

Java projekt

Zespół wykonujący: Karol Borowski, Jakub Bębacz

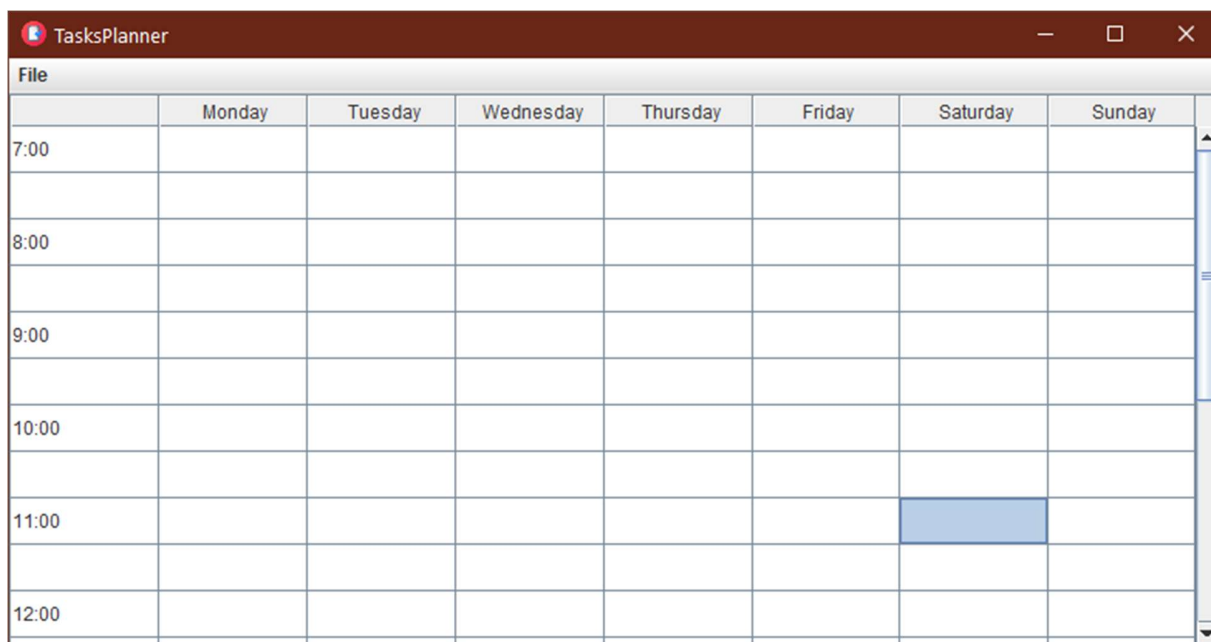
Grupa: 2ID11A

Temat projektu: *Tasksplanner*

Opis programu

Program ma za zadanie ułatwić wspólne planowanie zadań. Uruchamiając program naszym oczom ukazuje się rozkład dnia z polami w które użytkownik może dodać event o określonej porze i o określonym typie. Aplikacja działa na zasadzie klient-serwer oraz zapisuje utworzone wydarzenia do bazy. Backend opierający się na Springu pozwoli na jednoczesne otwarcie kilku aplikacji klienckich i bez problemu obsłuży nadchodzące z nich zapytania.

Screenshots



Create n...

Event name

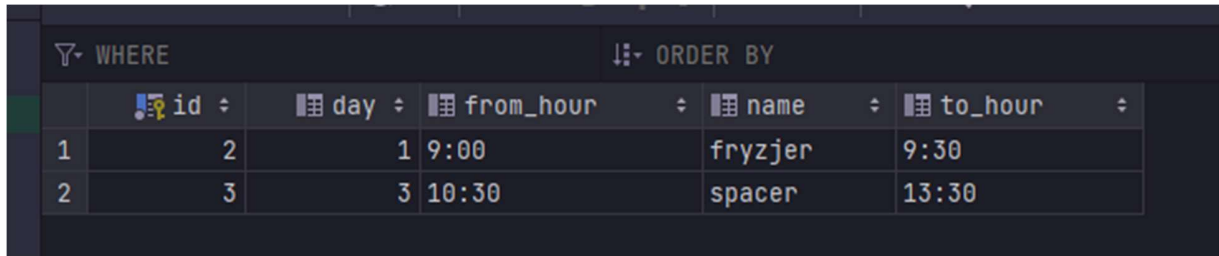
From: 9:00 To: 9:30

Accept

TasksPlanner							
File							
	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
9:00	Zakupy						
10:00							
11:00							
12:00							
13:00							

Baza danych

Backend aplikacji korzysta z informacji o eventach zapisanych w bazie danych. Tam każdy event dodany z poziomu klienta, jest przechowywany i pobierany w sytuacji, gdy klient tego zażąda. Baza na której działa program to PostgreSQL.



The screenshot shows a PostgreSQL query result with columns: id, day, from_hour, name, and to_hour. The results are as follows:

	id	day	from_hour	name	to_hour
1	2	1	9:00	fryzjer	9:30
2	3	3	10:30	spacer	13:30

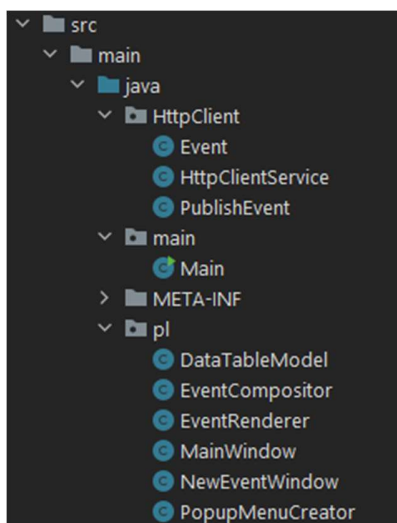
Serwer

Serwer aplikacji jest wykonany we frameworku Spring. Jest to typowo backendowy framework, który opiera swoje działanie na zapytaniach o protokole HTTP. Serwer zawiera pewne endpointy, do których odwołujemy się z aplikacji klienckiej. Na przykład, gdy chcemy dodać nowy event, klient pobiera dane z GUI i wywołuje odpowiedni endpoint wraz zserializowanym body. Serwer odbiera zapytanie i jednocześnie wysyła dane do bazy danych Postgre.

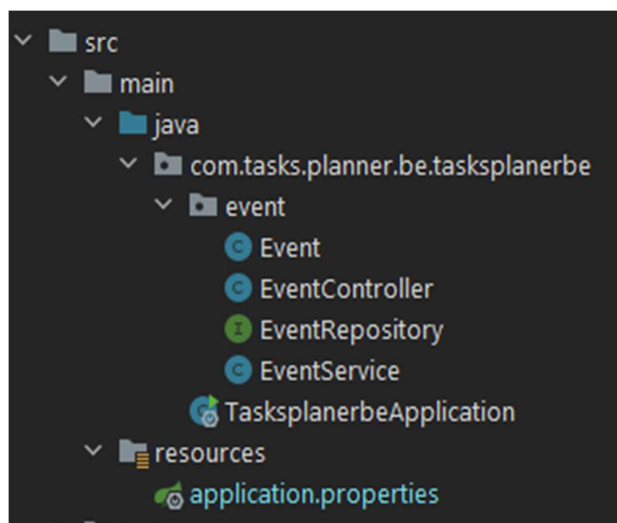
Opis struktury aplikacji

Program dzieli się na dwie aplikacje: klient oraz serwer. Klient zawiera GUI oraz załączone odpowiednie biblioteki umożliwiające odwoływanie się do konkretnych endpointów na serwerze. Serwer natomiast posiada typową dla backendu strukturę kontrolerów oraz serwisów.

Drzewo klienta:



Drzewo server:



Klient

Klient opiera się w głównej mierze na komponencie JTable do którego dodajemy eventy. W nim wyświetlają się wszystkie informacje potrzebne do zidentyfikowania danego eventu. Aby mógł on działać musieliśmy stworzyć, na potrzeby tej aplikacji, własny model tabeli:

```
package pl;
import javax.swing.table.AbstractTableModel;
public class DataTableModel extends AbstractTableModel {
    private final String[] columnNames = {
        "", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
        "Saturday", "Sunday"
    };
    private final Object[][] data;
    public DataTableModel(int numberOfHours, int startHoursFrom) {
        data = new Object[numberOfHours*2+1][8];
        Object[] hoursLabels = new Object[numberOfHours*2+1];
        int counterFullHours = 0;
        for(int i=0; i<hoursLabels.length; i++) {

            if(i%2 == 0) {
                hoursLabels[i] = counterFullHours + startHoursFrom + ":00";
                counterFullHours++;
            } else hoursLabels[i] = "";
        }

        for(int i=0; i< data.length; i++) {
            for(int j=0; j<data[i].length; j++) {
                if (j == 0) {
                    data[i][j] = hoursLabels[i];
                } else data[i][j] = "";
            }
        }
    }

    @Override
    public int getRowCount() {
        return data.length;
    }

    @Override
    public int getColumnCount() {
        return columnNames.length;
    }

    @Override
    public String getColumnName(int column) {
        return columnNames[column];
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        return data[rowIndex][columnIndex];
    }

    public void setValueAt(Object aValue, int row, int col) {
        data[row][col] = aValue;
        fireTableCellUpdated(row, col);
    }
}
```

Miejszem, gdzie program tymczasowo przechowuje potrzebne dane jest ***Vector<SingleEvent> components*** . to w nim znajdują się wydarzenia, które aplikacja ma wyrenderować. Owy wektor jest typu ***SingleEvent***, co oznacza, że przechowuje dane z stworzonej klasy ***SingleEvent***:

```
public class SingleEvent {
    String name;
    String fromHour;
    String toHour;
    int day;
}
```

Samo zapisanie do vectora, nie wyświetli użytkownikowi danych o ewencie, dlatego też musiała zostać utworzony customowy Cell Renderer:

```
package pl;
import javax.swing.*;
import javax.swing.table.DefaultTableCellRenderer;
import java.awt.*;

public class EventRenderer extends DefaultTableCellRenderer {
    Color backgroundColor;

    public EventRenderer(Color backgroundColor) {

        this.backgroundColor = backgroundColor;
    }
    @Override
    public Component getTableCellRendererComponent(JTable table, Object
value, boolean isSelected, boolean hasFocus, int row, int column) {
        Component renderComp = super.getTableCellRendererComponent(table,
value, isSelected, hasFocus, row, column);

        if (MainWindow.signatures[column][row])
            renderComp.setBackground(backgroundColor);
        else
            renderComp.setBackground(Color.WHITE);
        return renderComp;
    }
}
```

Pobiera on konkretną komórkę z tabeli i na podstawie tabeli `MainWindow.signatures[][]` ją koloruje jeśli warunek zostanie spełniony.

Tabela `signatures[][]` to dwuwymiarowa struktura boolowska, która określa nam jaka komórka powinna zostać zamalowana. Mianowicie, specjalny algorytm ustawia wartość *true* jeśli w danej komórce powinien widnieć event.

Tak przygotowana struktura musi jeszcze zostać odpowiednio wywołana. Do tego celu posłuży:

```
static void update() {
    for(int i=0 ;i<8; i++)
        for(int j=0; j<numberOfHours*2+1; j++) {
            signatures[i][j] = false;
        }

    int currentHandleColumn;
    EventCompositor ev = EventCompositor.getInstance();
    Iterator<EventCompositor.SingleEvent> it = ev.components.iterator();
    while(it.hasNext()) {
        EventCompositor.SingleEvent singleEvent = it.next();
        currentHandleColumn = singleEvent.day;
        int indexFrom = getNumberOfRowFromHour(singleEvent.fromHour);
        int indexTo = getNumberOfRowFromHour(singleEvent.toHour);
        timetable.setValueAt(singleEvent.name, indexFrom,
            singleEvent.day);
        for(int i=0; i<numberOfHours*2; i++) {
            if(i>=indexFrom && i <= indexTo) {
                signatures[currentHandleColumn][i] = true;
            }
        }
    }
    for(int i=0; i<8; i++) {
        TableColumn col =
            MainWindow.timetable.getColumnModel().getColumn(i);
        col.setCellRenderer(new EventRenderer(Color.magenta));
    }
    MainWindow.timetable.repaint();
}
```

który ustawia całą tabelę *signatures* na *false*, następnie pobiera wszystkie dane z wektora *components* i ustawia odpowiednie wartości na *true*. Następnie wywołuje renderer dla kolejnych 8 kolumn i w taki sposób odświeża tabelę.

W pliku **HttpClientService.java** znajdują się metody pozwalające odwołać się do backendowych kontrolerów.

```
public void postEvent(PublishEvent event) throws IOException {
    RequestBody body = RequestBody.create(JSON,
mapper.writeValueAsString(event));
    Request request = new Request.Builder()
        .url(BASE_URL + "add")
        .post(body)
        .build();
    try (Response response = client.newCall(request).execute()) {
        if(response.isSuccessful()) {
            System.out.println("Successfully");
        } else {
            System.out.println(response.message());
        }
    }
}
```

Powyższa metoda pozwala na uploadowanie nowego eventu. Metoda serializuje body i wysyła je w stronę serwera do metody **add**.

Pobieranie danych obsługuje metoda:

```
public List<Event> getAllEvents() throws IOException {
    Request request = new Request.Builder()
        .url(BASE_URL + "all")
        .build();
    try (Response response = client.newCall(request).execute()) {
        if (response.isSuccessful()) {
            ResponseBody body = response.body();
            String value = body.string();
            return mapper.readValue(value,
mapper.getTypeFactory().constructCollectionType(List.class, Event.class));
        }
    }
    return Collections.emptyList();
}
```

Zwraca ona wszystkie eventy w formie jednego obiektu, który bardzo łatwo można przekonwertować do listy **components**:

```
List<Event> list = null;
    try {
        list = service.getAllEvents();
    } catch (IOException ioException) {
        ioException.printStackTrace();
    }
    for(int i=1; i<8; i++) {
        for (int j=0; j<numberOfHours*2+1; j++) {
            timetable.setValueAt("", j, i);
        }
    }
    EventCompositor ev = EventCompositor.getInstance();
    ev.components.clear();
    Iterator<Event> it = list.iterator();
    while(it.hasNext()) {
        Event event = it.next();
        EventCompositor.SingleEvent singleEvent = ev.new SingleEvent();
        singleEvent.name = event.getName();
        singleEvent.fromHour = event.getFromHour();
        singleEvent.toHour = event.getToHour();
        singleEvent.day = event.getDay();
        ev.components.add(singleEvent);
    }
}
```

Serwer

Zasada działania serwera opiera się na kontrolerach. Zapytanie http zostaje wysłane na adres serwera, a odpowiedni endpoint reaguje na to zdarzenie.

```
public class EventController {

    private final EventService eventService;
    @Autowired
    public EventController(EventService eventService) {
        this.eventService = eventService;
    }
    @GetMapping("all")
    public List<Event> getEvents() {
        return eventService.getEvents();
    }
    @PostMapping("add")
    @ResponseStatus(HttpStatus.CREATED)
    public Event createEvent(@RequestBody Event event) {
        return eventService.addEvent(event);
    }
    @DeleteMapping("drop")
    public void deleteAll() {
        eventService.dropAll();
    }
};
}
```


Wykonując zapytanie **add** z odpowiednio przygotowanym body, kontroler przekazuje pracę serwisowi.

```
public Event addEvent(Event event) {  
    return eventRepository.save(event);  
}
```

Ten natomiast zapisuje dane przy pomocy repo w bazie oraz zwraca response.

Plik **application.properties** zawiera konfigurację bazy danych.