się z wieloma klasami warstwy logiki, korzystają tylko z metod udostępnianych przez klasę `ManagerFacade`.

Klasa `ManagerFacade` jest także Singletonem. Dzięki temu projekt może ograniczyć liczbę instancji tej klasy oraz zapewnić jednolity dostęp do jej funkcji w całej aplikacji. W trakcie działania aplikacji wszędzie tam, gdzie potrzebna jest komunikacja z warstwą logiki biznesowej, korzysta się z tej samej instancji klasy `ManagerFacade`.

Klasa `ManagerFacade` jako składowe zawiera obiekty, którym deleguje zadania.

```
public class ManagerFacade {
    3 usages
    private static ManagerFacade INSTANCE;

    8 usages
    private final TaskStorageManager taskManager;
    7 usages
    private final ProjectStorageManager projectManager;
    6 usages
    private final SubtaskManager subtaskManager;

    1 usage
    private ManagerFacade() {
        taskManager = new TaskStorageManager();
        projectManager = new ProjectStorageManager();
        subtaskManager = new SubtaskManager();
    }

    public static ManagerFacade getInstance() {
        if (INSTANCE == null) {
            INSTANCE = new ManagerFacade();
        }
        return INSTANCE;
    }
}
```

Publiczne metody klasy:

```
2 usages
public Project createEmptyProject() { return projectManager.createEmptyProject(); }

1 usage
public Task createTaskIfPossible(TaskInputData taskInputData) {
    return taskManager.createTaskIfPossible(taskInputData);
}

1 usage
public Project createProjectIfPossible(ProjectInputData projectInputData) {
    return projectManager.createProjectIfPossible(projectInputData);
}

1 usage
public Subtask createSubtaskIfPossible(SubtaskInputData subtaskInputData) {
    return subtaskManager.createSubtaskIfPossible(subtaskInputData);
}

10 usages
public void addTask(Task task) { taskManager.addTask(task); }

1 usage
public void editTask(int index, Task newTask) { taskManager.editTask(index, newTask); }
1 usage
public void deleteTask(int index) { taskManager.deleteTask(index); }

4 usages
public void addProject(Project project) { projectManager.addProject(project); }

1 usage
public void editProject(int index, Project newProject) {
    projectManager.editProject(index, newProject, taskManager.getTaskStorage());
}

1 usage
public void deleteProject(int index) { projectManager.deleteProject(index, taskManager.getTaskStorage()); }

1 usage
public void addSubtask(Task operatedTask, Subtask subtask) { subtaskManager.addSubtask(operatedTask, subtask); }

1 usage
public void editSubtask(Task operatedTask, int index, Subtask newSubtask) {
    subtaskManager.editSubtask(operatedTask, index, newSubtask);
}

1 usage
public void deleteSubtask(Task operatedTask, int index) { subtaskManager.deleteSubtask(operatedTask, index); }

2 usages
public void evaluateSubtask(Task operatedTask, Subtask subtask, int newGrade) {
    subtaskManager.evaluateSubtask(operatedTask, subtask, newGrade);
}

1 usage
public void updateStateOfElementIfCompletionStatusHasChanged(ArrayList<? extends TaskElement> elementsList) {
    for (TaskElement taskElement : elementsList) {
        taskElement.operationsIfCompletionStatusHasChanged();
    }
}

1 usage
public void changeElementCompletionStatus(TaskElement element) {
    element.setFinished(!element.isFinished(), CurrentTime.getCurrentSystemTime());
}
```