“TaskMaster”

Dokumentacja

Opis aplikacji

TaskMaster to mobilna aplikacja na systemy Android, umożliwiająca tworzenie i zarządzanie listą rzeczy do zrobienia.

Uruchomienie aplikacji powoduje wyświetlenie ekranu powitalnego zachęcającego do **utworzenia pierwszego zadania** za pośrednictwem naciśnięcia w fioletowy przycisk z ikoną "+" znajdujący się na środku ekranu. Wykonanie tej akcji powoduje pojawienie się nowego fragmentu zawierającego listę zadań — w tym momencie na liście pokaże się utworzone właśnie pierwsze, skonfigurowane domyślnie, zadanie. Domyślna konfiguracja zadania posiada następujące wartości atrybutów:

- **Nazwa:** "Bez tytułu*[id]"*, gdzie *[id]* to autoinkrementowana liczba całkowita, która jest przechowywana w pamięci aplikacji i zwiększa się o 1 z każdym dodanym zadaniem.
- **Priorytet**: 1.
- **Status**: niewykonane.
- **Przypomnienie:** wyłączone.
- **Ostateczny termin wykonania:** [data utworzenia zadania].
- **Załączniki:** brak.

Przycisk z ikoną "+" pozostaje widoczny po dodaniu zadania i umożliwia **tworzenie kolejnych**.

W dowolnej chwili, użytkownik może **wczytać listę z pliku tekstowego**, naciskając w przycisk menu "Wczytaj plik". Czynność ta powoduje wyświetlenie eksploratora plików, który umożliwia wybranie lokalizacji i zaznaczenie właściwego pliku. Zatwierdzenie wczytania pliku wypełnia listę zadań tymi odczytanymi z pliku, zastępując dotychczas utworzone zadania.

Utworzoną w aplikacji listę plików można także **wyeksportować do pliku tekstowego** z użyciem przycisku menu "Zapisz do pliku". Jest to możliwe tylko w wypadku, gdy lista nie jest pusta. Jeśli nie dodano jeszcze żadnego zadania, naciśnięcie w ów przycisk powoduje wyświetlenie komunikatu *Toast* informującego, że nie można wyeksportować do pliku, gdyż lista jest pusta.

Aplikacja umożliwia import i eksport zadań tylko i wyłącznie za pośrednictwem specjalnie ustrukturyzowanych i sformatowanych plików tekstowych. W takich plikach zapisane są zadania wraz z wszystkimi wartościami atrybutów.

Listę zadań można **sortować** stosując ostatni przycisk menu. Jego naciśnięcie skutkuje rozwinięciem listy opcji sortowania, których jest sześć:

- Wg. nazwy, malejąco.
- Wg. nazwy, rosnąco.
- Wg. priorytetu, malejąco.
- Wg. priorytetu, rosnąco.
- Wg. statusu, najpierw niewykonane.
- Wg. statusu, najpierw wykonane.

Usunięcie zadania umożliwia przycisk z czerwonym symbolem “X”, znajdujący się w prawej części elementu listy reprezentującego zadanie. Jego aktywacja wywołuje okno dialogowe ostrzegające o nieodwracalnych skutkach usunięcia zadania i pytające o potwierdzenie wykonania. Usunięte zadanie natychmiastowo znika z listy.

Wybrane zadanie można **oznaczyć jako wykonane lub niewykonane**, odpowiednio zaznaczając lub odznaczając przycisk typu *checkbox*, który widnieje w lewej części elementu listy (zadania). Oznaczenie zadania jako zrobione dodatkowo przekreśla znajdującą się po prawej stronie przycisku nazwę zadania, aby wyraźnie zasygnalizować, że zadanie zostało wykonane.

Ostatnią ikoną widoczną bezpośrednio z poziomu listy zadań jest **priorytet zadania**. Jest to liczba z zakresu 1-10, wyświetlona na okrągłym tle o odpowiednim kolorze, wybranym z palety barw przypisanej do tego zakresu. Paleta w postaci par *[priorytet]. [barwa]* wygląda następująco:

1. #400000FF
2. #400066FF
3. #400099FF
4. #4000FFFF
5. #40FFFF00
6. #40FFCC00
7. #40FF9900
8. #40FF6600
9. #40FF3300
10. #40FF0000

Każde zadanie można dodatkowo **edytować**. Aby otworzyć okno dialogowe edycji, wystarczy nacisnąć na element listy (zadanie). Okno dialogowe umożliwia:

- Zmianę **nazwy** z użyciem komponentu *EditText*.
- Zmianę **priorytetu** z użyciem komponentu *NumberPicker*.
- Zmianę **ostatecznego terminu wykonania** z użyciem komponentu *DatePickerDialog*.
- Ustawienie **przypomnienia** z użyciem komponentu *Switch*.
- Dodanie/usuwanie/zapisanie **załączników** z użyciem komponentów *Chip* oraz mechanizmu *ActivityResultContracts*.

Edycję można anulować przyciskiem “Anuluj” — żadne zmiany nie zostaną zapisane — lub zatwierdzić przyciskiem “Ok” — wszystkie zmiany zostaną zapisane, a widok natychmiastowo zaktualizowany.

Włączenie **przypomnienia** powoduje zbudowanie notyfikacji z wykorzystaniem mechanizmu *NotificationCompat*, które wygeneruje przypomnienie o zadaniu w dniu ustawionym jako ostateczny termin wykonania.

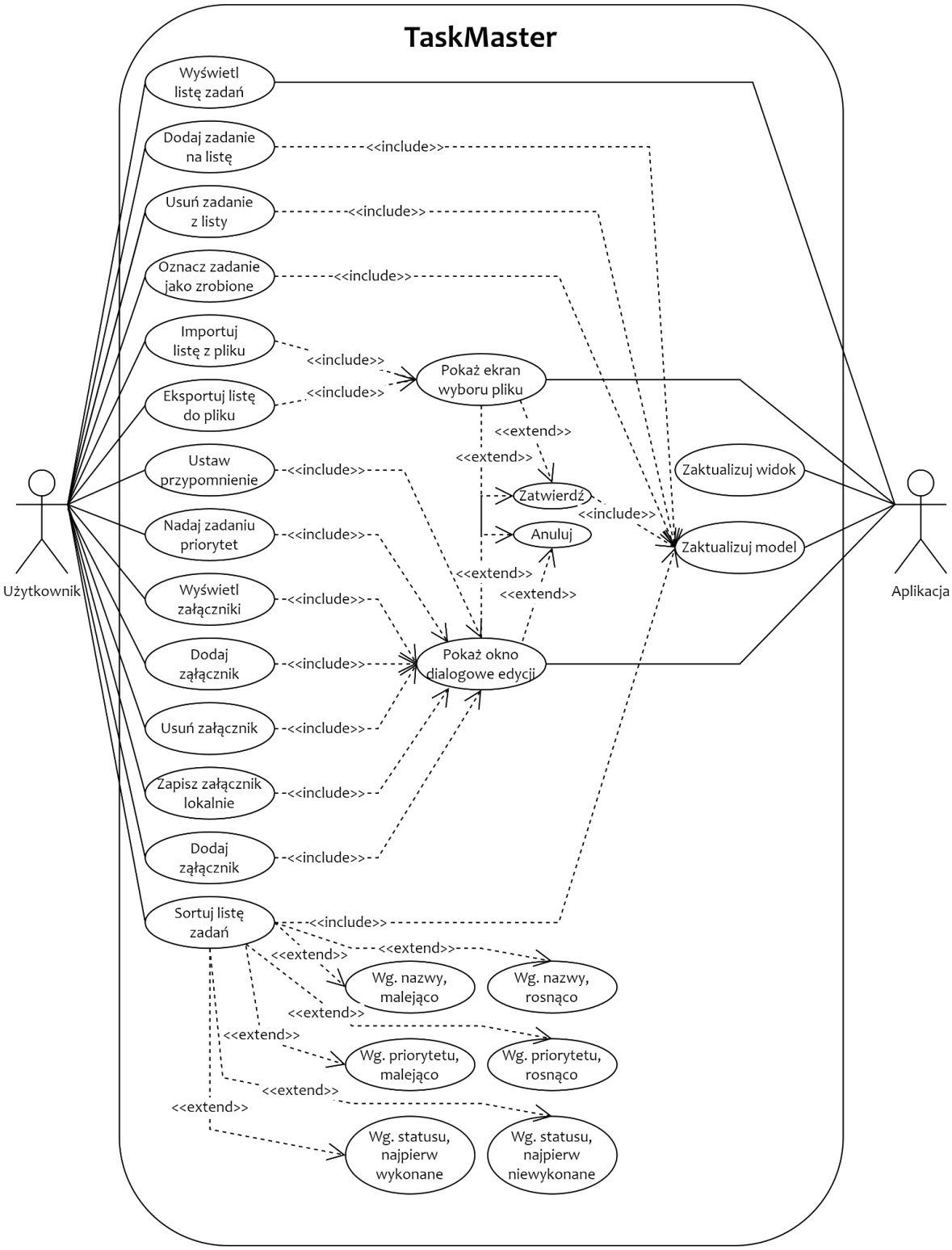
Załączniki modyfikować można tylko z poziomu dialogu edycji zadania. Aby **dodać załącznik**, należy nacisnąć szary przycisk “+”. **Usunięcie załącznika** możliwe jest poprzez naciśnięcie czarnego przycisku “X”, znajdującego się po prawej stronie elementu *Chip*, reprezentujacego załącznik. Lewą stronę elementu *Chip* zajmuje miniaturka typu pliku danego załącznika.

Centralną jego część wypełnia nazwa załącznika. Naciśnięcie w dowolną (poza przyciskiem “X”) część załącznika powoduje **zapisanie załącznika** lokalnie, w katalogu *Downloads* pod nazwą “attachment_[oryginalna nazwa pliku”. Użytkownik jest informowany o pomyślnym zapisaniu załącznika za pośrednictwem komunikatu typu *Toast*.

Lista funkcji aplikacji

- Wyświetlenie listy zadań.
- Dodanie zadania na listę.
- Edycja zadania z listy za pośrednictwem okna dialogowego edycji:
    - Zmianę nazwy z użyciem komponentu EditText.
    - Zmianę priorytetu z użyciem komponentu NumberPicker.
    - Zmianę ostatecznego terminu wykonania z użyciem komponentu DatePickerDialog.
    - Ustawienie przypomnienia z użyciem komponentu Switch.
    - Dodanie/usuwanie/zapisanie załączników z użyciem komponentów Chip oraz mechanizmu ActivityResultContracts.
- Usunięcie zadania z listy.
- Oznaczenie zadania jako zrobione.
- Generowanie przypomnienia o zadaniach w dniu ich ostatecznego terminu wykonania.
- Import i eksport listy zadań z/do pliku tekstowego.
- Nadanie zadaniom priorytetu z zakresu 1-10.
- Sortowanie listy zadań na dowolny z sześciu sposobów:
    - Wg. nazwy, malejąco.
    - Wg. nazwy, rosnąco.
    - Wg. priorytetu, malejąco.
    - Wg. priorytetu, rosnąco.
    - Wg. statusu, najpierw niewykonane.
    - Wg. statusu, najpierw wykonane.
- Wyświetlenie załączników wraz z ich nazwą i ikoną przedstawiającą typ pliku.
- Dodanie załącznika, w postaci lokalnego pliku, do zadania.
- Usunięcie załącznika z zadania.
- Zapisanie dowolnego załącznika lokalnie na urządzeniu.

Diagram przypadków użycia



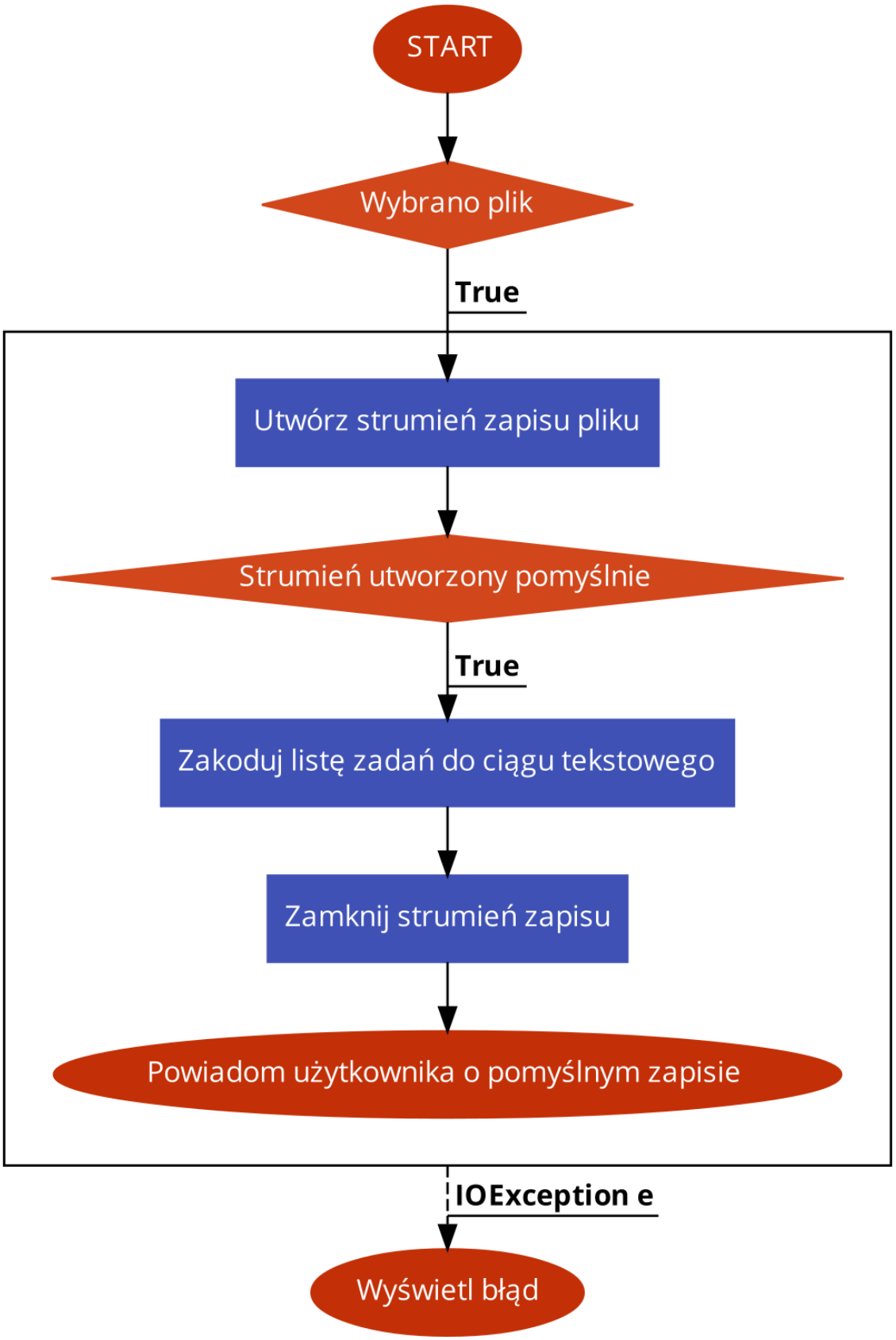
Opisy najważniejszych zastosowanych algorytmów

Obsługa zapisu zadań do pliku

Kod

```
createDocumentLauncher = registerForActivityResult(
        new ActivityResultContracts.CreateDocument(),
        uri -> {
            if(uri != null) {
                try {
                    OutputStream outputStream =
context.getContentResolver().openOutputStream(uri);
                    if (outputStream != null) {
                        String taskList_asString =
parseTasks(fragment_checkboxList.getTaskList());
                        outputStream.write(taskList_asString.getBytes());
                        outputStream.close();
                        Toast.makeText(context,
context.getString(R.string.file_saved_successfully), Toast.LENGTH_SHORT).show();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                    // Error occurred while writing the file
                }
            }
        }
);
```

Schemat blokowy

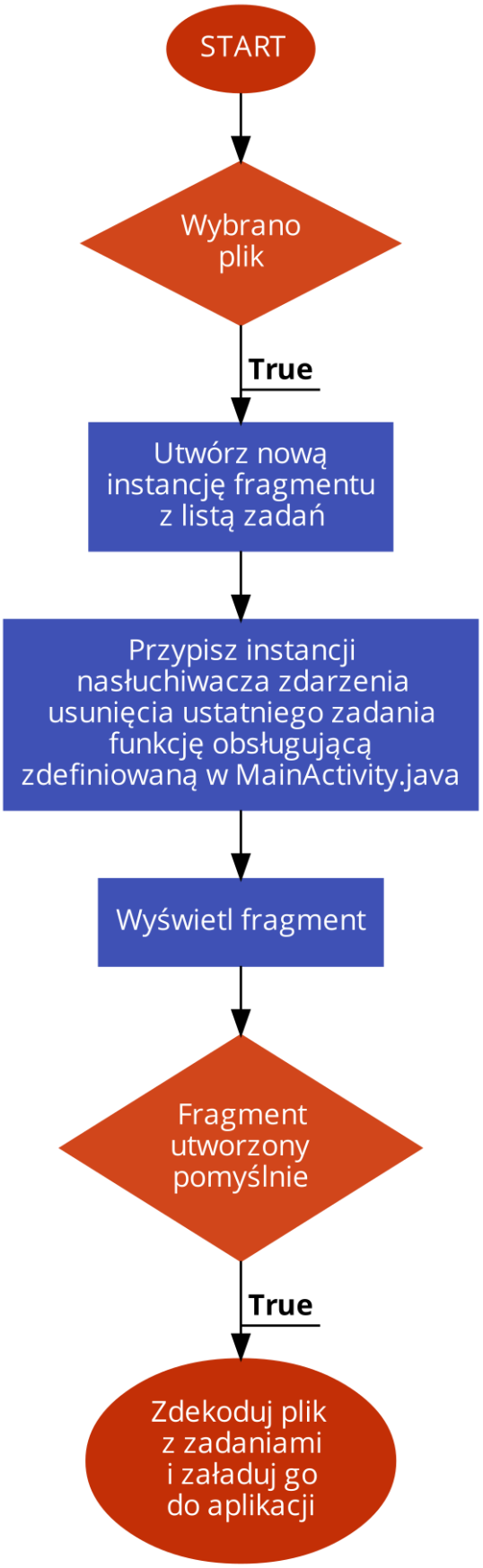


Obsługa odczytu zadań z pliku

Kod

```
fileContentLauncher = registerForActivityResult(
        new ActivityResultContracts.GetContent(),
        uri -> {
            if(uri != null) {
                fragment_checkboxList = Fragment_checkboxList.newInstance();
                fragment_checkboxList.setLastTaskRemovedListener(this);
                switchViews(R.id.rootLayout, fragment_checkboxList);
                if (fragment_checkboxList != null) {
                    fragment_checkboxList.updateTaskList(getTasksFromFile(uri));
                }
            }
        }
);
```

Schemat blokowy

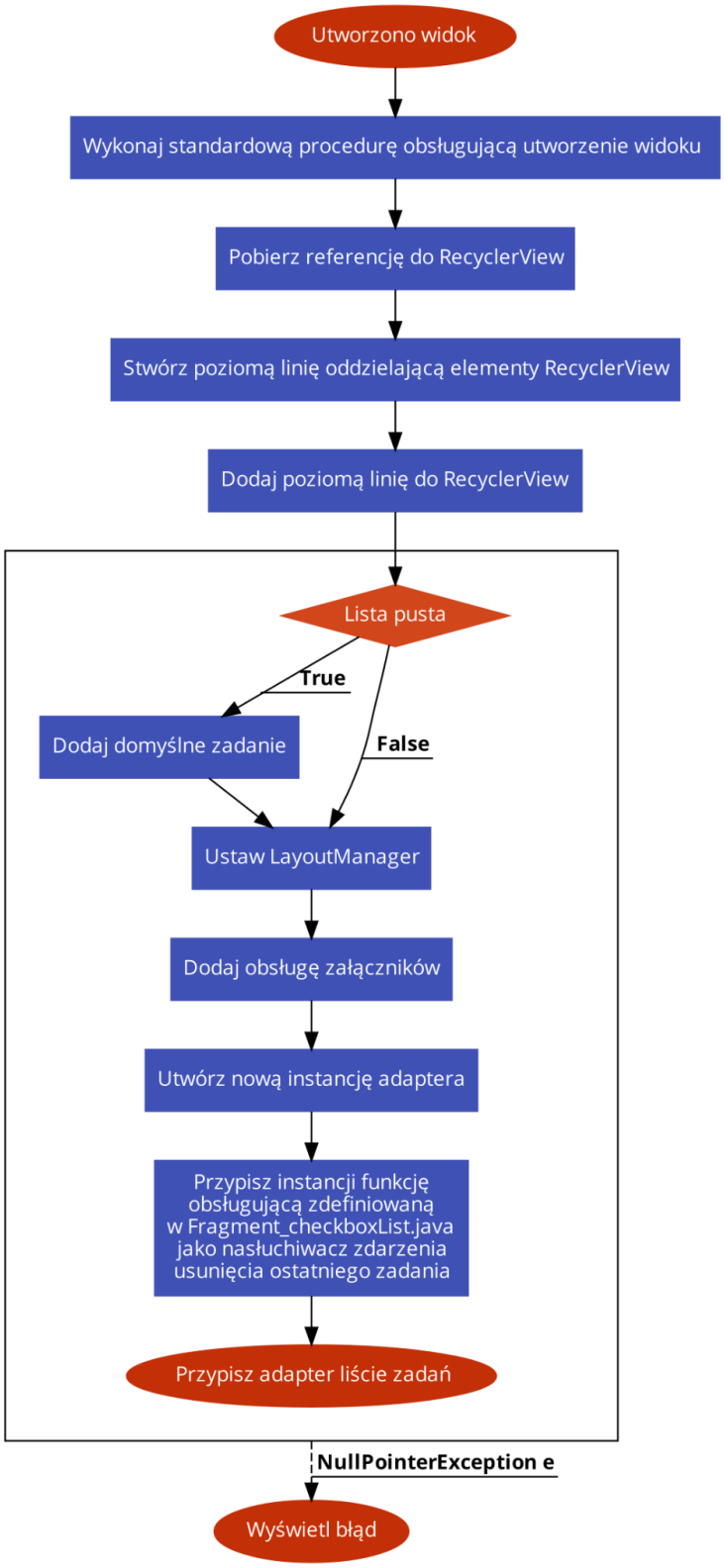


Obsługa zdarzenia utworzenia widoku Fragment_checkboxList

Kod

```
@Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        RecyclerView checkboxList = requireView().findViewById(R.id.recyclerView_checkboxList);

        /* Dodanie do recyclerView linii rozdzielającej elementy */
        RecyclerView.ItemDecoration dividerItemDecoration = new
DividerItemDecorator(ContextCompat.getDrawable(context,
android.R.drawable.divider_horizontal_bright));
        checkboxList.addItemDecoration(dividerItemDecoration);

        /* Dodanie pierwszego zadania */
        /* Musi być tutaj, ponieważ wymaga istnienia widoku tworzonego w poprzedniej metodzie,
onCreateView */
        try {
            /* Inicjalizacja taskList pierwszym zadaniem jeśli nie wczytano zadań z pliku */
            if(taskList.isEmpty()) {
                taskList.add(new Task(context));
            }

            /* Przygotowanie adaptera dla recyclerView, zainicjalizowanego danymi z taskList */
            checkboxList.setLayoutManager(new LinearLayoutManager(context));

            // Obsługa wyboru pliku podczas dodawania załącznika
            ActivityResultLauncher<String> attachmentContentLauncher =
registerForActivityResult(
                    new ActivityResultContracts.GetContent(),
                    uri -> {
                        if (uri != null) {
                            CheckboxListAdapter checkboxListAdapter = (CheckboxListAdapter)
checkboxList.getAdapter();
//                            int adapterPosition =
Objects.requireNonNull(checkboxListAdapter).getAdapterPosition();
//                            Task task = taskList.get(adapterPosition);
//                            task.addAttachment(uri);
//                            checkboxListAdapter.notifyItemChanged(adapterPosition); // model

Objects.requireNonNull(checkboxListAdapter).addAttachmentChip(checkboxListAdapter.getAttachmen
tChipGroup(), uri); // widok
                        }
                    }
            );
            checkboxListAdapter = new CheckboxListAdapter(context, this.taskList,
attachmentContentLauncher);
            checkboxListAdapter.setLastTaskRemovedListener(this);
            checkboxList.setAdapter(checkboxListAdapter);
        }
        catch(NullPointerException e) {
            Log.e(String.valueOf(R.string.fragment_checkboxlist_args_error), e.getMessage());
        }
    }
```

Schemat blokowy

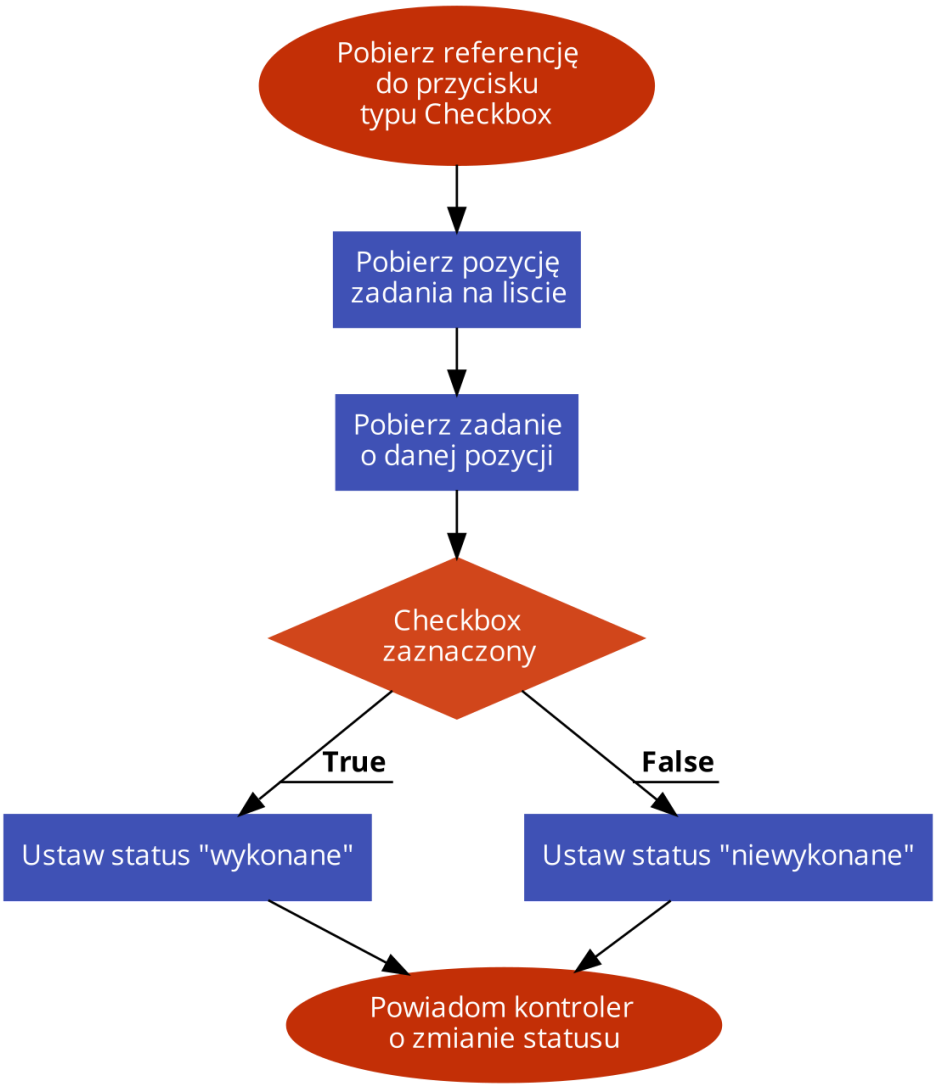


Przypisanie funkcji nasłuchującej kliknięcie w przycisk typu Checkbox, ustawiającej status zadania

Kod

```
taskCheckbox.setOnClickListener(view -> {
    adapterPosition = getAdapterPosition();
    task = checkboxList.get(adapterPosition);
    if (((AppCompatCheckBox) view).isChecked()) {
        task.setStatus(context.getString(R.string.done));
    } else {
        task.setStatus(context.getString(R.string.not_done));
    }
    notifyItemChanged(adapterPosition); // Powiadom kontroler o kliknięciu
});
```

Schemat blokowy



Dodanie wyniku gracza do rankingu 10 najlepszych wyników

Kod

```
/* Usuń zadanie po kliknięciu w przycisk deleteTask */
deleteTask.setOnClickListener(view -> {
    // A właściwie to najpierw zapytaj o potwierdzenie...
    AlertDialog.Builder builder = new AlertDialog.Builder(context);
    builder.setTitle(R.string.niebezpieczna_operacja);
    builder.setMessage(R.string.delete_task_confirmation_prompt);

    builder.setPositiveButton(R.string.usun, (dialog, id) -> {
        adapterPosition = getAdapterPosition();
        task = checkboxList.get(adapterPosition);
        checkboxList.remove(task);
        if(checkboxList.size() == 0) {
            // Powiadom fragment o usunięciu ostatniego zadania
            lastTaskRemovedListener.onLastTaskRemoved();
        }
        notifyItemRemoved(adapterPosition);
    });
    builder.setNegativeButton(R.string.cancel, (dialog, id) -> {});

    AlertDialog dialog = builder.create();
    dialog.show();

    Button deleteBtn = dialog.getButton(DialogInterface.BUTTON_POSITIVE);
    deleteBtn.setTextColor(Color.RED);
});
```

Schemat blokowy