|  |
| --- |
|  |
| Technical solution description  Enterprise content management system of medical rehabilitation center  ECM  by Shevelev Yuri |
|  |

Content

[**1 Overview** 3](#_Toc19136026)

[**2 Used technologies and frameworks** 4](#_Toc19136027)

[**3 Additional features** 5](#_Toc19136028)

[**4 Database scheme** 6](#_Toc19136029)

[**5 Explanation of model implementation** 8](#_Toc19136030)

[**6 Application modules** 9](#_Toc19136031)

[**7 UI layer description** 10](#_Toc19136032)

[**8 Business layer description** 11](#_Toc19136033)

[**9 Data layer description** 12](#_Toc19136034)

[**10 Application screenshots** 13](#_Toc19136035)

[**11 Unit tests** 14](#_Toc19136036)

[**12 Deployment** 15](#_Toc19136037)

[**13 Logging configuration** 15](#_Toc19136038)

[**14 Application improvements** 15](#_Toc19136039)

1. **Overview**

The enterprise content management system of the medical rehabilitation center «ECM» is a multi-user client-server application. Its main purpose is to automate the workflow of medical rehabilitation center. The application provides different options depending on the user's role: doctor or nurse. Doctors have options focused on patients, while the functionality for nurses is to view and change the status of treatment events. The application also has a web service that provides information about the events, scheduled for today.

1. **Used technologies and frameworks**

* JDK 1.8
* Apache Tomcat
* Spring framework
* Spring security
* JPA, Hibernate framework
* PostgreSQL
* Jackson
* Maven
* Log4j
* Slf4j
* JUnit
* Mockito
* Bean validation API
* Hiberate Validator
* Spring security test
* H2 embedded database
* JSP, JSTL
* jQuery
* Bootstrap
* EJB
* JSF
* JAX-RS
* Primefaces
* iText
* AS Wildfly
* Flyway DB migration
* Hikari connection pool
* Active MQ
* Lombok
* Sonar scanner
* Jacoco

1. **Additional features**

In addition to the required features, the following features has been implemented:

* Internationalization (English and Russian languages);
* 100% test coverage;
* database migration;
* full javadoc documentation;
* application builds on CodeShip CI/CL server;
* ---------
* selenium browser auto-tests;
* PDF creation of today events;

1. **Database scheme**

The application uses the following entities:

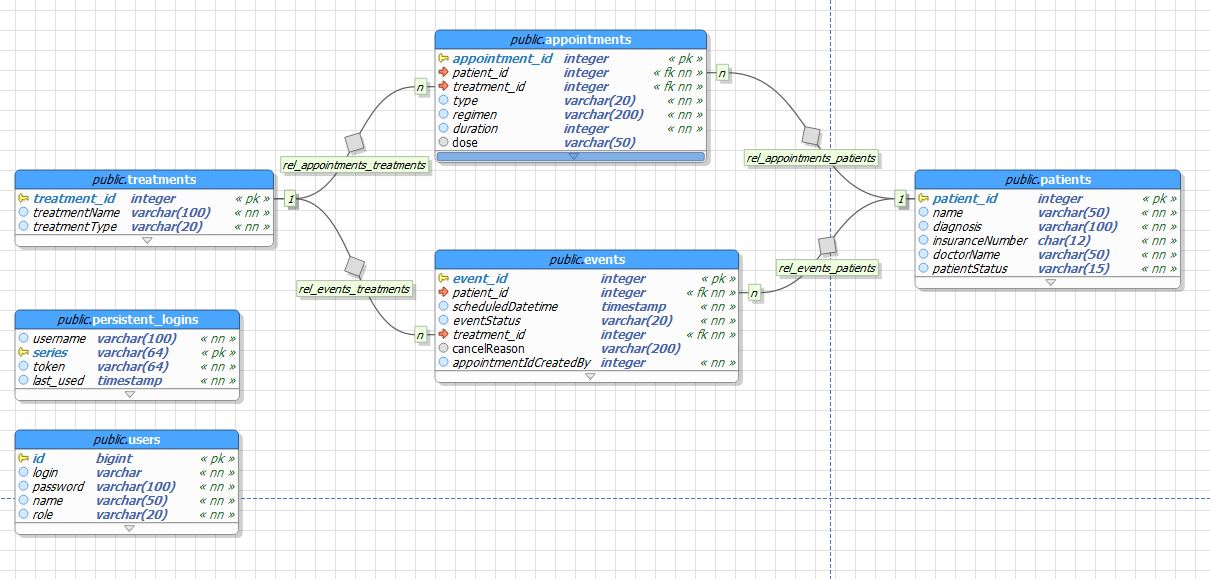
1. User – represents users of the application.
2. Patient – represents the patients.
3. Appointment – represents the appointments for patients.
4. Treatment – represents the treatments (procedures or medications).
5. Event - represents the scheduled events.
6. Persistent\_logins – Spring Security table for «remember-me» feature.

Appointments table has many-to-one relationship with Patients table and one-to-one with Treatments table.

Events table has many-to-one relationship with Patients table and one-to-one with Treatments table.

Treatments table, Users table and Persistent\_logins table has no relationships.

Entity relationship model is shown on the next page in Figure 1.

Fig. 1 – Entity relation model.

1. **Explanation of model implementation**

The application is intended to automate the workflow of medical rehabilitation center, and covers following business processes, divided by the user role.

For doctors:

1. adding new patients to the center;
2. discharging patients from the center;
3. assign treatments to patients (procedures and medications);
4. editing assigned treatments;
5. cancelling assigned treatments.

For nurses:

1. viewing all scheduled events;
2. filtering scheduled events (for today, for next hour) and by patients;
3. changing events’ statuses from «Scheduled» to «Accomplished» and from «Scheduled» to «Cancelled».

When changing event’s status to «Cancelled» the nurse must specify the reason for cancellation.

1. **Application modules**

Application is built on 3-layered architecture.

All classes are running inside Dependency Injection container, provided by Spring Framework.

The data access layer or DAO layer – provides data accessing with JPA implementation – Hibernate. Application has entity objects that are persisting to PostgreSQL database.

The service layer – all business-logic is implemented here. Services are using DAO classes’ injections to get persistent data. All service methods are running in transactions, so all operations with data satisfies the «ACID» properties.

The representation or view layer – representation of information and handling of requests managed by Controller classes. Services are injected to controllers. View layer is built on Spring MVC technology. Methods of controllers handle requests and call service methods to process. Views are JSP pages which uses Bootstrap and jQuery libraries.

In addition, there is the module that connects with application by REST service, its purpose is to show events, scheduled for today.

1. **UI layer description**

UI layer consists of 6 controllers. Controllers are built, using Spring MVC technology.

Templates is a JSP with JSTL and Spring tags. Each page template use layouts. Layout is a JSP fragment that contains all repeating content of a page (i.e. header and footer). There are three layouts in the application:

\_head.jsp – basic layout, contains meta-data, styles, icons and scripts links.

\_header.jsp – contains header navigation bar with localization and logout buttons.

\_footer.jsp – contains footer.

All text and messages used in the UI layer are provided by messages.properties files that enable localization. Currently the application supports English and Russian languages.

Application uses Bootstrap and jQuery libraries.

REST webservice represented by EventRestController. Service forms list of events, scheduled for today and sends it in JSON format.

1. **Business layer description**

Business layer has 6 service interfaces, 6 corresponding implementations and one custom implementation of Spring Security interface – UserDetailsService.

List of services with description is listed below:

1. AppointmentService – provides logic for operations with appointments.
2. EventService – provides logic for operations with events, such as adding, getting events, scheduled for today, change status etc.
3. PatientService – provides logic for operations with patients.
4. RegimenProcessorService – provides logic for regimen operations and events generation.
5. TreatmentService – provides logic for operations with treatments.
6. UserService – provides logic for operations with users.
7. UserDetailsServiceImpl – custom implementation of Spring Security interface – UserDetailsService, provides logic for user-specific data.

All methods in service layer are transactional. Spring Framework provides transactions management.

1. **Data layer description**

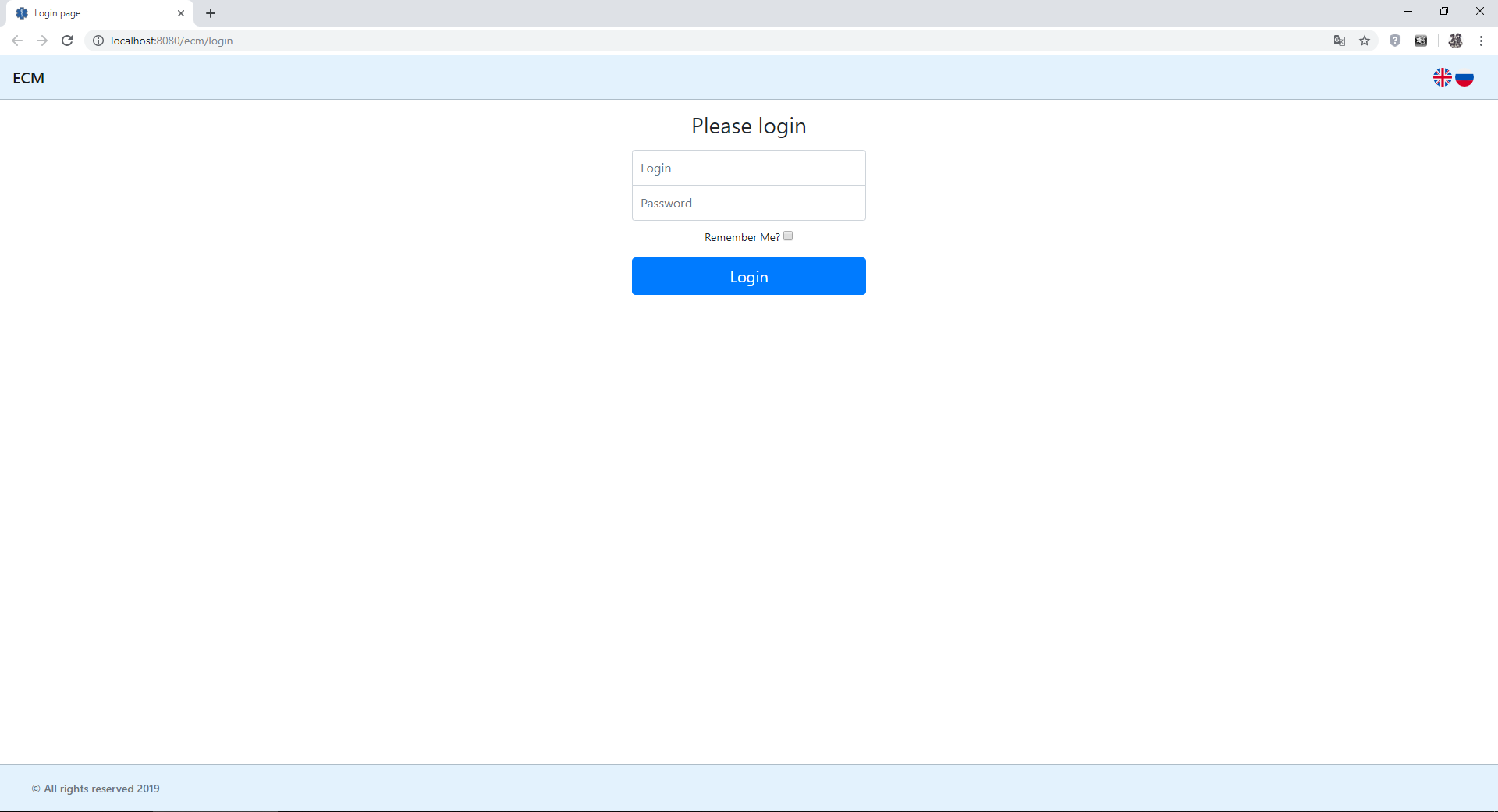
Dao layer contains 5 JPA entities, 5 DAO classes and 6 DTO classes (1 for REST service).

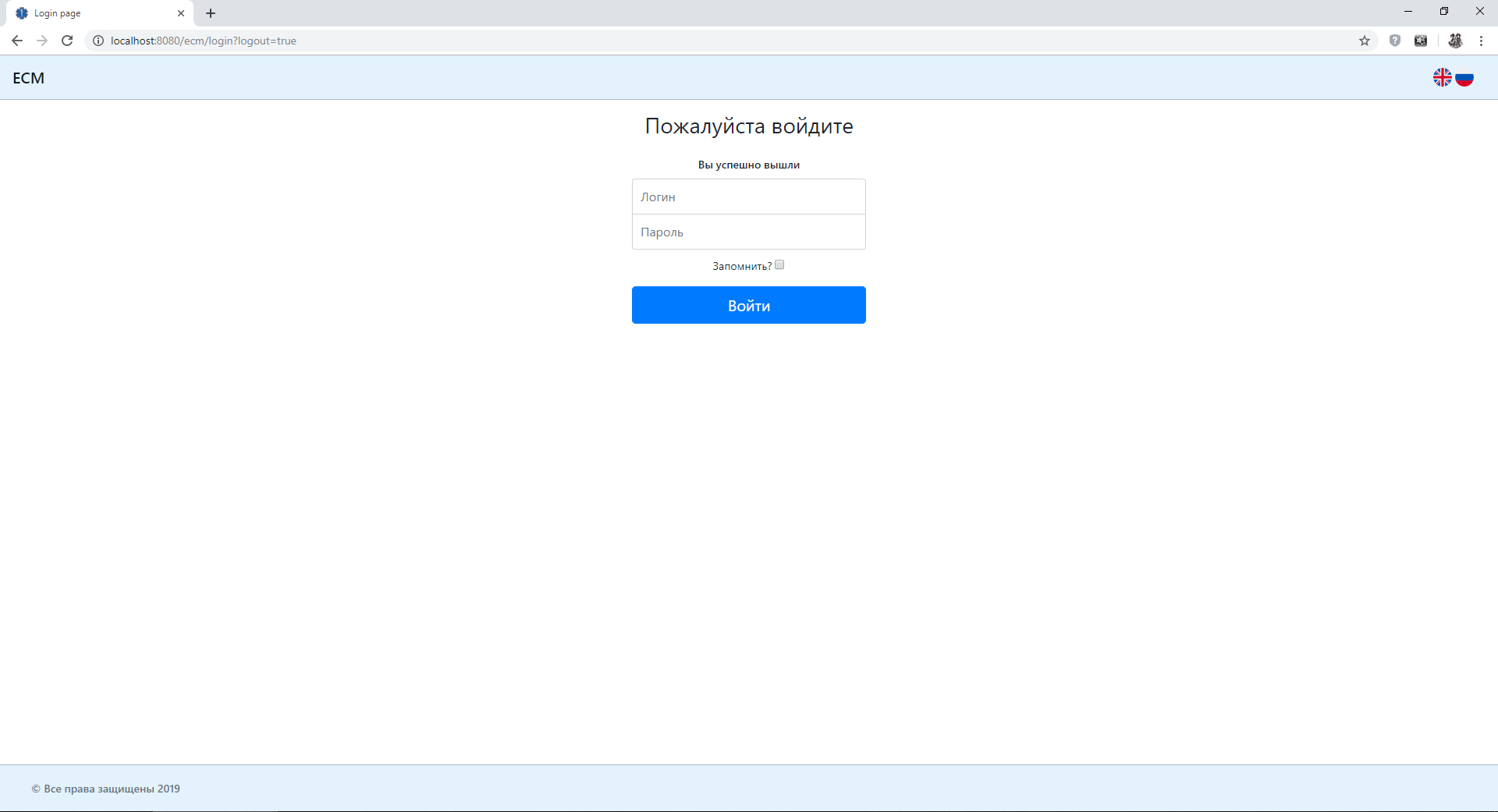
There is AbstractDAO class that provides common operations. All other DAO’s extends this class and add their methods, specific for the class.

Each DAO class manages one JPA entity.

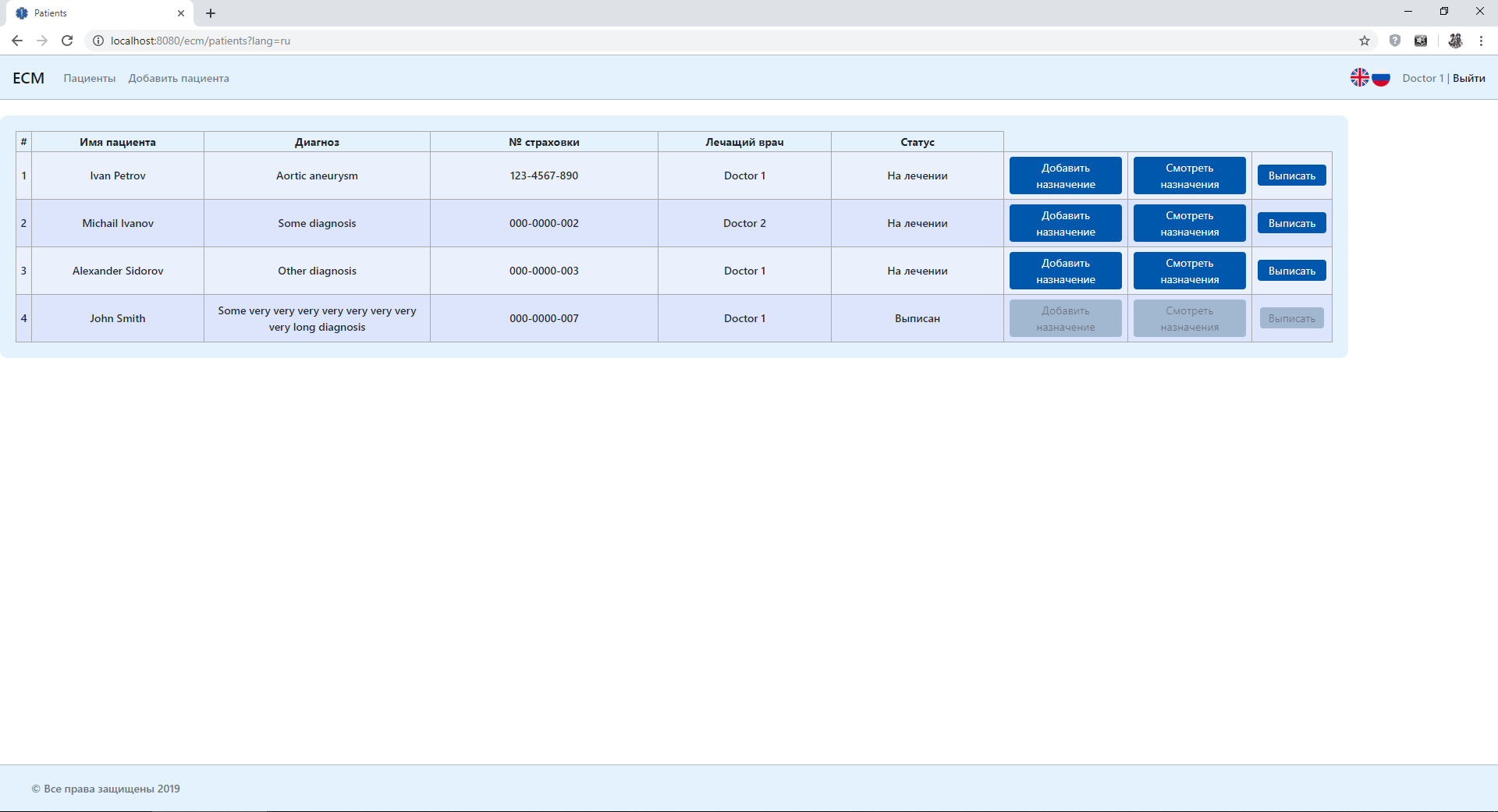
1. **Application screenshots**

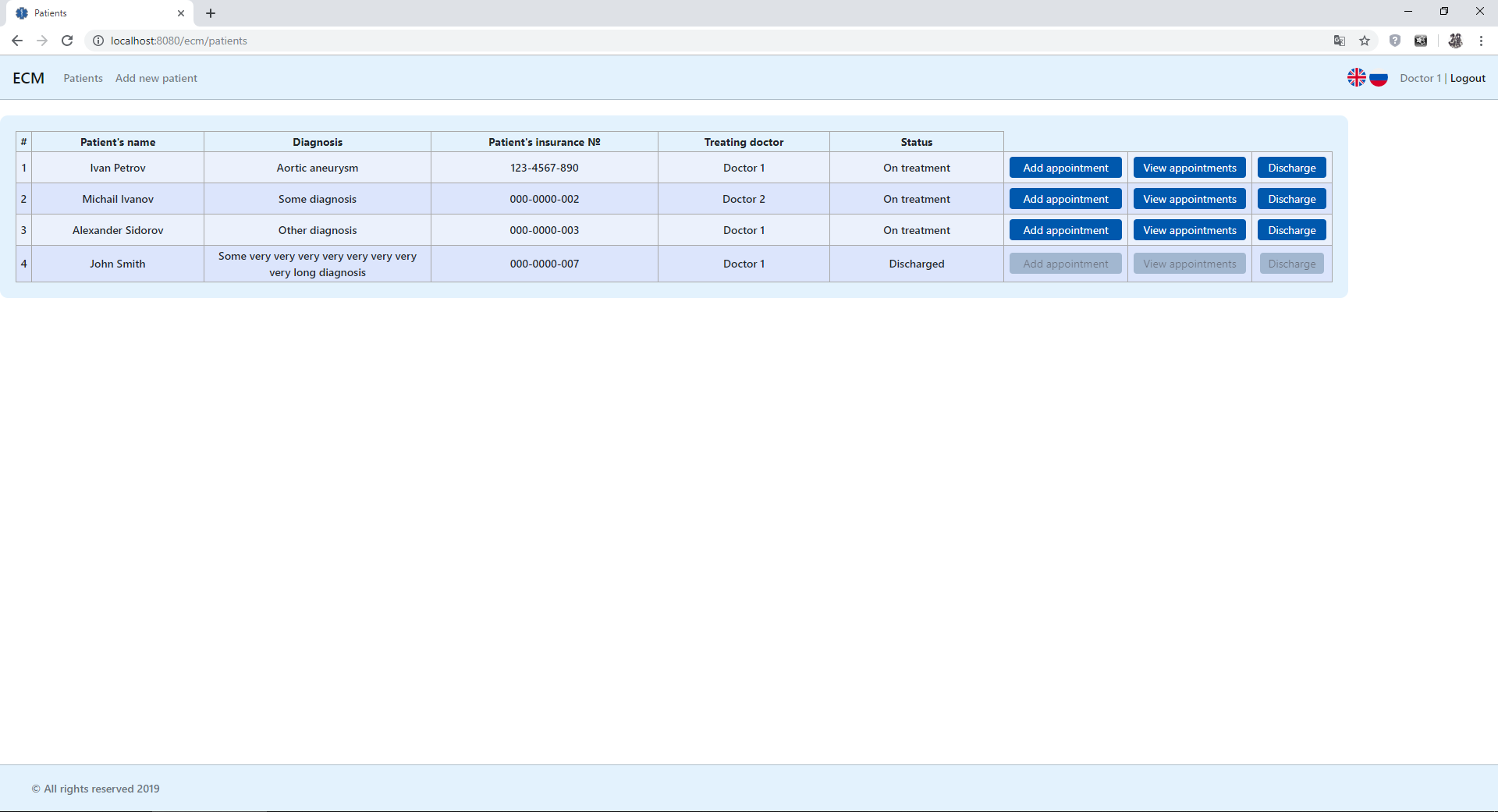
Main application

Login page (en)

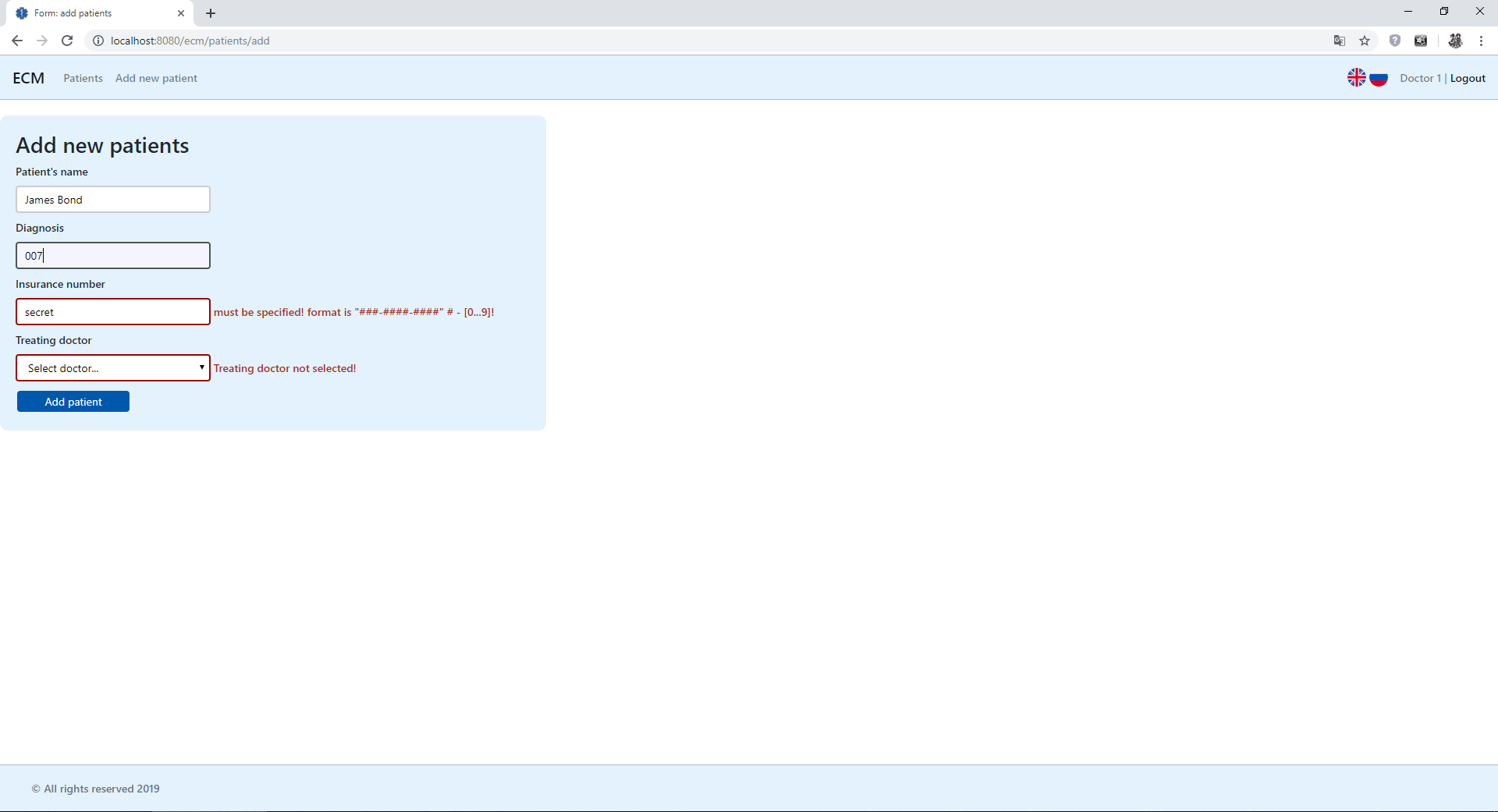


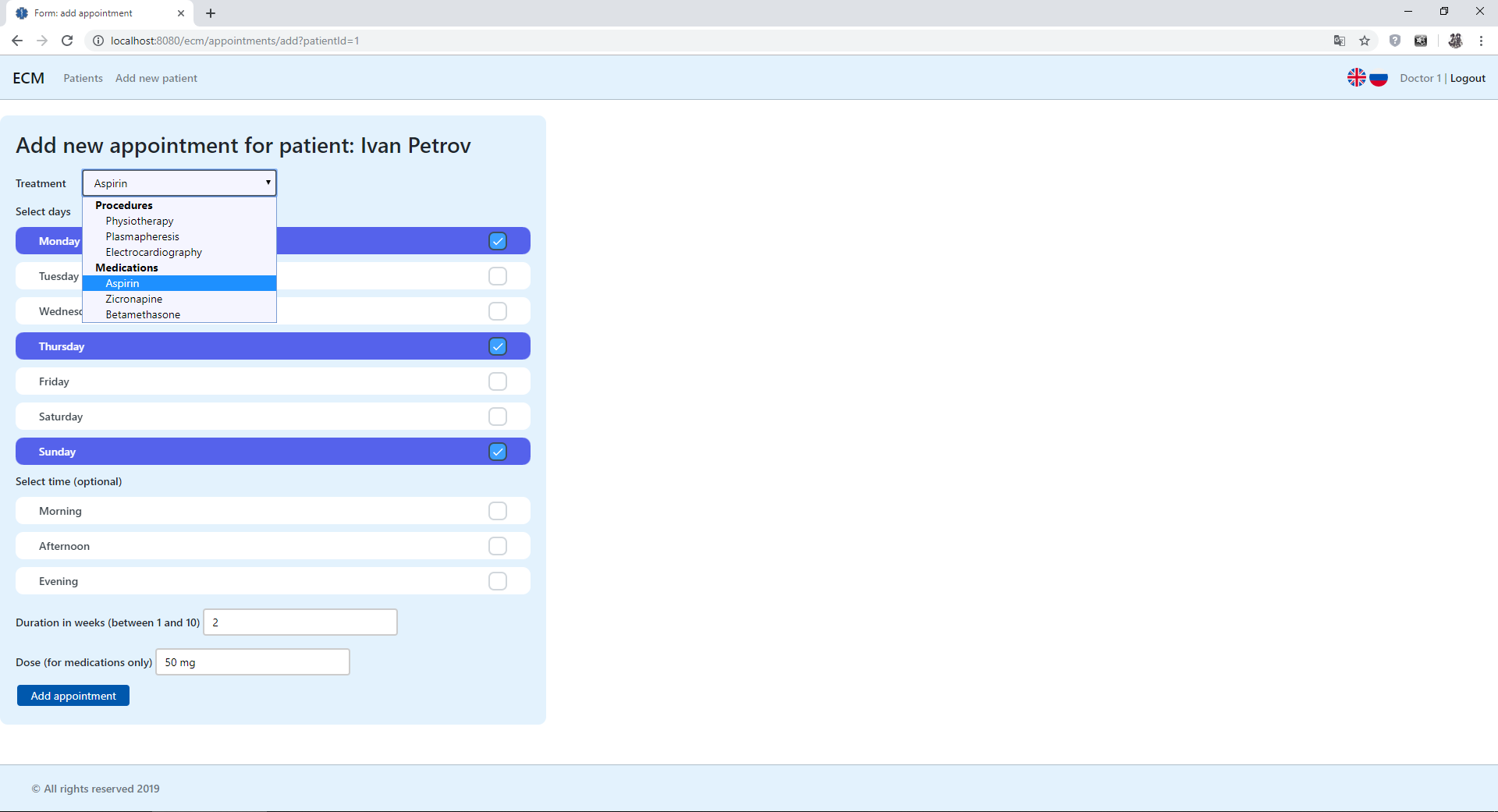
Login page after logout (ru)

Patients page (ru)

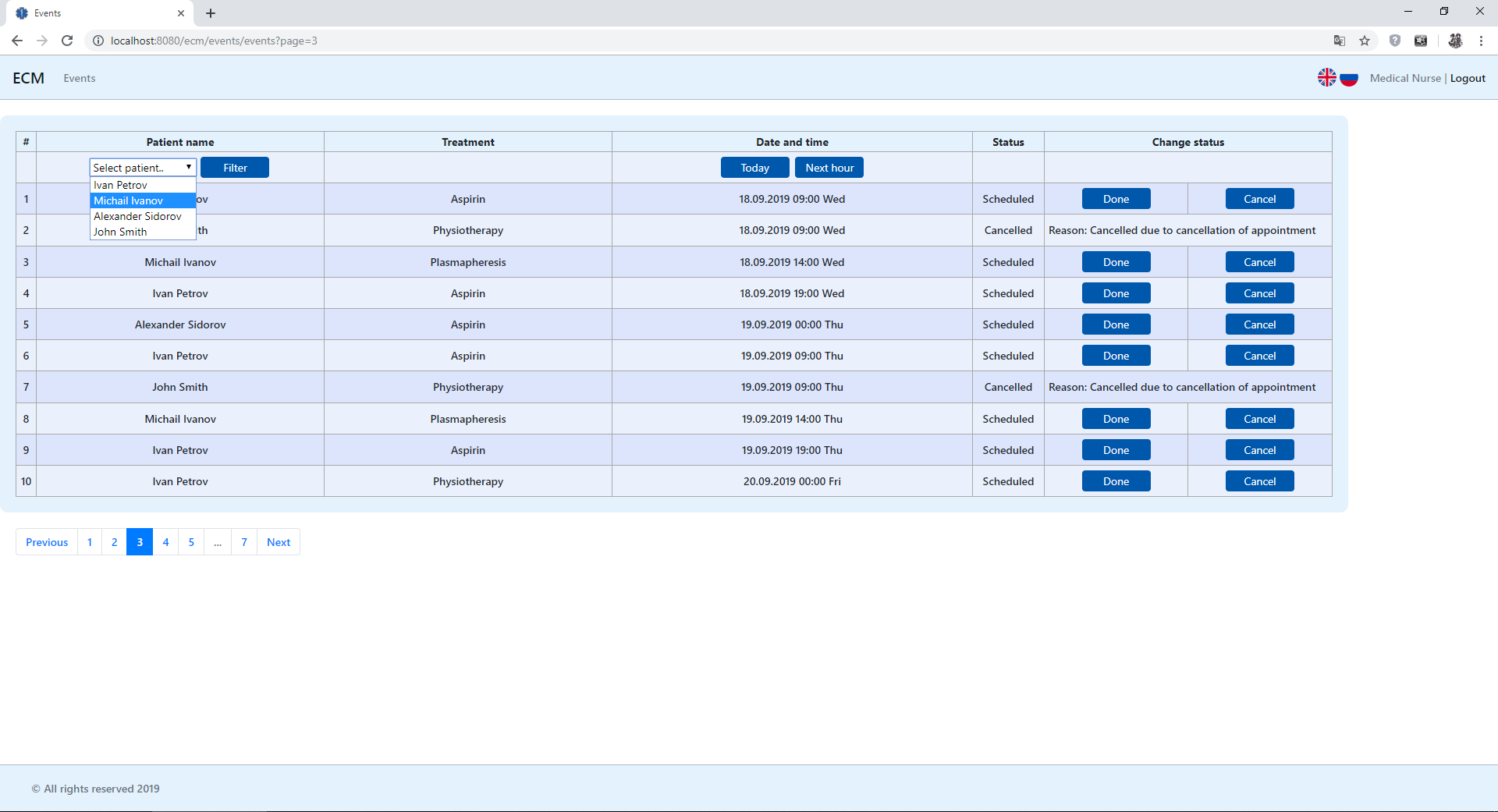


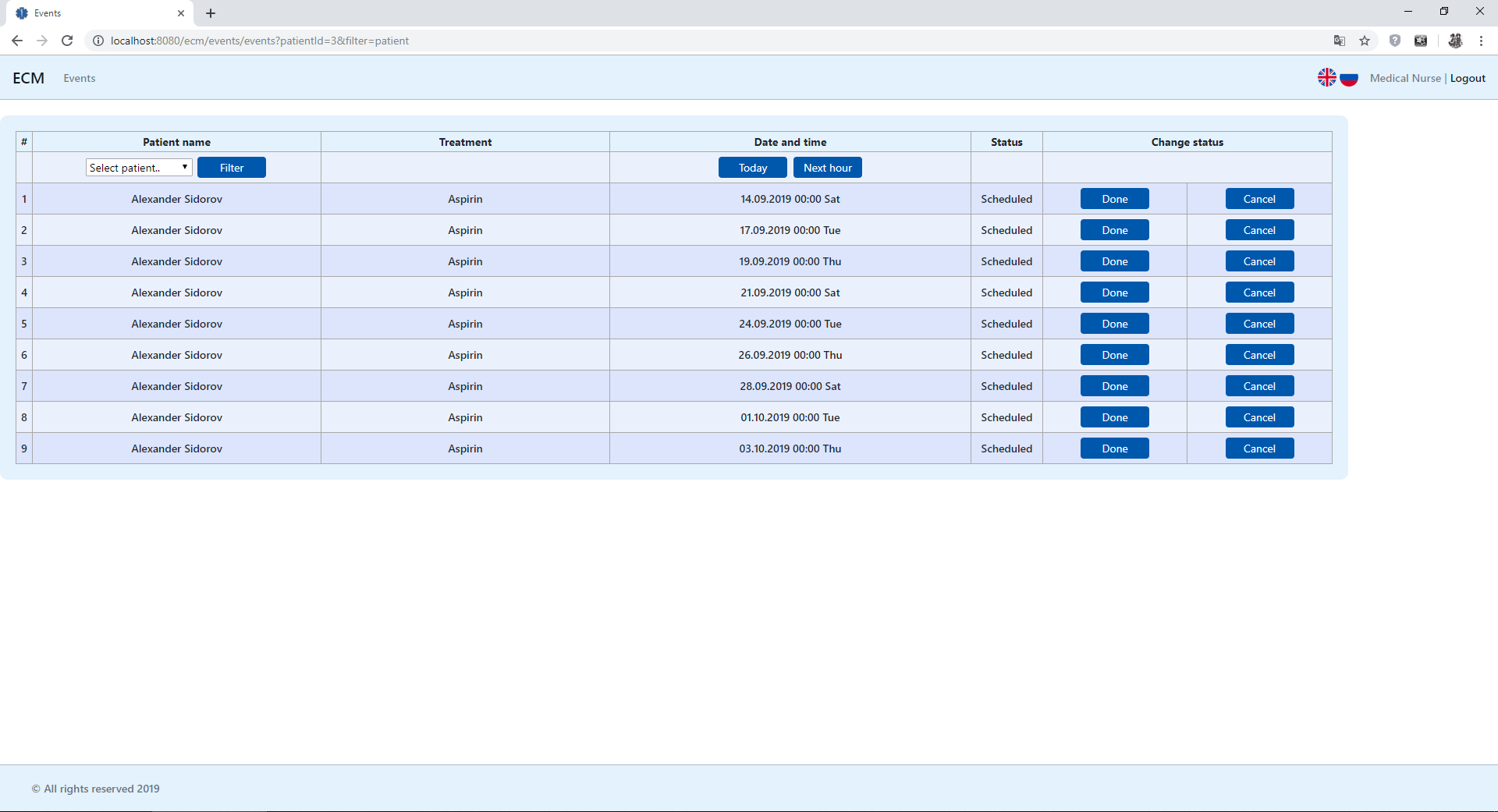
Patients page (en)

Add new patient page with errors (en)

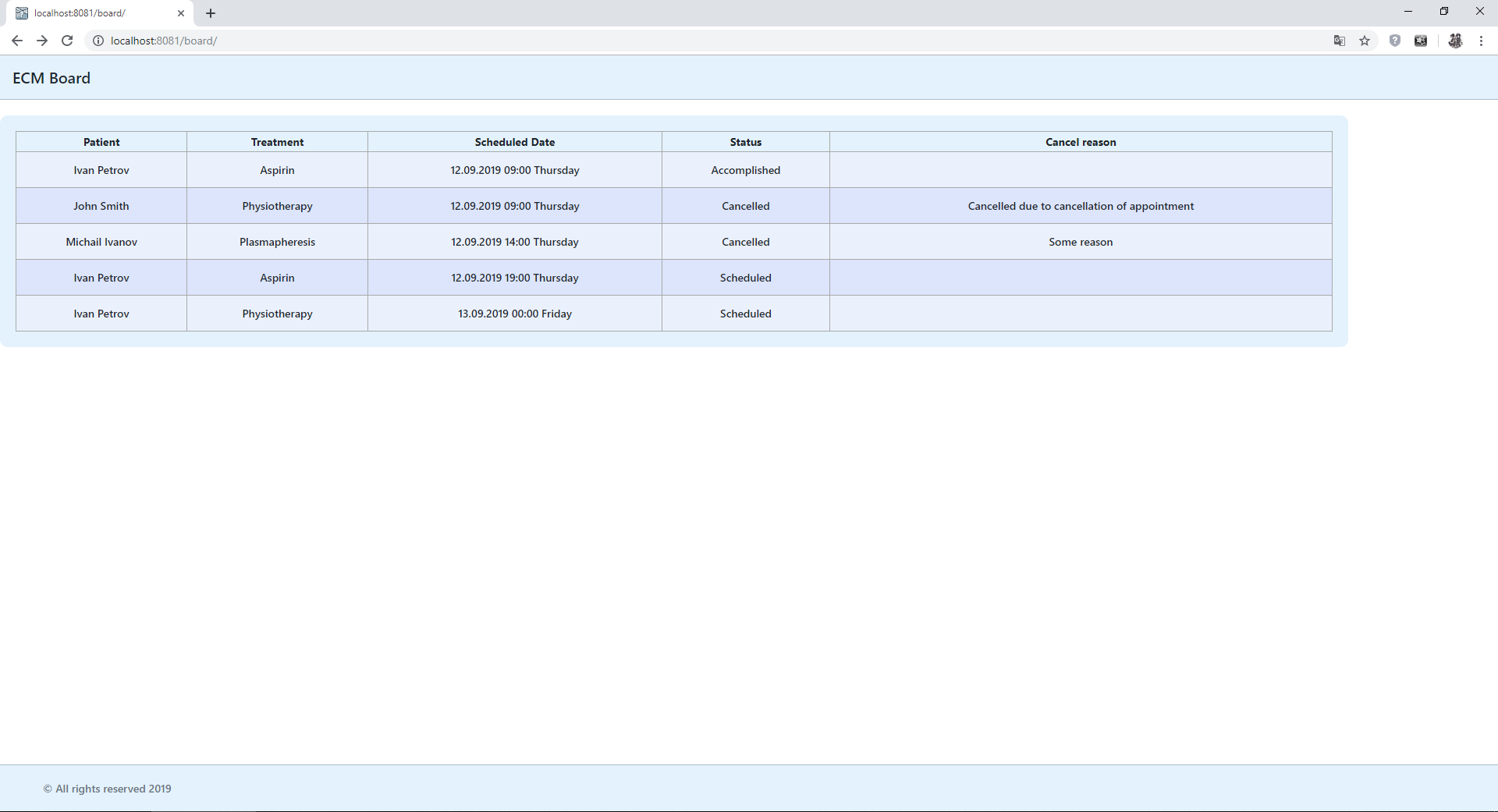


Add new appointment page (en)

All events page (en)

Events page with filter by the patient (en)

Webservice application

Today events page (en)

1. **Unit tests**

Unit tests are written on all classes of service layer, each test class tests one service class.

All methods have at least one positive test. Some methods have negative scenarios too.

List of Junit test classes for Service layer:

1. TreatmentServiceImplTest;
2. PatientServiceImplTest;
3. UserServiceImplTest;

List of Junit test classes for the controllers:

1. IndexControllerTest;
2. **Deployment**

Applications uses IDE building process. IDE automatically builds, starts servers and deploys applications.

To deploy the applications press the «Run configuration» (green arrow) button.

This configuration skips unit tests. To run the tests use Maven command: mvn test.

1. **Logging configuration**

Logging is provided by Log4j2 library with Slf4j simple logging façade library.

Both applications have different filters for log files. There are two types of logging levels: all level log that writes to the console and info and above level file log that writes into a file with RollingFileAppender.

All service methods calls are logged at trace level, all exceptional situations logged at error level. Some methods have debug level logs – for development process and debugging.

Below is a short example of a log output when a doctor user opens up appointment edit form:

12-09-2019 08:48:12 DEBUG RequestParamsInterceptor:27 - Request parameters:

12-09-2019 08:48:12 DEBUG RequestParamsInterceptor:29 - patientId=[2]

12-09-2019 08:48:12 DEBUG RequestParamsInterceptor:29 - appointmentId=[2]

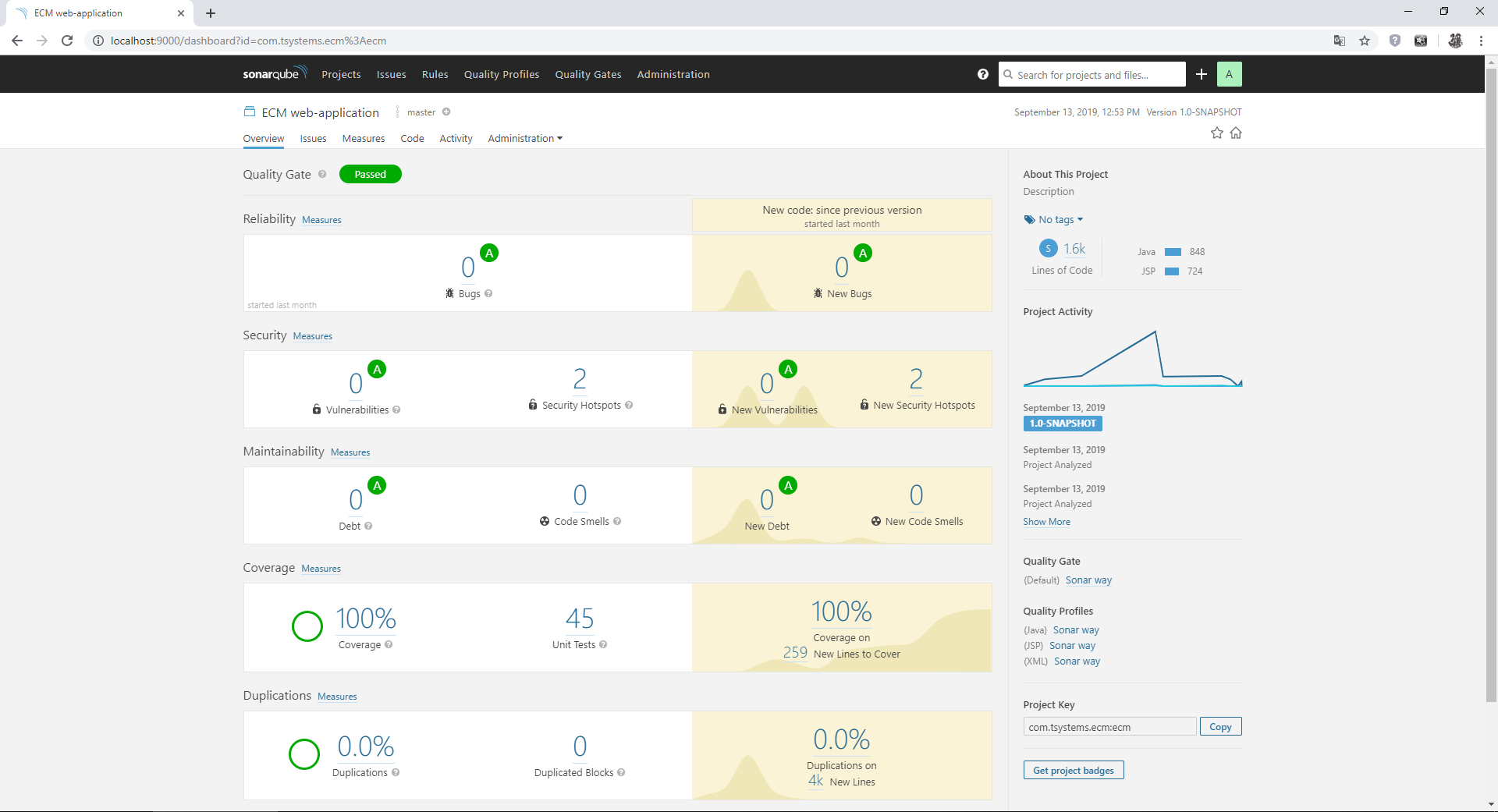
12-09-2019 08:48:12 TRACE PatientServiceImpl:74 - get method called

12-09-2019 08:48:12 TRACE TreatmentServiceImpl:57 - getAll method called

12-09-2019 08:48:12 TRACE AppointmentServiceImpl:87 - get method called

12-09-2019 08:48:12 TRACE RegimenProcessorServiceImpl:40 - parseRegimen method called

Request parameters interceptor is very useful while debugging the controllers’ logics, as it is allows checking the parameters of either GET or POST HTTP method requests.

1. **Project code statistics**
2. **Application improvements**

In further releases of the application, next improvements and features are planned:

* in-application user registration and user management;
* non-hardcoded time picker for appointments;
* appointments’ statuses;
* new event status – in process;
* ability to change discharged patients’ status to «On treatment»;
* ability for «doctor» users to create new treatments;
* the REST service with ability for the patients to view their events, authorization with temporary tokens managed by medical center stuff (eg. a QR-code, that patient can scan and that grants access to this service;
* assignments to events for the «nurse» users, with notifications;
* management of medications quantity;
* autocompletion and advanced search for treatment selection for appointments;
* medical tests feature (i.e. CT, ECG, blood tests etc.), with automatic test results generation and treatment recommendations accessible to «doctor» users;
* patient’s medical history autogeneration accessible to «doctor» users;
* deployment to IaaS providers using Docker, with auto-tests and auto-deploys (CI\CD);
* transition to micro service architecture;
* further localization support;