Łukasz Mieczkowski

PersonalCalendar

This document briefly describes project which was created as job interview task. Source code can be found on my GitHub account: https://github.com/mieczyk/PersonalCalendar.

Described project is not a fully working application. Project can be considered as scaffolding for more complex solution. It takes an advantage of N-tier architecture approach.

Used technologies and libraries

- ASP.NET MVC 5.1
- Entity Framework 6
- Unity IoC Container
- SQL Server 2014
- NUnit
- Moq
- Bootstrap
- jQuery

Project structure

Solution consists of following projects:

- `PersonalCalendar.Data` – Data access layer which contains Entity Framework's DbContext. I've resigned from Repository pattern as EF ORM can be treated as repository layer. Project also contains SQL script for database creation.

- `PersonalCalendar.Domain` – Represents application's domain model. It stores POCO classes, which are mapped to proper database tables, and other entities which compose business logic layer.

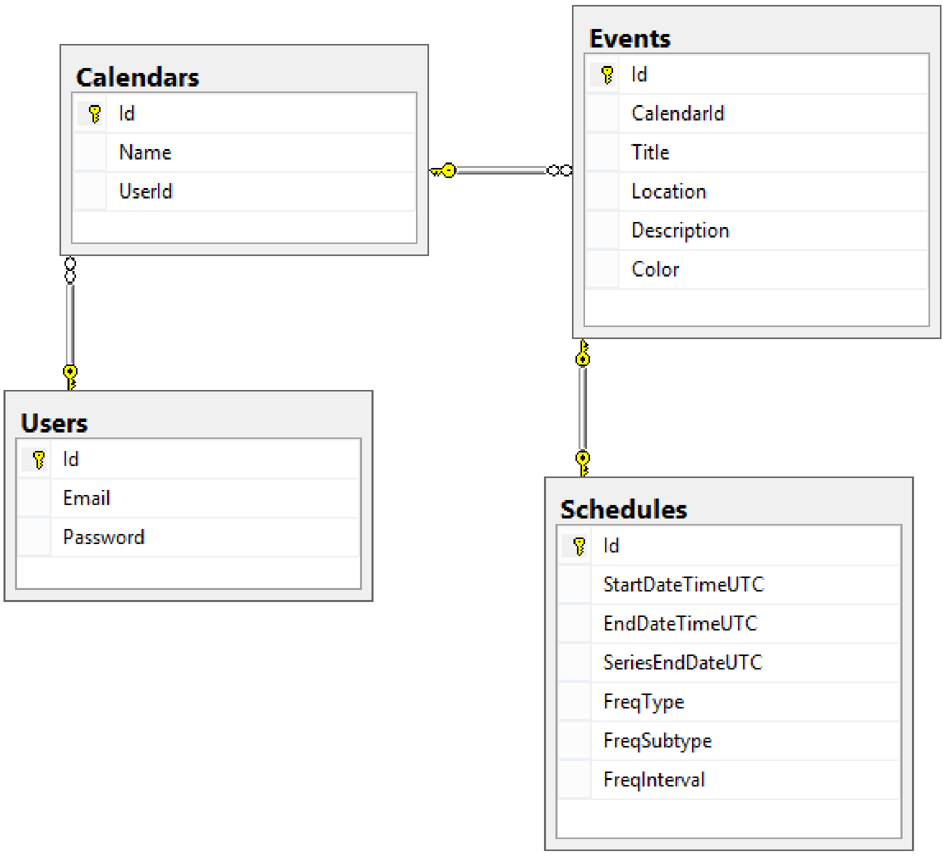
- `PersonalCalendar.Infrastructure` – Contains code that helps to build application's infrastructure (not business layer). E.g. extension methods.

- `PersonalCalendar.Service` – Services layer: business logic responsible for operating on entities (retrieving, saving, deleting, etc.).

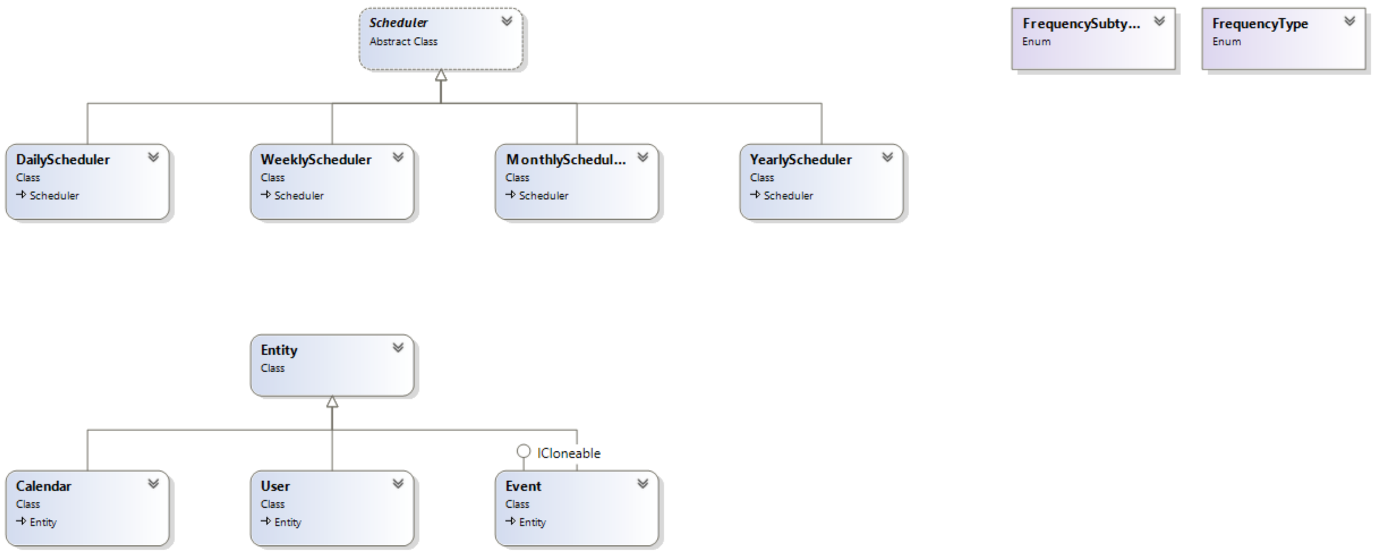
- `PersonalCalendar.Tests` – Unit and integration tests with usage of NUnit and Moq frameworks.

- `PersonalCalendar.Web` – UI layer. Web application implementing MVC pattern.

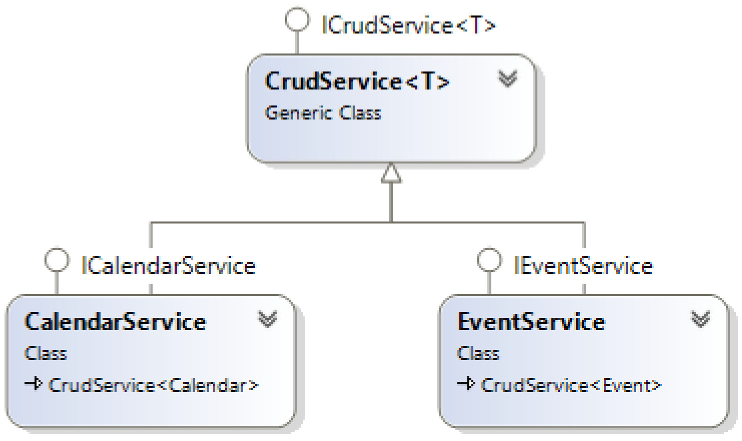
Database structure

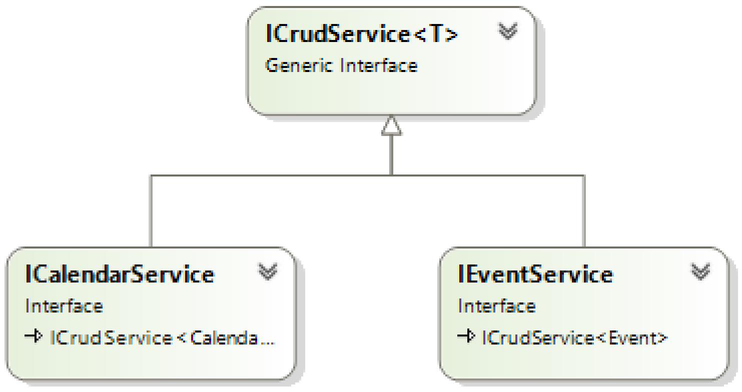


Class diagram



PersonalCalendar.Domain





PersonalCalendar.Service

Solution description

Issue: *scheduling events: one-time / full day / recurring on different intervals (same intervals that Google is using)*

Both one-time and recurring events are represented by *Event* class, which is mapped to two tables: *Events* and *Schedules* (1-1 relationship). When event occurs only once, then *Schedules* table entry is irrelevant. The columns description below explains, how particular event types are stored in database:

- *Schedules.StartDateTimeUTC* - Represents start date time of particular event. When it's recurring event, this date time also represents start of the series.
- *Schedules.EndDateTimeUTC* – Represents end date time of particular event. For example event can last from 13.03.2015 to 14.03.2015 (all day event).

- *Schedules.SeriesEndDateUTC* – Represents the finish date of event's series. When it's NULL, it means that event series should never stop.
- *Schedules.FreqType* – Frequency type. This field decides whether event is one-time or recurring:
    - 0 = one-time event,
    - 1 = daily recurring event,
    - 2 = weekday recurring event (except Saturday and Sunday) [not implemented],
    - 4 = only on odd weekdays [not implemented],
    - 8 = only on even weekdays [not implemented],
    - 16 = weekly recurring events,
    - 32 = monthly recurring events,
    - 64 = yearly recurring events.
- *Schedules.FreqSubtype* – Frequency subtype determines options for particular frequency type. Meaning of this value is dependent on *FreqType* value:
    - 0 = none (default),
    - 1 – 127 = represents weekdays combination by combining bit numbers. E.g. Monday (1) and Saturday (32) combination is represented by 33 (1 and 32 combined with bitwise 'or' operator). This range is used only when *FreqType=16* (weekly recurring events).
    - 128 = day of the month. Used only with *FreqType=32*,
    - 256 = day of the week. Used only with *FreqType=32*.
- *Schedules.FreqInterval* – Frequency interval. Meaning of this value depends on *FreqType* option. For example, *FreqType=1* (daily recurring event) determines that this value stores interval in days and so on.

Issue: *editing scheduled events (one-time events / event series / single event in a series)*

This functionality is not implemented in delivered code, but I will explain how it could be done. For one-time events editing means standard modification of event with given ID. Editing of event series is more challenging. Event series is a list of different occurrences of THE SAME event. When we want to edit recurring event, we have 3 options (like in Google Calendar):

- This event only – we could duplicate selected event (create new record) and set it as one-time event. We also would have to split interrupted series into two different recurring events (with the same data though).
- Following events – we could split editing series into two recurring events (we duplicate recurring event and change data for duplicate).
- All events – The easiest variant – we have to modify only one row in database (like in one-time event case).

Issue: *displaying events on a calendar control (both one-time events and event series infinitely in the future)*

I have implemented displaying events for current month view.

Issue: *sending reminder emails about upcoming tasks.*

Not implemented, but I would create dedicated service responsible for email notification. I would also use Quartz.NET library for sending emails job scheduling.

Issue: *consider possible differences in the algorithm depending on a country if you find any.*

I store dates in UTC in order to eliminate time zone's differences. Unfortunately I didn't have time to get user's local time so I recalculate dates using server's time zone settings.

I also skipped user's authorization/authentication mechanism for simplicity. Right after database creation, sample user with assigned calendar is created.