Deutsche Telekom IT Solutions

Author: Kozin Anton

**Technical document for**

**Rehab Application.**

Saint-Petersburg, 2021

Content

[Task 3](#_Toc89077759)

[Application description 4](#_Toc89077760)

[Technologies 6](#_Toc89077761)

[Architecture 7](#_Toc89077762)

[User interface 13](#_Toc89077763)

[JUnit tests 20](#_Toc89077764)

[Possible improvements 23](#_Toc89077765)

# Task

The main task is to develop web-application for rehabilitation hospital for electronic document flow between employees. There are different types of functions that should have different employees (depending on what job position they have). At least, the application should have special functions only for doctors and nurses. For example, creating new patients, making diagnosis, prescribing cures – for doctors, and manipulating with events – for nurses.

Also, there should be a separate application (electronic display) that will be showing specific information (e.g., events planned for today date) for patients.

# Application description

The application is developed for rehabilitation hospital, for electronic document flow between employees.

Any user has to be authenticated to use the application.

Depending on what role user has, the application provides different functions for them.

The following actions are available for any role:

* see lists of events, patients, cures
* see their own profile, change their passwords
* create new cures

The following actions are available for role «DOCTOR»:

* see lists of treatments, prescriptions
* create and discharge patients
* create and close treatments
* create, cancel and update prescriptions

The following actions are available for role «ADMIN»:

* see list of employees
* create employees, change their passwords
* create and edit patients

The following actions are available for role «NURSE»:

* choose, discard, perform and cancel events

When patient comes to rehabilitation hospital, doctor creates a treatment and makes diagnosis. But if it is the first visit, at first, the doctor creates patient. After that, the doctor creates prescription, chooses particular cure and time pattern. There are two types of time pattern: several times a day (for example, 2 times a day (e.g., morning and evening)) and several times a week (for example, 3 times a week (e.g., Monday, Wednesday, Friday)). The doctor chooses time period, as well (for example, 5 days, 2 weeks, 1 month etc.). After that, the application creates events, based on time pattern and period of time. The doctor can create several prescriptions based on current treatment.

Any nurse can choose any planned event to become a performer for the event. The nurse can perform or cancel event (if there is a reason). The nurse can discard event, as well, then another nurse can choose this event.

The doctor can cancel the prescription (but only prescription that was created by him/her). In this case, all planned events for this prescription will be cancelled.

The doctor can update the prescription (but only prescription that was created by him/her), as well. In this case, all planned events for this prescription will be cancelled and new prescription will be created (based on cancelled prescription), then the doctor can change cure, pattern and period.

The doctor can close the treatment (but only treatment that was created by him/her). Before closing the treatment must have only inactive prescriptions.

The patient can have several treatments at the same time (even created by different doctors).

The doctor can discharge the patient, but only if all patient’s treatments are closed.

We can consider Treatments as an additional implemented feature. Since a patient can have several treatments at the same time, it means the patient can have several different diagnoses at the same time (made by different doctors). Also, we can see all treatments for the patient, therefore the medical history is saved and the new diagnoses do not erase the previous diagnoses.

# Technologies

Back-end part of the application is written on Java (JDK 15). Since it is web-application, front-end part is written using HTML and CSS.

Here is the list of technologies that were used in the first application:

* Maven
* PostgreSQL
* Spring Boot
* Spring Data-JPA
* JSP
* Bootstrap
* jQuery

Here is the list of technologies that were used in the second application:

* Gradle
* Enterprise JavaBeans
* JSF

The first application notifies the second one (about changes in list of events) using message broker – RabbitMQ.

# Architecture

The application is divided on packages.

List of the packages and their descriptions are below:

* com.rehab.model – contains classes of domain model (entities)
* com.rehab.repository – contains interfaces to manipulate with entities in database
* com.rehab.dto – contains DTO classes for model classes. They are used for transferring data between services and controllers
* com.rehab.service – contains services. These classes are considered to perform business logic of the application
* com.rehab.controller – contains controllers, that executes http request from users
* com.rehab.exception – contains custom exception class and exception handler
* com.rehab.util – contains classes with static utility methods
* com.rehab.config – contains classes to make configurations relating to web security, message queue, and creates all unnecessary beans.

The main scenario of interaction modules between each other is: a user performs an action, controller executes http request, controller calls service, service maps given dto to entity (if it is necessary), service calls repository, repository manipulates with database, repository returns the result to service, service maps given entity to dto (if it is necessary), service returns the result to controller, the user gets response.

List of domain model classes:

* Employee – employee of rehabilitation hospital which is a user of this application, as well.
* Patient – patient which is a client of rehabilitation hospital.
* Cure – particular cure which is used for patient's treating.
* Treatment – created by doctor when patient comes to rehabilitation hospital in order to get medical help.
* Prescription – it is that doctor creates to prescribe cure for patient, with particular frequency and for particular period of time.
* Period – particular period of time, during of that patient has to take medicines or to be treated with procedure.
* Pattern – how often patient has to take medicines or to be treated with procedure.
* Event – event which is supposed to be performed by nurse. It contains information about patient, what day/time and which cure they have to be given.

Service classes perform all business logic. They are called from controllers. Some methods are being executed in transactions, since they have several calls to database for reading and saving data. Mostly, any service has methods for getting, creating and filtering.

List of service classes:

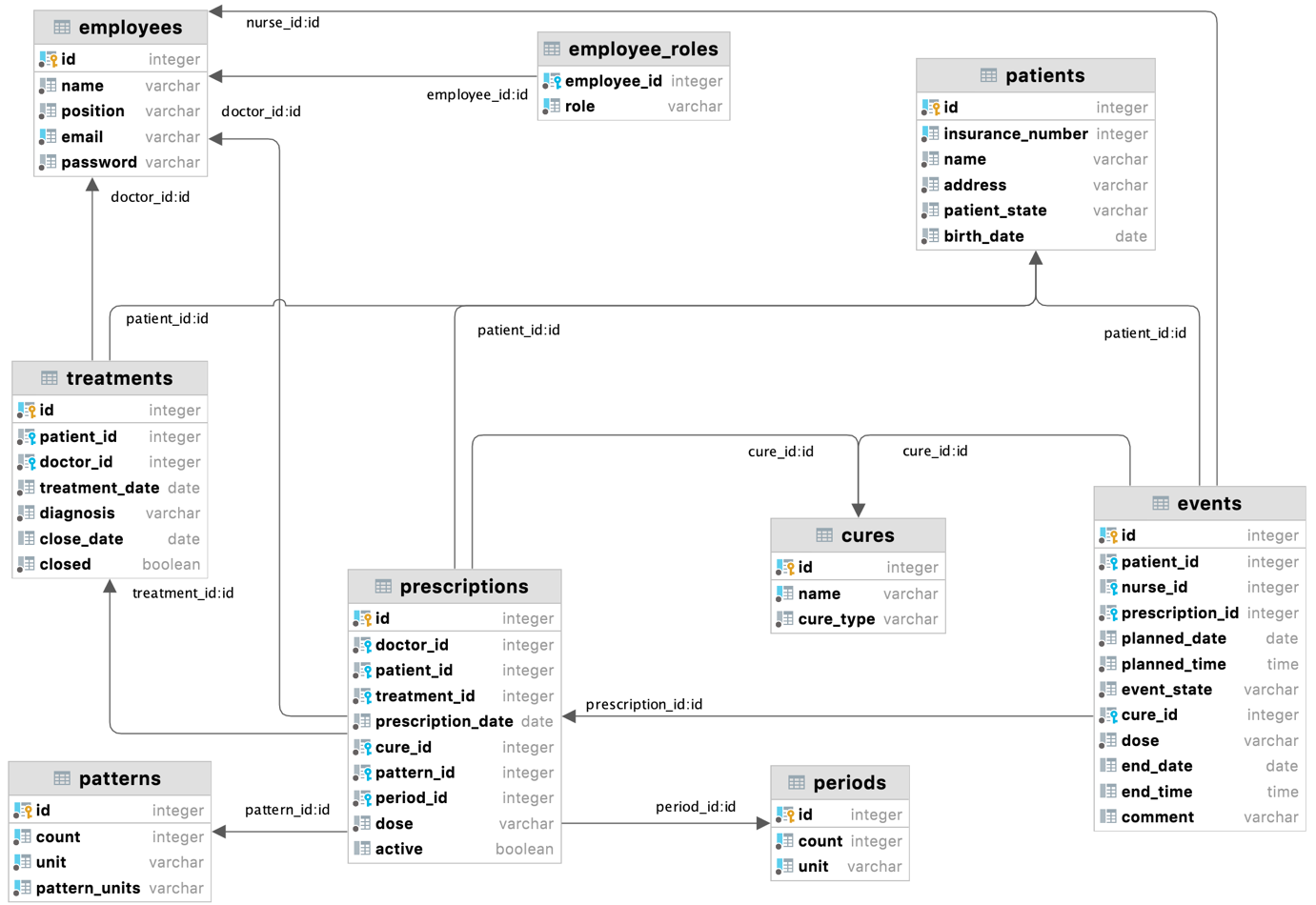
* CureService – service class for Cure. It operates with Cure, CureDto.
* EmployeeService – service class for Employee. It operates with Employee, UserDto, EmployeeDto.
* EmployeeDetailsService – service class for EmployeeDetails.
* EventService – service class for Event. It operates with Event, EventDto.
* PatientService – service class for Patient. It operates with Patient, PatientDto.
* PrescriptionService – service class for Prescription. It operates with Prescription, PrescriptionDtoOut, PrescriptionDto.
* TreatmentService – service class for Treatment. It operates with Treatment, TreatmentDto.

List of repository interfaces:

* CureCrudRepository – allows working with cures entities in database.
* EmployeeCrudRepository – allows working with employees entities in database.
* EventCrudRepository – allows working with events entities in database.
* PatientCrudRepository – allows working with patients entities in database.
* PatternCrudRepository – allows working with patterns entities in database.
* PeriodCrudRepository – allows working with periods entities in database.
* PrescriptionCrudRepository – allows working with prescriptions entities in database.
* TreatmentCrudRepository – allows working with treatments entities in database.

Database contains the following tables:

* employees – table of rehabilitation hospital’s employees, they are users of the application as well. Columns: id, name, position, email, password.
* employee\_roles – table stores employee’s roles. Columns: employee\_id (refers to column “id” in table “employees”), role.
* patients – table of patients (clients of rehabilitation hospital). Columns: id, insurance\_number, name, address, patient\_state (treating or discharged), birth\_date.
* treatments – table of treatments (when patient comes to rehabilitation hospital to get medical help). Columns: id, patient\_id (refers to column “id” in table “patients”), doctor\_id (refers to column “id” in table “employees”), treatment\_date (when treatment was created), diagnosis, close\_date (when treatment was closed), closed (or not).
* cures – table of cures. Columns: id, name, cure\_type (medicine or procedure).
* patterns – table of time patterns for prescriptions. Columns: id, count, unit, pattern\_units.
* periods – table of time periods for prescriptions. Columns: id, count, unit.
* prescriptions – table of prescriptions (that doctor makes for patient). Columns: id, doctor\_id (refers to column “id” in table “employees”), patient\_id (refers to column “id” in table “patients”), treatment\_id (refers to column “id” in table “treatments”), prescription\_date (when prescription was created), cure\_id (refers to column “id” in table “cures”), pattern\_id (refers to column “id” in table “patterns”), period\_id (refers to column “id” in table “periods”), dose, active (or not).
* events – table of events. Columns: id, patient\_id (refers to column “id” in table “patients”), nurse\_id (refers to column “id” in table “employees”), prescription\_id (refers to column “id” in table “prescriptions”), planned\_date, planned\_time, event\_state (planned, performed or cancelled), cure\_id (refers to column “id” in table “cures”), dose, end\_date, end\_time, comment.



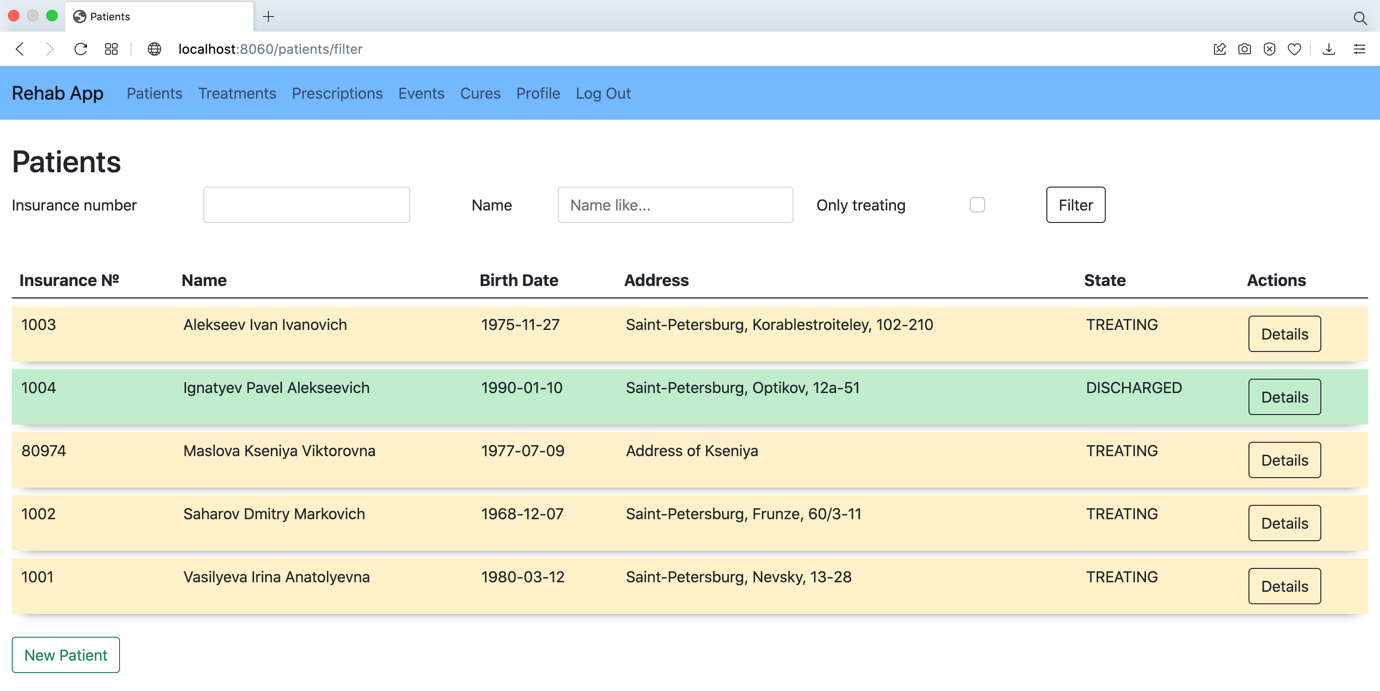
# User interface

HTML and CSS is the basis of UI. Many elements were written using Bootstrap (e.g., forms, buttons, modal windows). Some features are implemented with jQuery (e.g., disabled buttons, depending on some conditions).

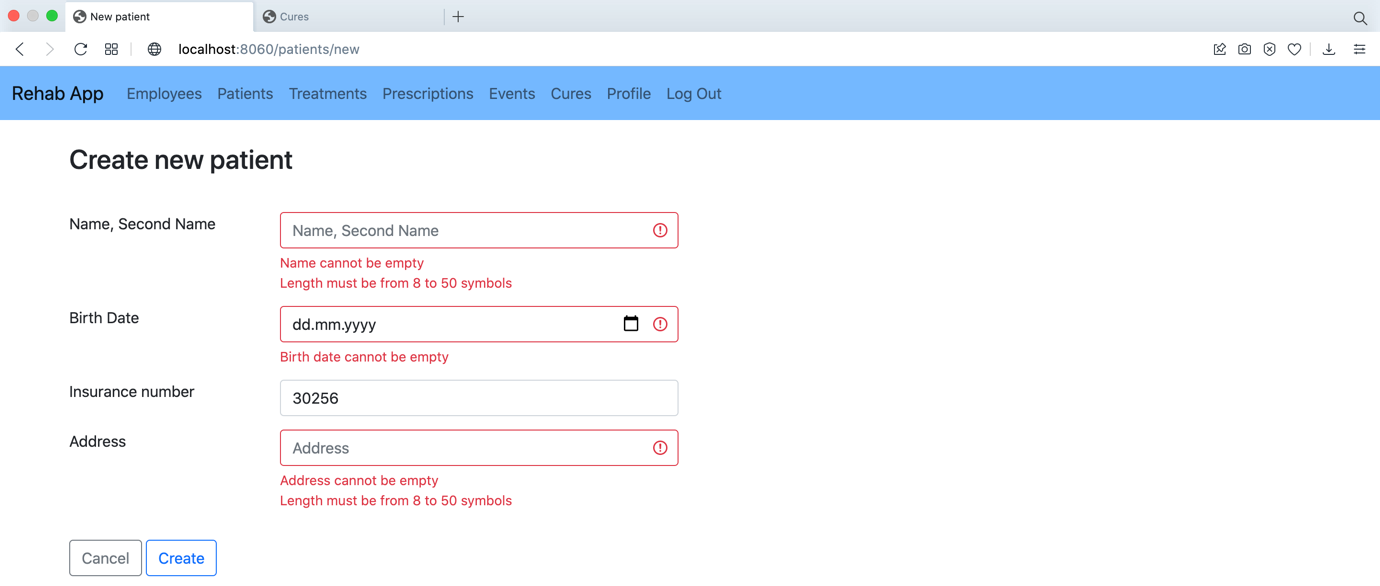
Here are filters for following lists:

* patients: insurance number, name…, only treating (or any).
* treatments: patient insurance number, date, only open (or any), only for current doctor (or any).
* prescriptions: patient insurance number, date, only active (or any), only for current doctor (or any).
* events: patient insurance number, planned date, only planned (or any), only for current nurse (or any).
* cures: name…

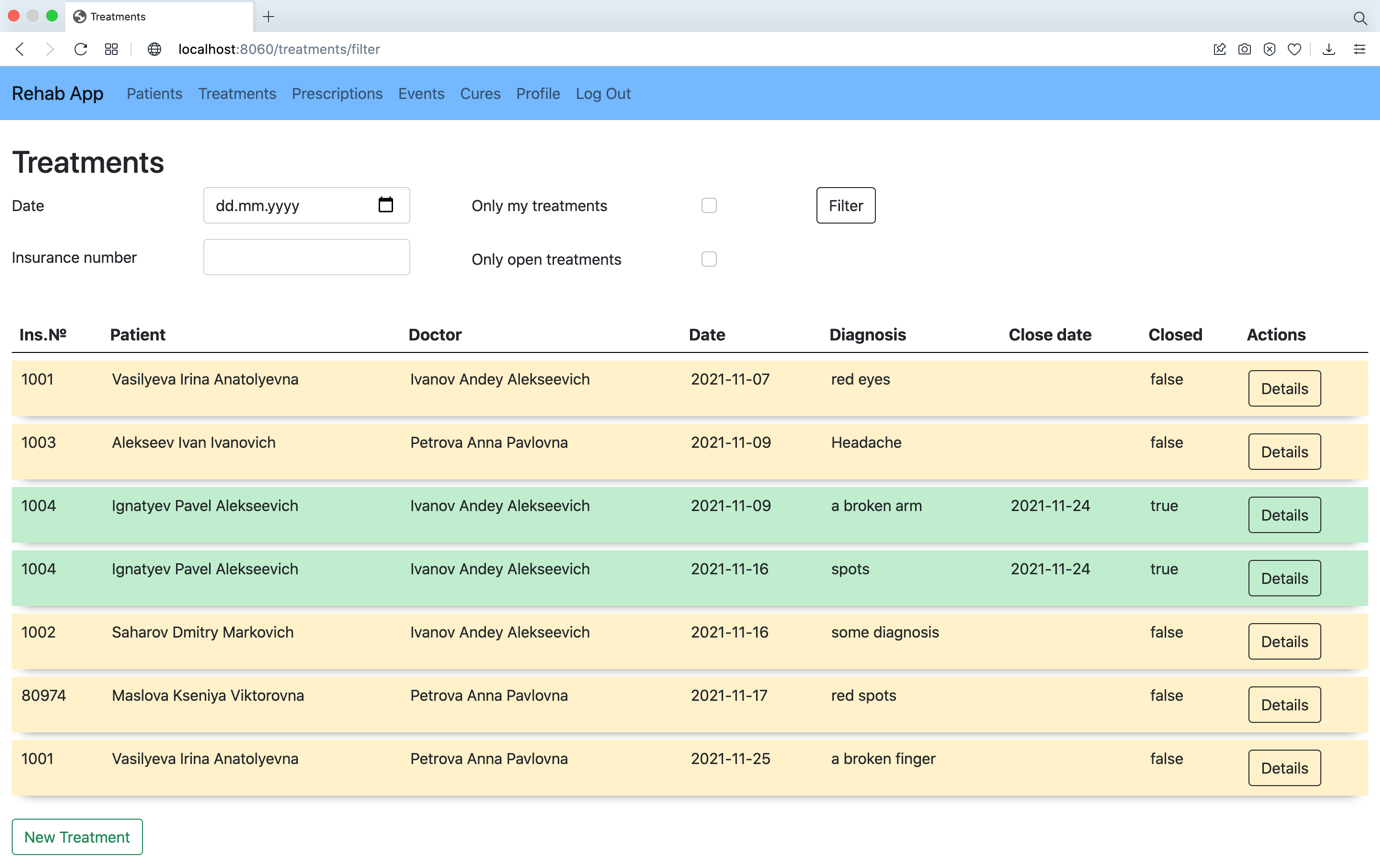
*Page: List of patients*



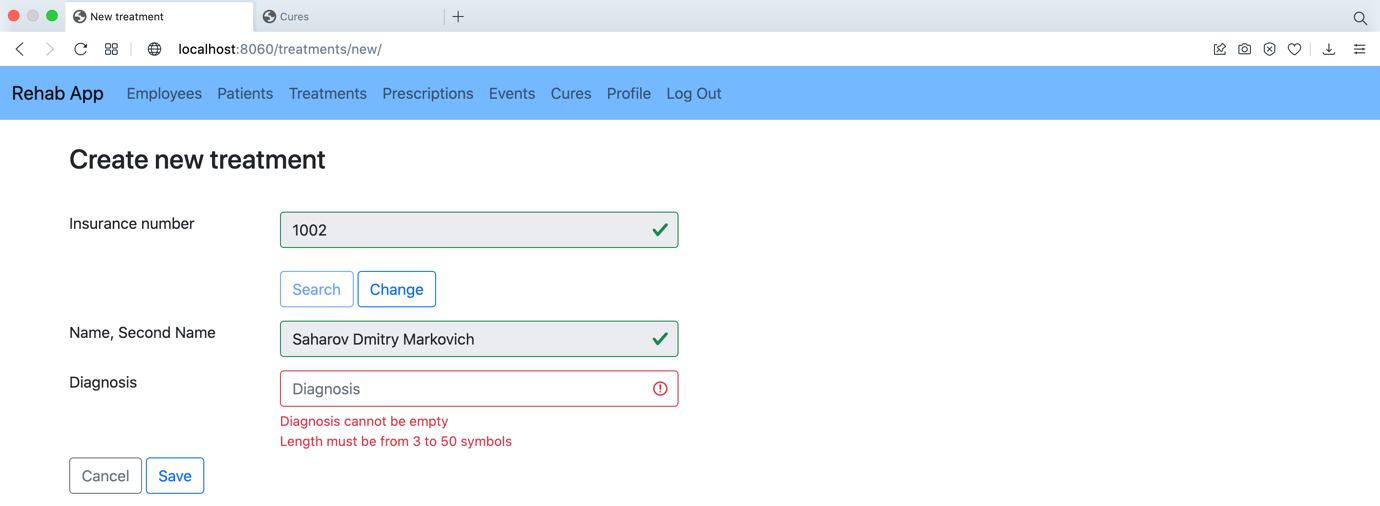
*Page: Create new patient (with validation)*



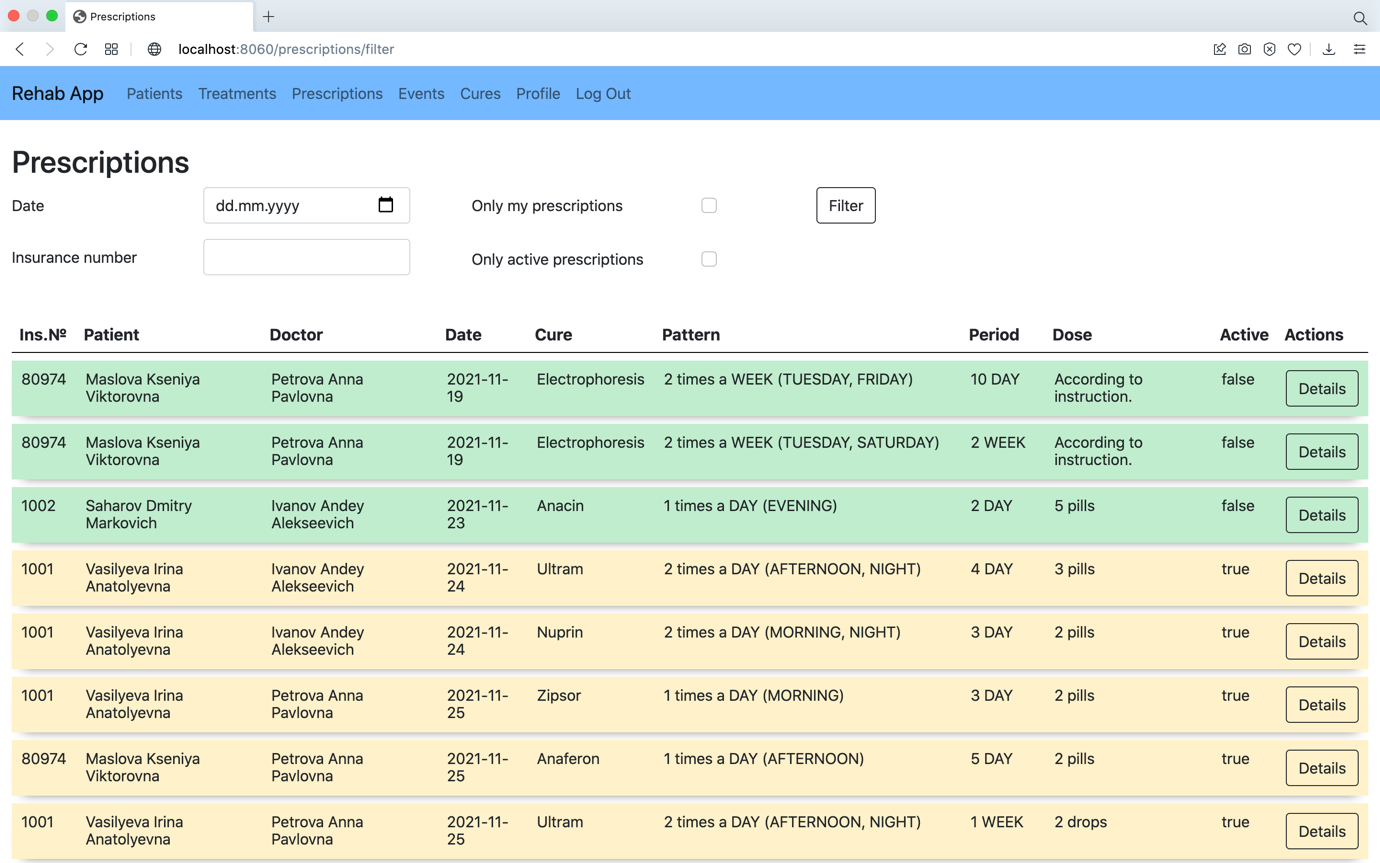
*Page: List of treatments*



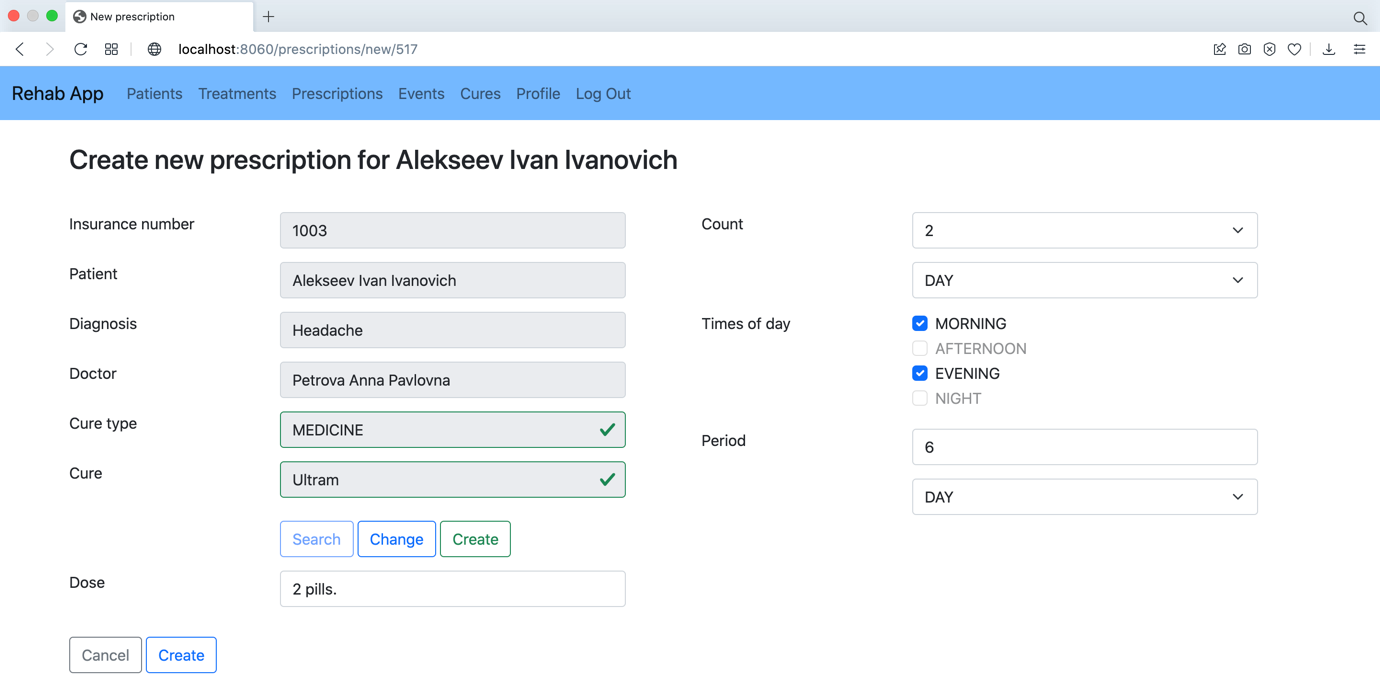
*Page: Create new treatment (with validation)*



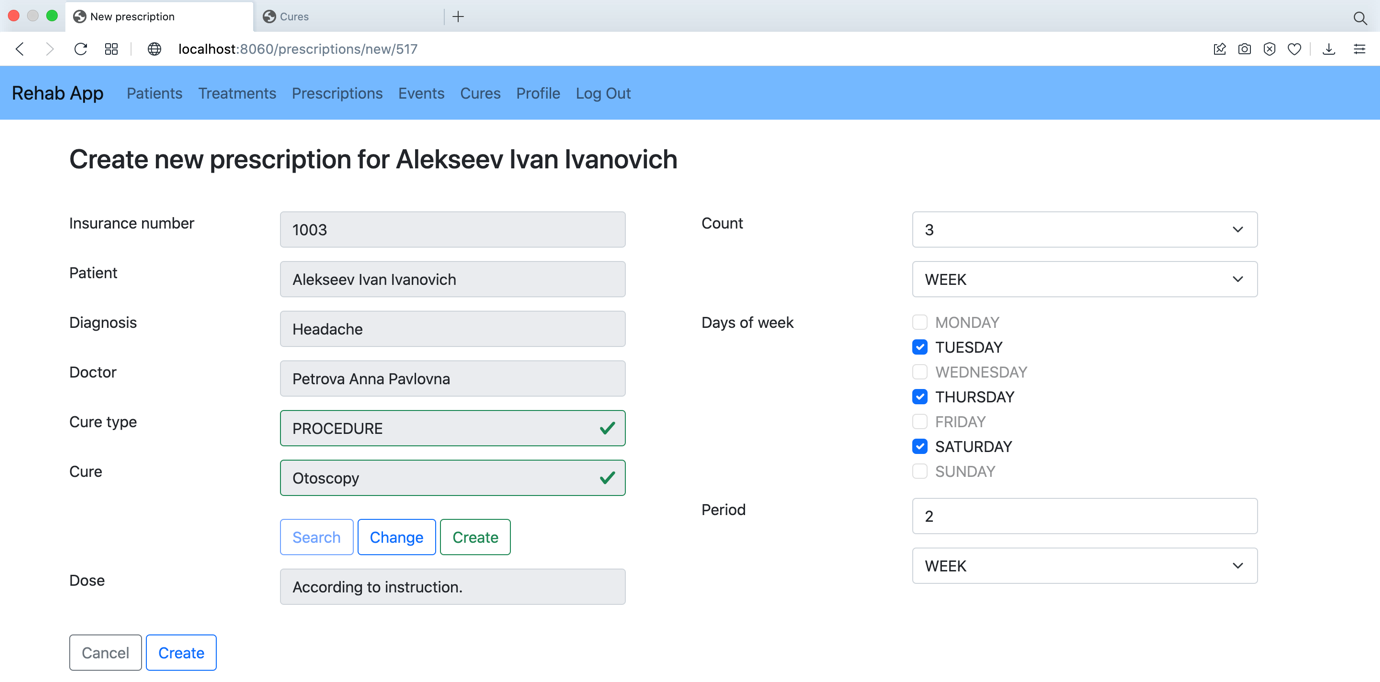
*Page: List of prescriptions*



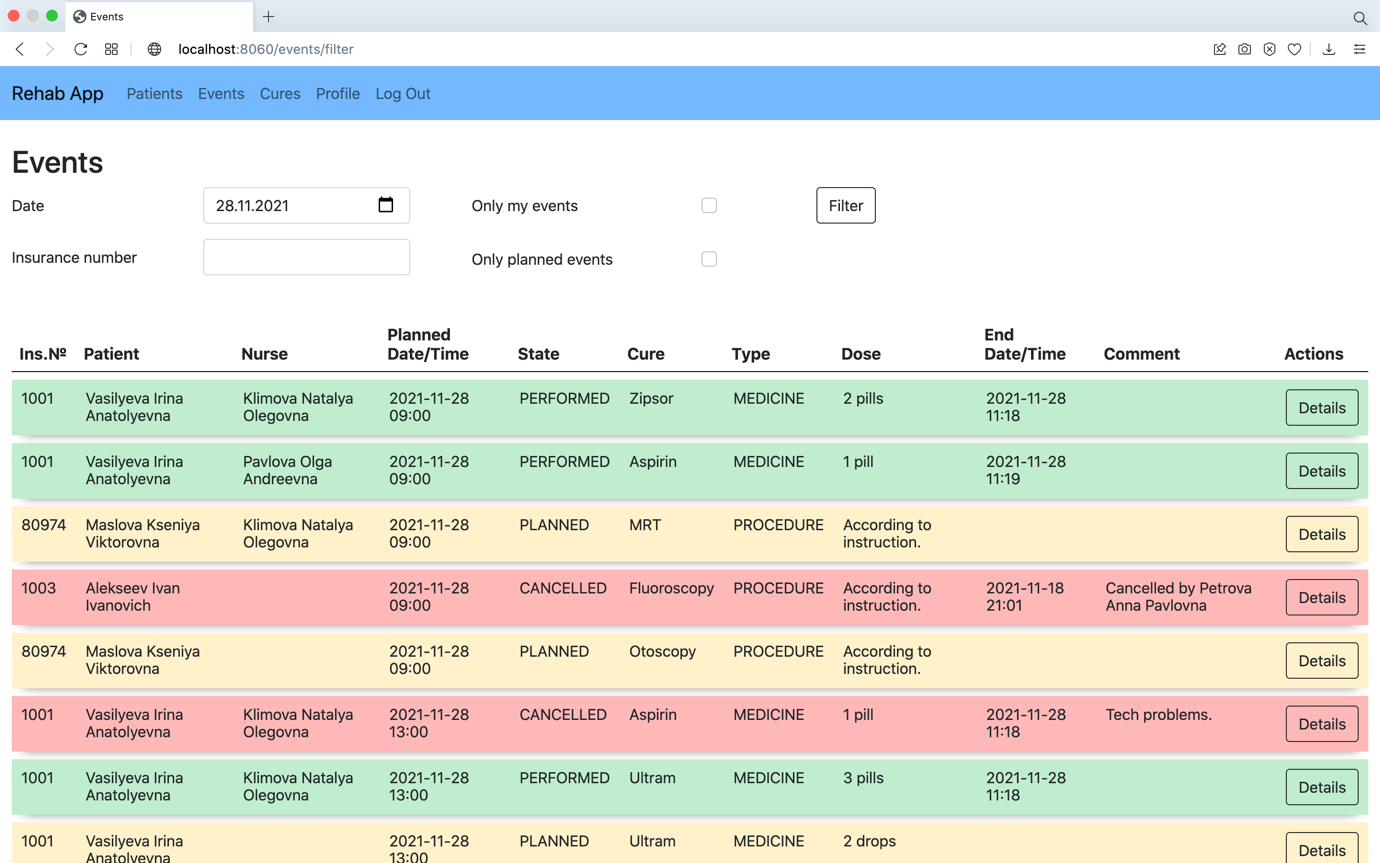
*Page: create new prescription (example for medicine)*



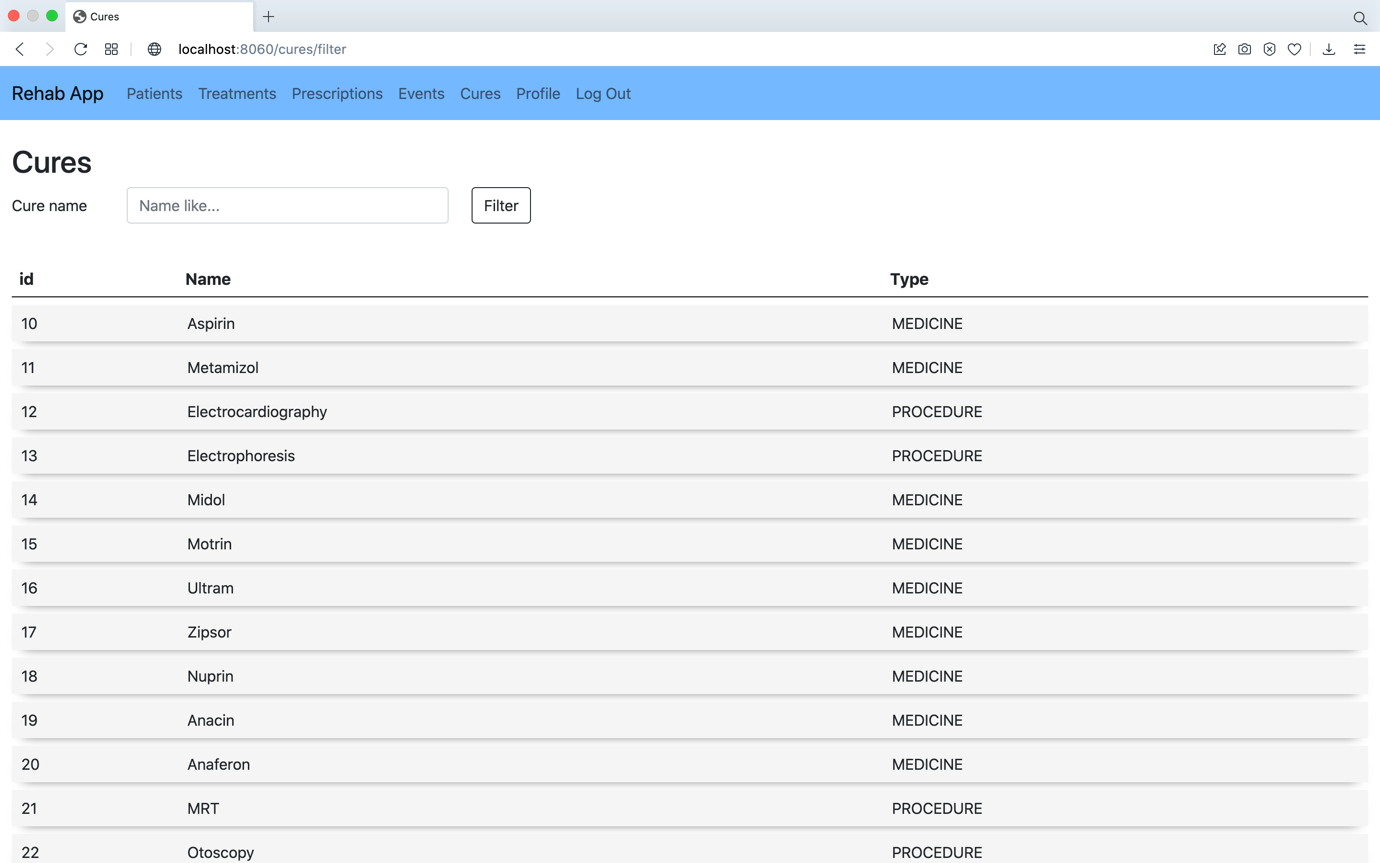
*Page: create new prescription (example for procedure)*



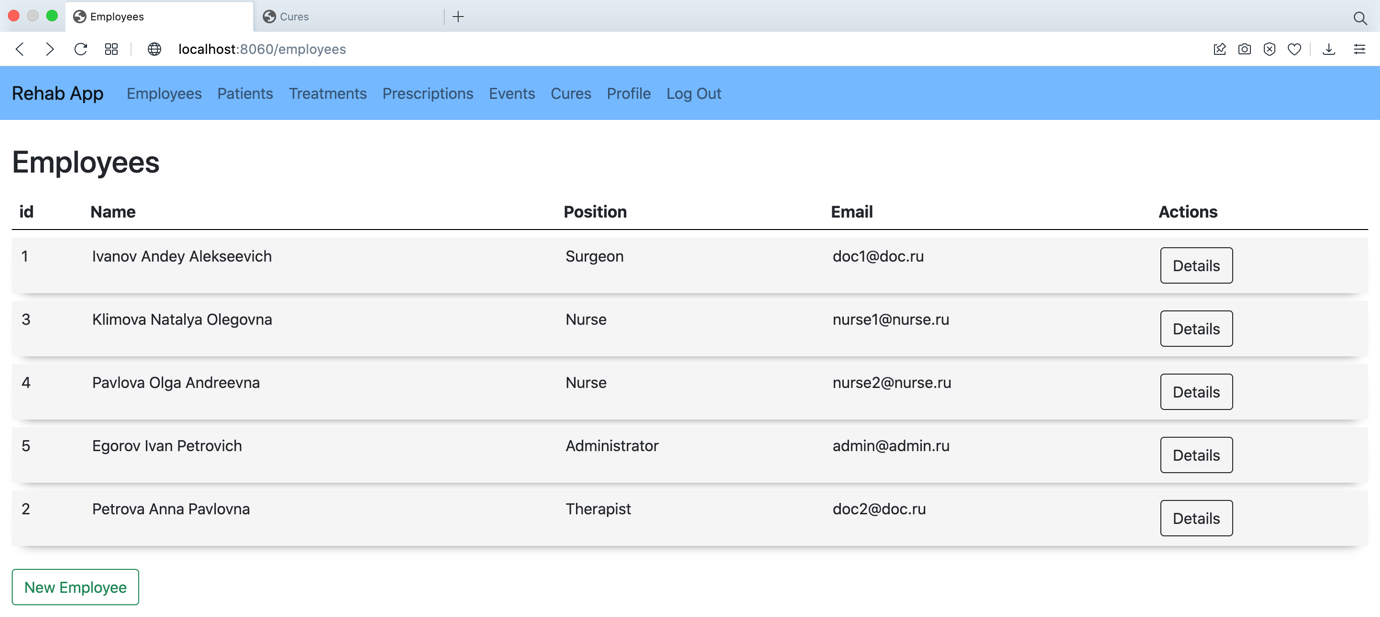
*Page: List of events*



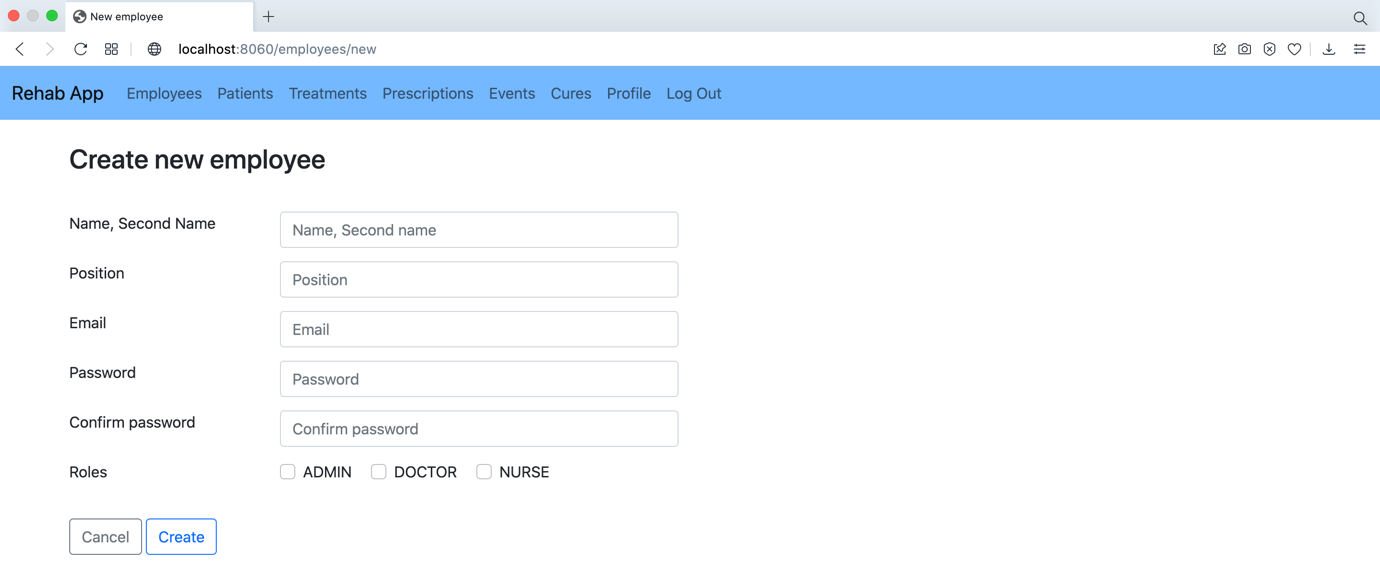
*Page: List of cures*



*Page: List of employees*



*Page: Create new employee*



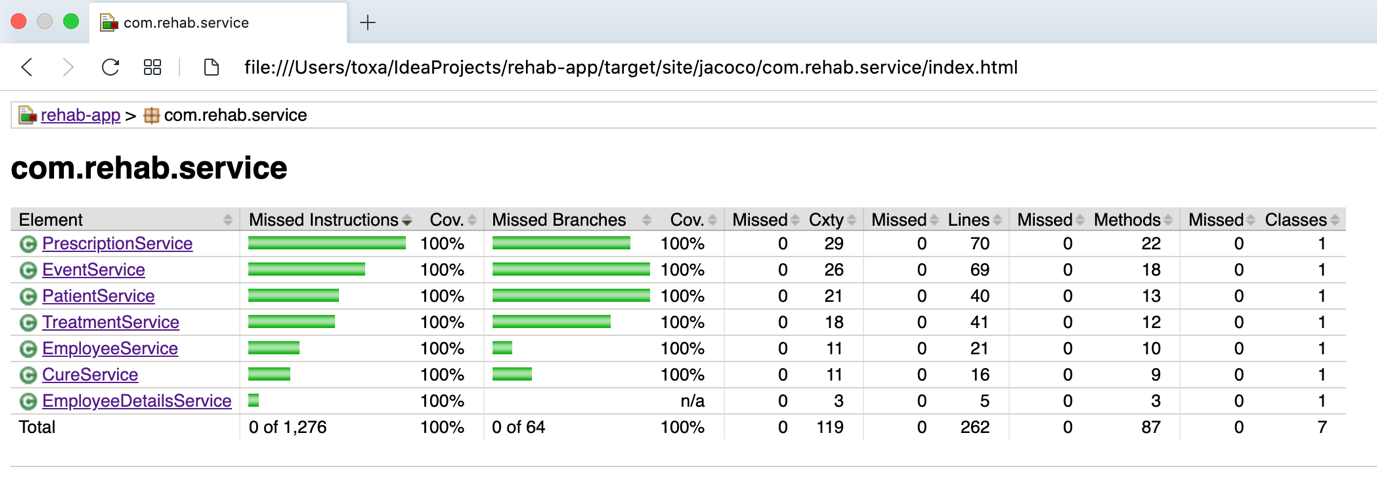
*Page: Display for events (the second application)*



# JUnit tests

We only test business logic of the application. We use H2 inmemory database in test configuration. The list of tests is below. Unit tests cover 100% code of services.

*Code coverage for services*



There are many similar tests for all services that perform the following:

getById – found object (by given id) and expected one are equal

getByIdNotFound – object (by given id) does not exist

getBy\*GivenField\* – found object (by given field) and expected one are equal

getBy\*GivenField\*NotFound – object (by given field) does not exist

save – saved object and expected one are equal

saveWithExisting\*Field\* – it is impossible to save object, if another one has the same value of given field

getAll – found list of all objects and expected list are equal

getWithFilter – found list of objects (by given filters) and expected list are equal

Test list for Employee Service:

changePassword – changed password and expected password are equal

getAuth – authenticated employee and expected one are equal

Test list for Patient Service:

update – updated patient and expected one are equal

updateWithExistingInsNum – it is impossible to update patient, if another one has the same insurance number

discharge – discharged patient and expected one are equal

Test list for Treatment Service:

close – closed treatment and expected one are the same

closeWhenClosed – it is impossible to close treatment if it is already closed

closeWhenHasActivePrescription – it is impossible to close treatment if it has active prescriptions

closeWithDifferentDoctor – it is impossible to close treatment if it was created by different doctor

Test list for Prescription Service:

cancel – cancelled prescription and expected one are the same

cancelWhenAlreadyCancelled – it is impossible to cancel prescription if it is already cancelled

cancelWithDifferentDoctor – it is impossible to cancel prescription if it was created by different doctor

update – updated prescription and expected one are the same

Test list for Event Service:

setNurse – chosen event and expected are equal

setNurseWhenAlreadySet – it is impossible to choose event if it is already chosen

setNurseWhenHasNotPlannedState – it is impossible to choose event if it does not have «planned» state

unSetNurse – discarded event and expected are equal

unSetNurseWhenHasNotNurse – it is impossible to discard event if it is not chosen

unSetNurseWhenHasDifferentNurse – it is impossible to discard event if it was chosen by different nurse

unSetNurseWhenHasNotPlannedState – it is impossible to discard event if it is does not have «planned» state

perform – performed event and expected one are equal

performWhenHasNotNurse – it is impossible to perform event if it was not chosen

performWhenHasDifferentNurse – it is impossible to perform event if it was chosen by different nurse

performWhenHasNotPlannedState – it is impossible to perform event if it is does not have «planned» state

perform – cancelled event and expected one are equal

# Possible improvements

There are some ideas and improvements that can be implemented in the next versions of the application.

1. In this version Admin can change only password for existing employee. We want to add ability to edit any field for them (name, position etc.).
2. In this version there is no filter for list of employees. We want to add filter for employees by name…, position.
3. In this version prescription can be active or not. Doctor can either cancel or update the prescription (in both cases, current prescription becomes inactive). So, even if all events for the prescription were performed, doctor has to cancel the prescription before closing treatment. It is a bit confusing. We want to add ability to close the prescription (if all events were performed).
4. In this version doctor can operates only with his/her own treatments and prescriptions, which is right. We want to add role «HEAD\_DOCTOR», that will be able to operate with any treatments and prescriptions.
5. In this version when doctor is searching cure while creating prescription, doctor has to type fully correct cure name to find it (or watch the correct name in list of cures, where filter is). In some cases, it is not convenient (e.g., cure name is too long). We want to add ability to search cure by several letters in order to get list of suitable cures and choose the right one.