Weather calendar

Tomasz Wojtasek, Paweł Kurek

Podstawowe założenia

Podstawowymi założeniami projektu są:

- Użytkownik może sprawdzić datę w kalendarzu oraz zaplanować spotkanie w wybranym dniu.
- Aplikacja ma zapisywać plany użytkownika wraz z zapisem do pliku.
- Program poprzez łączenie się z API ma pobierać informacje o stanie pogodowym, w tym: temperatura, zachmurzenie, ciśnienie, szansa na opady, opad śniegu, wschód i zachód słońca itd.
- Stworzenie przejrzystego interface’u.
- Wykorzystanie potencjału Java FX.
- Dokładne zaplanowanie architektury.

Wybrane technologie

Do projektu zastosowaliśmy takie technologie jak:

- GSON - Serializacja oraz deserializacja JSON’a.
- FXML - Tworzenie deklaratywnego interfejsu użytkownika dla JavaFX.
- Java FX - Stworzenie przejrzystego interface’u użytkownika.
- VisualCrossing API - Źródło danych pogodowych aplikacji.
- Wzorzec architektury warstwowej.
- CSS - Ustalenie stylów do kalendarza.

Wykorzystanie architektury warstwowej:

Wykorzystanie architektury warstwowej:

CalendarItem
CalendarButton
CalendarLabel
StageAssistant
AlertException
AlertSucces

klasy wspomagające UI.

uiutil

ScheduledEvent
WeatherLocation
WeatherDay
WeatherQuery

klasy określające dane
z minimalną logiką biznesową

Model

GlobalStateAssistant
QueryAssistant
GlobalStateException
ApiException

Klasy wspomagające.
Np. ogólna komunikacja z API,
zapis do pliku binarnego
(metody statyczne, typy generyczne)

util

AppConstants

ustawienia aplikacji w trakcie
kompilacji.
W przyszłości dedykowany
plik z configiem (ustawienia w trakcie
działania aplikacji)

config

Relacja controller-service-manager

EventController

```
@FXML
private void saveData() {
    try {
        String location = this.comboBoxLocation.getSelectionModel().getSelectedItem();
        this.eventService.addEvent(selectedItem, this.textFieldEventName.getText(),
                this.textAreaEventDesc.getText(), location);
        this.weatherService.updateWeather(selectedItem, location);
        new AlertSucces("Pomyślnie udało się zapisać nowe spotkanie.").showAndWait();
    }
    catch(Exception ex) {
        new AlertException(ex).showAndWait();
    }
}
```

obiekt selectedItem (klasa CalendarItem) jest referencją do wybranego przez użytkownika dnia w kalendarzu (wybranego w klasie CalendarController)

EventService

```
public void addEvent(CalendarItem item, String eventName, String eventDesc, String location) {
    this.validateCalendarItem(item);
    this.validateEventName(eventName);
    this.validateEventDesc(eventDesc);
    this.validateLocation(location);

    ScheduledEvent newEvent = new ScheduledEvent(eventName, eventDesc, location);

    this.eventManager.addEvent(item.getDate(), newEvent);
    this.bindEventToCalendarItem(item, newEvent);
}
```

EventManager

```
public void addEvent(LocalDate date, ScheduledEvent event) {
    events.put(date, event);
}
```

WeatherService

```
public void updateWeather(CalendarItem item, String location) throws ApiException{
    this.validateCalendarItem(item);
     try {
        WeatherQuery result = this.makeQuery(location);
        WeatherLocation weatherLocation = this.weatherManager.getOrCreateWeatherLocation(location);

        for(WeatherDay weatherDay : result.getDays()) {
            LocalDate date = item.getDate();
            LocalDate queryDate = LocalDate.parse(weatherDay.getDatetime());
            if(date.equals(queryDate)) {
                // aktualizacja modelu tylko dla określonego dnia
                weatherLocation.addWeatherDay(weatherDay);
                this.bindWeatherIconToCalendarItem(item, weatherDay);
            }
        }
    }
    catch(ApiException ex) {
        throw new ApiException("Wystąpił błąd podczas aktualizowania danych pogodowych.", ex);
    }
}
```

WeatherManager

```
public WeatherLocation getOrCreateWeatherLocation(String location) {
    WeatherLocation weatherLocation = this.getWeatherLocation(location);
    if (weatherLocation == null) {
        weatherLocation = new WeatherLocation();
        this.addWeatherLocation(location, weatherLocation);
    }
    return weatherLocation;
}
```

Zwraca referencje do istniejącej WeatherLocation.

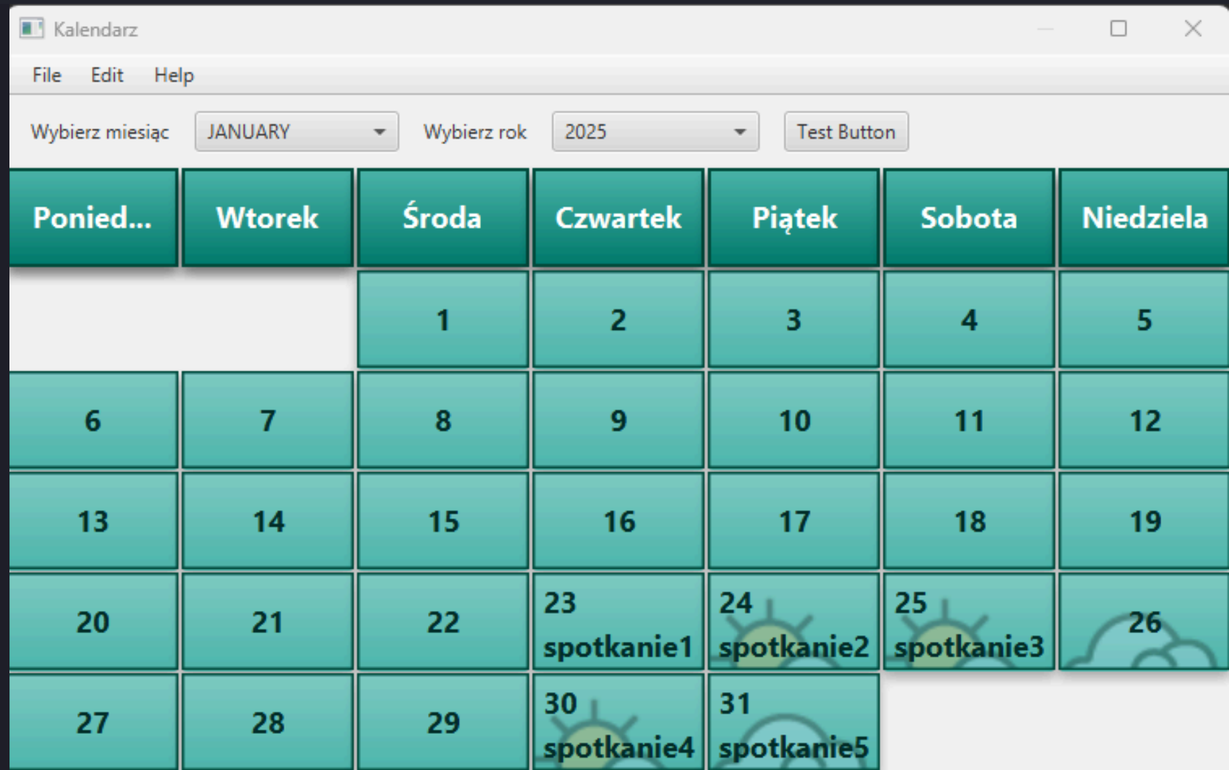
Gdy obiekt nie istnieje, tworzy nowy.

Sposób implementacji

W aplikacji można wyróżnić:

- Wybierz miesiąc
- Wybierz rok
- Kalendarz
- Konkretny dzień

```
private void fillComboBoxYears() {
    ObservableList years = FXCollections.observableArrayList();
    int curYear = Year.now().getValue();
    for (int i = 0; i < AppConstants.YEARS_FORWARD_SCOPE; i++) {
        years.add(Year.of(curYear + i));
    }
    this.comboBoxYears.setItems(years);
    if(!this.comboBoxYears.getItems().isEmpty()){
        this.comboBoxYears.getSelectionModel().select(Year.now());
    }
    else {
        // SHOW ALERT
    }
}
```



Sposób implementacji

Sposób generacji kalendarza w kontrolerze:

```
@FXML
public void loadCalendarData(){
    Year year = this.comboBoxYears.getSelectionModel().getSelectedItem();
    Month month = this.comboBoxMonths.getSelectionModel().getSelectedItem();
    this.cleanCalendar();
    try {
        this.calendarService.generateCalendar(year, month, true, true).forEach(item -> {
            CalendarButton button = item.getButton();
            button.setOnAction(e -> calendarButton_click(item));
            this.gridPaneCalendar.add(button, item.getColumn(), item.getRow());
        });
    }
    catch(ApiException ex) {
        // SHOW ALERT
    }
}
```

Dziękujemy za uwagę

Tomasz Wojtasek, Paweł Kurek