[Дата]

Веселин Николов

[име на фирмата]

[Заглавие на документа]

[Подзаглавие на документа]

# Задание

## Обобщение:

Да се разработи информационна система – Хотел. Програмата съхранява и обработва данни за хотелски услуги (резервации и допълнителни услуги). Системата позволява множествен достъп.

## Видове потребители:

Системата поддържа два вида потребители администратор и клиенти (рецепционист, мениджър, собственик) с различни роли за достъп до функционалностите в системата.

## Операции за работа с потребители:

* Създаване на собственици на хотел(и) от администратор.
* Създаване на хотел с мениджър от собственик.
* Създаване на рецепционист от мениджър.

## Системата поддържа операции за работа с резервации:

* Създаване на клиенти.
* Създаване на резервация от рецепционист (Номер на резервация, Тип на резервация, Тип на прекратяване на резервацията, категории стаи,…).
* Създаване и предоставяне на допълнителни услуги, съобразени със сезона (отчетност на тип на услуга и брой ползвания).
* Рейтинговане на клиенти.

## Системата поддържа справки по произволен период за:

* Категория клиенти:
  + Информация за клиенти (лични данни).
  + Използвани на хотел и хотелски услуги.
  + Рейтинг на клиенти .
* Рецепционисти (създадени резервации, данни на рецепциониста).
* Създадени резервации (дата, статус, хотел, съдържание на формуляра).
* Стаи (Рейтинг на стаите за ползваемост).

## Допълнителни изисквания:

Мениджър на хотел достъпва справки само за хотел, за който е отговорен. Собственика достъпва справки за всички притежаващи хотели. Рецепциониста има право на справки за заетостта на стаите.

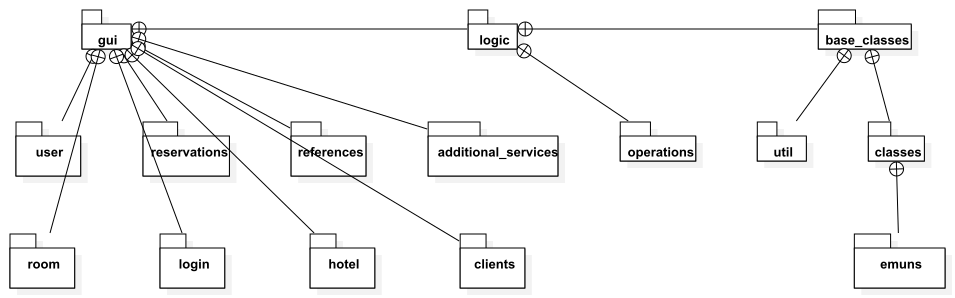
## Системата поддържа Известия за събития:

* Изтичаща резервация.
* Известия за рисков клиент (при създаване на нова резервация).

# Проектиране на системата

## Модули на системата

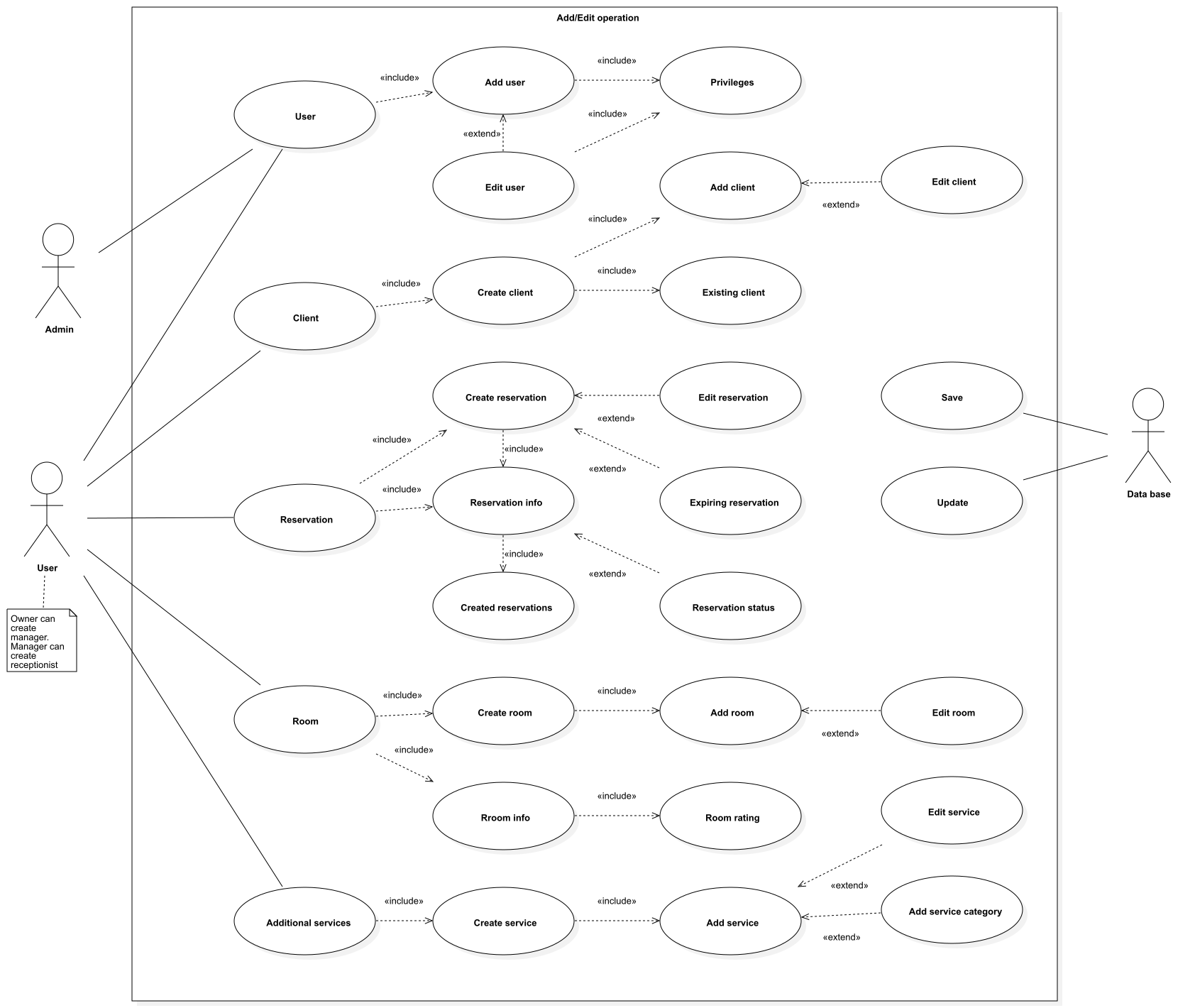
1. Base classes – съдържа базовите класове за целия проект заедно с връзката с базата данни, настройки за hibernate и log4j 2.
2. Business logic – съдържа статични класове. Всички заявки към базата данни минават през DecodeOperation класа. Той връща филтрирани данни в зависимост от типа на операцията.
3. Graphic user interface – съдържа всички изгледи за приложението и техните контролери.



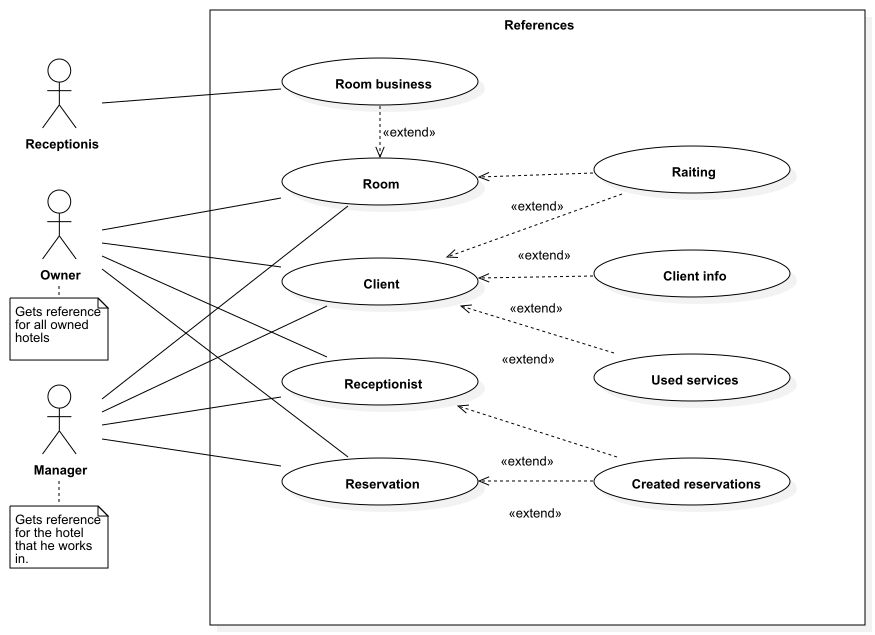
## UML

### Use case diagrams

#### Add/Edit/Delete:

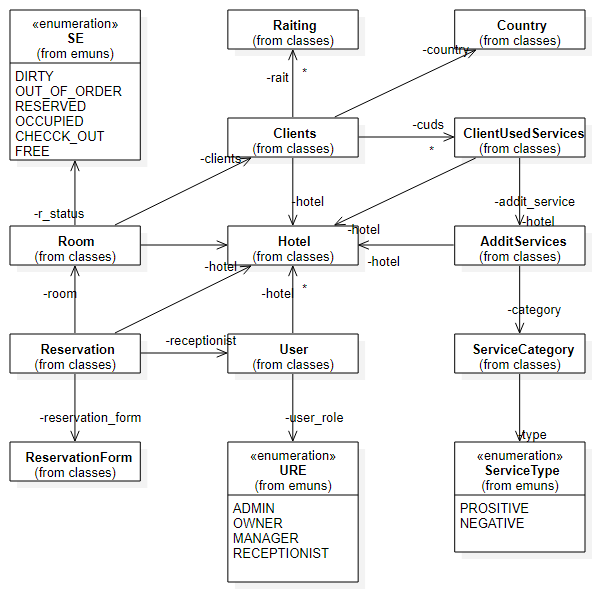


#### Reference

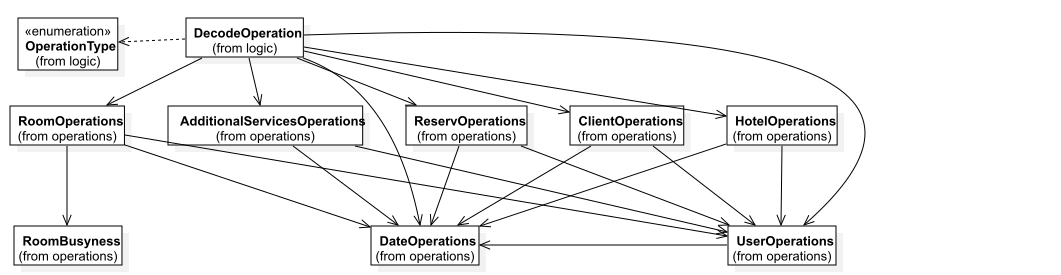


### Class diagrams

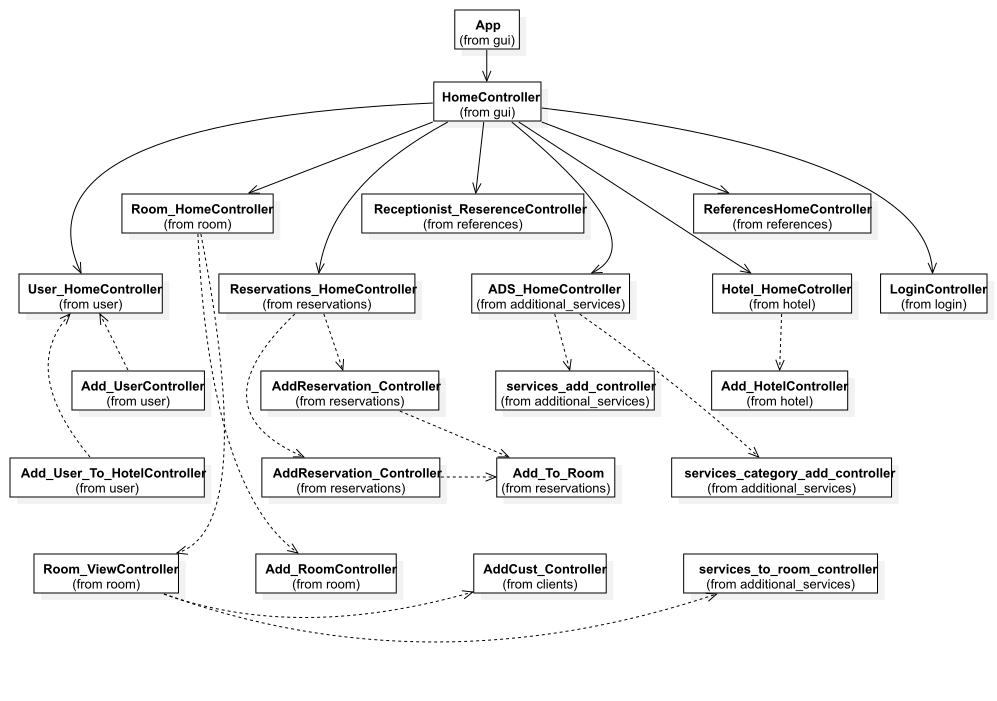
#### Base classes



#### Logic



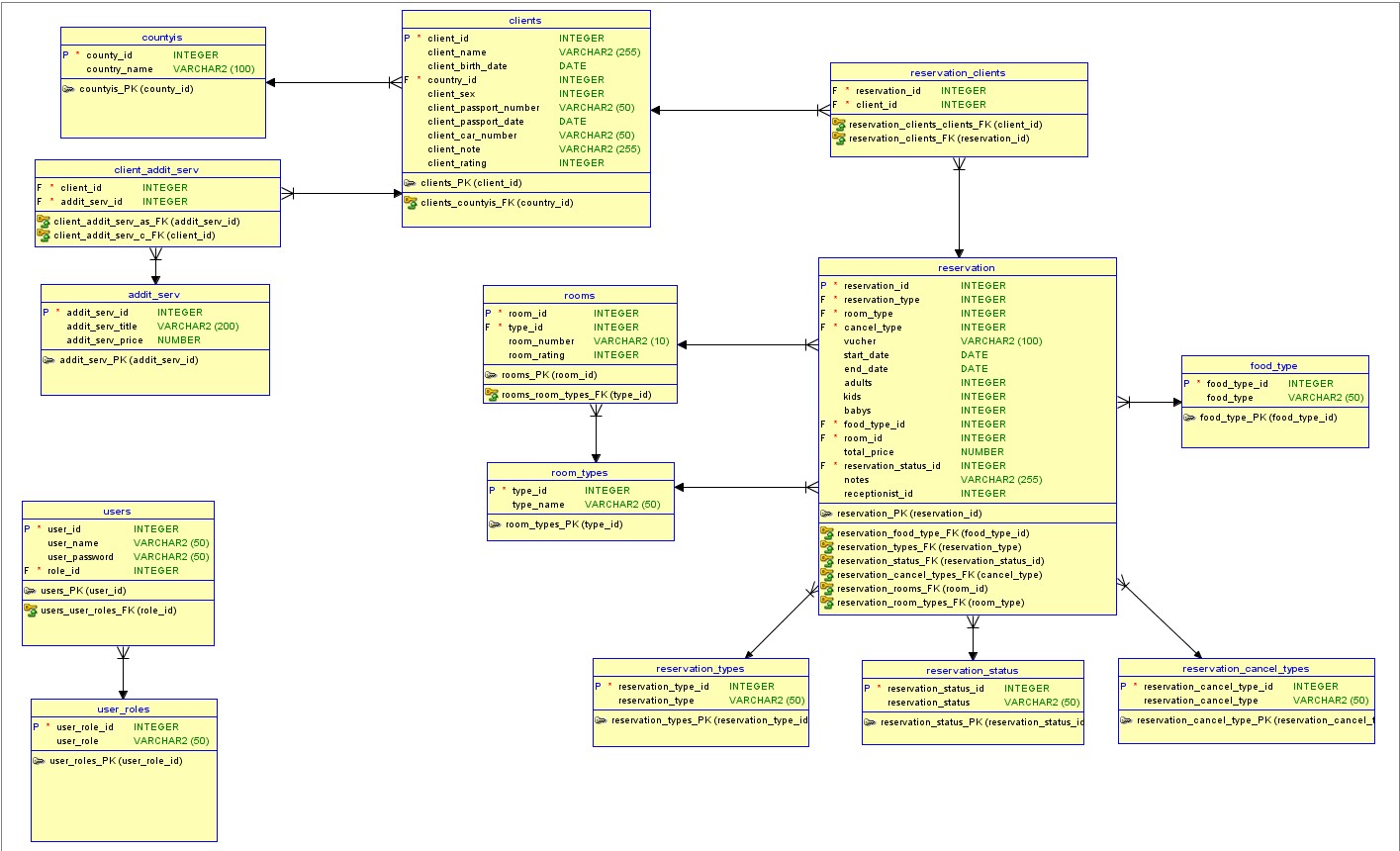
#### GUI



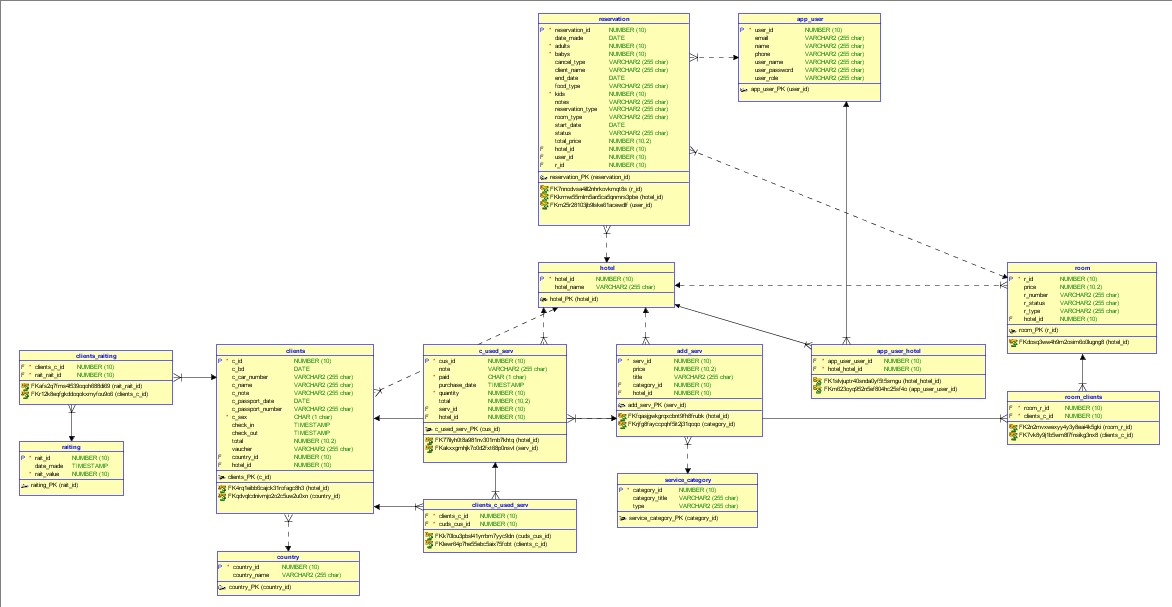
### Sequence diagram

## Модел на базата от данни

### Първоначален модел на базат данни:



### Краен модел на базата данни:

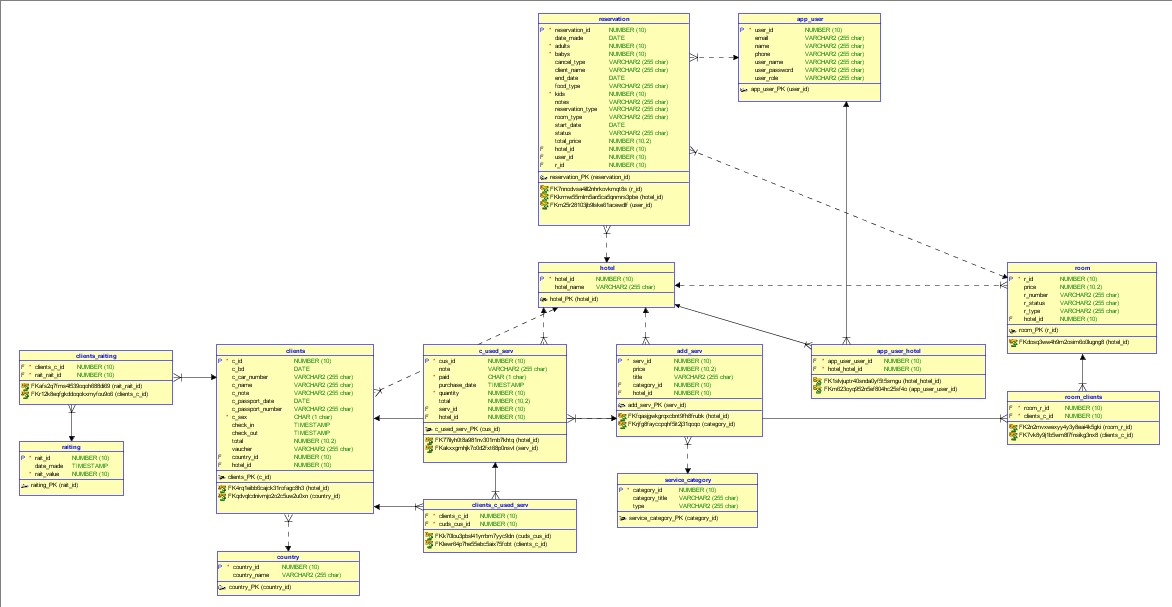


# Реализация на системата

## База данни – Oracle

### Описание на таблиците:

1. Service\_category – съдържа различните категории на предлаганите услуги, Полета: име и тип на категорията, който спада в 2 категории: позитивна и негативна. Според тези категории се определя, рейтинга на клиента.
2. Add\_serv – съхранява предлаганите услуги. Полета: име, цена, категория и хотела на предлаганата услуга.
3. App\_user – съхранява информацията за потребителите. Полета: име, телефонен номер, имейл, потребителско име, парола и роля на потребителите.
4. App\_user\_hotel – междинна таблица за съхранение на връзките между таблиците за хотели и потребители.
5. C\_used\_serv – съхранява използваните услуги от клиентите на хотела. Полета: код на услуга, дата на поръчка(използване), количество, обща цена, бележка и дали е платена.
6. Country – съхранява имената на държавите от които са клиентите на хотела.
7. Hotel – съхранява имената на хотелите.
8. Rating – съхранява рейтинга. Полета: дата на създаване и стойност на рейтинга.
9. Room – съхранява информация за стаите. Полета: номер, цена, тип, хотел, статус (като: заета, свободна, в ремонт, резервирана, мръсна и заминаваща(check out)).
10. Clients – съхранява информацията за клиентите на хотела. Полета: име, дата на раждане, номер на паспорт, дата на паспорта, код на страна, код на хотел, ваучер, пол, обща сума на не платените услуги, бележка и дата и час на пристигане и заминаване.
11. RESERVATION – съхранява информацията за резервациите. Полета: тип на резервация, тип на стая, тип на прекратяване, име на клиента направил резервацията, дата на пристигане и заминаване, възрастни, деца, бебета, тип на изхранване, статус на резервацията, обща цена, дата на създаване, потребител направил резервацията, хотел, стая в хотела и бележки.
12. CLIENTS\_C\_USED\_SERV – междинна таблица за връзките между клиентите и използваните услуги.
13. CLIENTS\_RAITING – междинна таблица за връзките между клиентите и техния рейтинг.
14. ROOM\_CLIENTS – междинна таблица за връзките между стоите и клиентите настанени в тях.



## Слой за работа с базата данни – Hibernate

### Слоя за работа с базата данни е реализиран с помощта на Hibernate. Всеки базов клас има анотации за конфигуриране на таблиците.

1. Полетата за ID са реализирани със SequenceGenerator анотация.
2. В класовете с променлива за хотел имат анотация за ManyToOne връзка.
3. Полета от тип boolean се записват като char от една буква (T, F).
4. Полета с тип double се записват, като Number(10,2) с точност 2 знака след запетаята.
5. Полета с тип enum се записват, като varchar2.
6. Полета от пит List са с анотация ManyToMany.
7. Класове Reservation и ReservationForm са с анотации: Embeddable за ReservationForm и Embedded за Reservation.

@Entity(name = "reservation")

public class Reservation {

    @Id

    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "reserv\_generator")

    @SequenceGenerator(name = "reserv\_generator", sequenceName = "reserv\_seq", allocationSize = 1)

    private int reservation\_id;

    @ManyToOne(cascade = CascadeType.ALL)

    @JoinColumn(name = "r\_id")

    private Room room;

    @Embedded

    private ReservationForm reservation\_form;

    @ManyToOne(cascade = CascadeType.ALL)

    @JoinColumn(name = "user\_id")

    private User receptionist;

    @ManyToOne(cascade = CascadeType.ALL)

    @JoinColumn( name = "hotel\_id")

    private Hotel hotel;

### Конфигурация за връзка с базата данни

Конфигурацията за връзка с базата данни става чрез файла hibernate.cfg.xml . С диалект Oracle10gDialect и драйвер ojdbc10.

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <property name="hibernate.connection.driver\_class">oracle.jdbc.driver.OracleDriver</property>

        <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>

        <property name="hibernate.connection.username">test\_</property>

        <property name="hibernate.connection.password">test\_</property>

        <property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>

        <property name="hbm2ddl.auto">update</property>

        <property name="connection.pool\_size">1</property>

        <property name="hibernate.connection.autocommit">after\_statement</property>

        <property name="current\_session\_context\_class">thread</property>

        <property name="cache.provider\_class">org.hibernate.cache.internal.NoCacheProvider</property>

        <property name="show\_sql">true</property>

        <property name="format\_sql">true</property>

        <!--property name="use\_sql\_comments">true</property-->

        <mapping class="base\_classes.classes.AdditServices" />

        <mapping class="base\_classes.classes.Clients" />

        <mapping class="base\_classes.classes.ClientUsedServices" />

        <mapping class="base\_classes.classes.Country" />

        <mapping class="base\_classes.classes.Hotel" />

        <mapping class="base\_classes.classes.Raiting" />

        <mapping class="base\_classes.classes.Reservation" />

        <mapping class="base\_classes.classes.ReservationForm" />

        <mapping class="base\_classes.classes.Room" />

        <mapping class="base\_classes.classes.ServiceCategory"/>

        <mapping class="base\_classes.classes.User"/>

    </session-factory>

</hibernate-configuration>

### Запис, промяна ,изтриване и извличане

Тези операции са реализирани в класа DBConnection.

1. Метод за търсене по ID.
2. Методи за търсене по хотел.
3. Метод за взимане на всички записи за определен клас.
4. Метод за взимане на различните типове стаи и предлагани услуги.
5. Метод за изтриване на данни от всички таблици.

#### Примери:

##### Записване

public void saveObject(Object object) {

        try {

            LOGGER.debug("Starting save.");

            if (session.getTransaction().isActive()) {

                session.getTransaction();

            } else

                session.beginTransaction();

            session.save(object.getClass().toString(), object);

            LOGGER.debug("Saved object: {}", object);

        } catch (Exception e) {

            LOGGER.error("Fail to begin transaction or save failure {}: {}", object, e);

            session.getTransaction().rollback();

        }finally {

            session.getTransaction().commit();

            LOGGER.debug("Save -> end with commit.");

        }

    }

##### Промяна

public void updateObject(Object object) {

        try {

            LOGGER.debug("Starting update.");

            if (session.getTransaction().isActive()){

                session.getTransaction();

            } else

                session.beginTransaction();

            session.update(object.getClass().toString(), object);

            LOGGER.debug("Updated object: {}", object);

        } catch (Exception e) {

            LOGGER.error("Fail to begin transaction or update failure {}: {}", object, e);

            session.getTransaction().rollback();

        }finally {

            session.getTransaction().commit();

            LOGGER.debug("Update -> end with commit.");

        }

    }

##### Изтриване

public void deleteObject(Object object) {

        try {

            LOGGER.debug("Starting delete.");

            if (session.getTransaction().isActive()) {

                session.getTransaction();

            } else

                session.beginTransaction();

            session.delete(object.getClass().toString(), object);

            LOGGER.debug("Deleted object: {}", object);

        } catch (Exception e) {

            LOGGER.error("Fail to begin transaction or deleting {}: {}", object, e);

            session.getTransaction().rollback();

        } finally {

            session.getTransaction().commit();

            LOGGER.debug("Delete -> end with commit.");

        }

    }

##### Търсене по ID

public User getUserByID(int id) {

        try {

            LOGGER.debug("Starting get user by id -> {}", id);

            return session.get(User.class, id);

        } catch (Exception e) {

            LOGGER.error("User with id not found {" + id + "} -> {} ", e);

            return null;

        }

    }

##### Търсене по хотел.

public List<User> getUserByHotel(int hotel\_id) {

        LOGGER.debug("Starting get user by hotel id -> {}", hotel\_id);

        Query<User> query = null;

        String sql\_comand = "select t from " + User.getTableName() + " t join t." + User.getFields().get(4) +

                " h where h." + Hotel.getFields().get(0) + " = :hotel\_id";

        try {

            LOGGER.debug("SQL -> {}", sql\_comand);

            query = session.createQuery(sql\_comand, User.class);

            query.setParameter("hotel\_id", hotel\_id);

            return query.list();

        } catch (Exception e) {

            LOGGER.error("Problem with getting users by hotel id: {} -> {}", hotel\_id, e);

            return null;

        }

    }

##### Взимане на всички записи за определен клас.

public List<User> getAllUsers(){

        LOGGER.debug("getUserByUsername");

        Query<User> query = null;

        String sql\_comand = "from " + User.getTableName();

        try {

            LOGGER.debug("SQL -> {}", sql\_comand);

            query = session.createQuery(sql\_comand, User.class);

            return query.list();

        } catch (Exception e) {

            LOGGER.error("Problem with getting all users. {}", e);

            return null;

        }

    }

##### Взимане на различните типове стаи и предлагани услуги.

public List<String> getDistinctRoomTypes() {

        LOGGER.debug("Starting get distinct room types.");

        String sql = "select DISTINCT r\_type from room";

        try {

            LOGGER.debug("SQL -> {}", sql);

            Query<?> q = session.createQuery(sql);

            List<?> result = q.list();

            List<String> distinct\_roomTypes = new ArrayList<>();

            for (Object object : result) {

                String tmp = (String) object;

                distinct\_roomTypes.add(tmp);

            }

            LOGGER.debug("Result -> {}", distinct\_roomTypes);

            return distinct\_roomTypes;

        } catch (Exception e) {

            LOGGER.error("Problem getting distinct room types -> {}", e);

            return null;

        }

    }

public List<String> getDistinctAdditionalServices() {

        LOGGER.debug("Starting get distinct additional services.");

        String sql = "select DISTINCT title from add\_serv";

        try {

            LOGGER.debug("SQL -> {}", sql);

            Query<?> q = session.createQuery(sql);

            List<?> result = q.list();

            List<String> distinct\_services = new ArrayList<>();

            for (Object object : result) {

                String tmp = (String) object;

                distinct\_services.add(tmp);

            }

            LOGGER.debug("Result -> {}", distinct\_services);

            return distinct\_services;

        } catch (Exception e) {

            LOGGER.error("Problem getting distinct additional services -> {}", e);

            return null;

        }

    }

##### Изтриване на данни от всички таблици.

public void truncateAllTables() {

        LOGGER.debug("Starting truncate all tables.");

        List<String> table\_names = new ArrayList<>();

        table\_names.add(Clients.getTableName());

        table\_names.add(ClientUsedServices.getTableName());

        table\_names.add(AdditServices.getTableName());

        table\_names.add(Country.getTableName());

        table\_names.add(Reservation.getTableName());

        table\_names.add(Room.getTableName());

        table\_names.add(Raiting.getTableName());

        table\_names.add(ServiceCategory.getTableName());

        table\_names.add(User.getTableName());

        table\_names.add(Hotel.getTableName());

        Query<?> q = null;

        for (String string : table\_names) {

            String sql = " delete from ";

            sql += string;

            try {

                LOGGER.debug("Sql command -> {}", sql);

                session.beginTransaction();

                q = session.createQuery(sql);

                q.executeUpdate();

                session.getTransaction().commit();

                LOGGER.debug("Succesfuly truncated table -> {}", string);

            } catch (Exception e) {

                LOGGER.error("Error while truncating tables -> {}", e);

            }

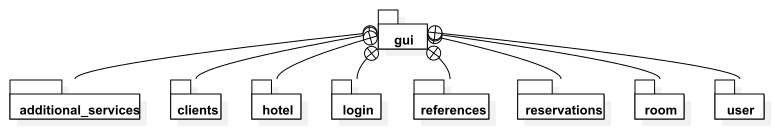
        }

    }

## Графичен интерфейс

Главно използвани елементи: Label, TableView, TableColumn, Button, DatePicker, ComboBox и TestField.

Всеки от главните изгледи съдържа бутон за добавяне на нов обект. При кликване с мишката два пъти се отваря прозорец за редактиране на елемента. Изтриване се реализира със избиране на елемент от таблицата и натискане на бутона Delete.



## Бизнес логика

### Base classes module

#### DBConnection

Клас съдържащ методи за работа с базата данни.

Методи:

1. За търсене по ID.
2. За търсене по хотел.
3. За взимане на всички записи за определен клас.
4. За взимане на различните типове стаи и предлагани услуги.
5. За изтриване на данни от всички таблици.

#### Util

HibernateUtil – клас за инициализиране на връзка с базата данни.

Методи:

1. Създаване на сесия.
2. Връщане на сесия.
3. Спиране на сесия.

#### Classes

Всички класове съдържат get и set методи за всички променливи, 2 или 3 конструктора. Методи toStrin, getTableName, getFields.

1. AdditServices –полета: име, цена, категория и хотела на предлаганата услуга.
2. Clients – полета: име, дата на раждане, номер на паспорт, дата на паспорта, код на страна, код на хотел, ваучер, пол, обща сума на не платените услуги, бележка и дата и час на пристигане и заминаване, списък с използваните услуги.
3. ClientUsedServices – полета: услуга, дата на поръчка(използване), количество, обща цена, бележка, дали е платена и хотел.
4. Country – полета: име на държава.
5. Rating – полета: дата на създаване и стойност на рейтинга.
6. Reservation – полета: дата на създаване, потребител направил резервацията, стая в хотела и обект от ReservationForm.
7. ReservationForm - полета: тип на резервация, тип на стая, тип на прекратяване, име на клиента направил резервацията, дата на пристигане и заминаване, възрастни, деца, бебета, тип на изхранване, статус на резервацията, обща цена, бележки
8. Room – полета: тип на стаята, номер, цена, статус, хотел и списък с клиенти.
9. ServiceCategory – полета: име на категория и тип на категория.
10. User – полета: име, телефон, имейл, потребителско име, парола, роля и списък на хотелите.

##### Enums

1. SE- съдържа състоянията на стаите.
2. ServiceType – съдържа 2-та типа на категориите услуги (позитивни, негативни).
3. URE – съдържа ролите на потребителите използващи програмата.

### Business logic module

DecodeOperation – междинен клас, който служи за връзка между графичния интерфейс(gui) и слоя за връзка с базата данни. Той приема заявки от gui и ги разпределя в зависимост от типа на операцията.

OperationType – съдържа различните типове операции свързани с базата данни.

#### Operations

Всеки от класовете взима данни от базата данни и при необходимост ги филтрира в зависимост от желания резултат.

1. AdditionalServicesOperations методи:
   1. За взимане на допълнителните услуги.
2. ClientOperations методи:
   1. Взимане на информацията за клиентите в даден период.
   2. Взимане на използваните допълнителни услуги от клиенти в даден период.
   3. Взимане на рейтинга на клиентите в даден период.
3. DateOperations методи:
   1. Преобразуване от стринг в дата.
   2. Преобразуване от стринг в дата и час в началото на деня.
   3. Преобразуване от стринг в дата и час в края на деня.
   4. Сравняване на дата и час, дали е равна или между други две дати и час.
   5. Сравняване на дата, дали е равна или между други две дати.
   6. Сравняване на две дати дали са равни.
4. HotelOperations методи:
   1. Взимане на хотели.
5. ReservOperations методи:
   1. Взимане на резервации за определена дата.
   2. Взимане на резервациите, които си заминават за деня.
   3. Взимане на създадените резервации за определен период.
   4. Взимане на създадените резервации от рецепционист за определен период.
6. RoomOperations методи:
   1. Взимане на стаи.
   2. Изчисляване на рейтинг на стаите за определен период.
   3. За премахване на клиенти от стаята (check out).
   4. Взимане на различните типове стаи.
   5. Изчисляване на заетостта на стаите в даден период.
   6. За запазване на списък с временните стаи.
   7. Взимане на списък с временните стаи.
   8. Добавяне на временна стая.
7. UserOperations методи:
   1. За потвърждаване за даден потребител.
   2. Взимане на потребителите.
   3. Взимане на рецепционистите.
   4. За добавяне на потребител.
8. RoomBusyness – клас за запазване на използването на стайте по типове, за определена дата.

### Graphical user interface module

App – е начална точка на приложението. Зарежда login изгледа.

HomeController – котролер на главния изглед, в зависимосто от кой бутон е натиснат, се зарежда дадената сцена.

#### User

##### User\_HomeController

Главна сцена за потребителите.

Методи:

1. Добавяне на потребители, зарежда в зависимост от потребителя, различните изгледи.
2. Промяна на вече съществуващ потребител.
3. Изтриване на потребители (с права за администратор).
4. Инициализиране.

##### Add\_UserController

Сцена за добавяне на потребители без списък за хотели.

Методи:

1. Инициализиране.
2. Запазване или промяна на потребител.
3. Изобразяване на информацията за вече съществуващ потребител ( за промяна).

##### Add\_User\_To\_HotelController

Сцена за добавяне на потребител със списък за хотели.

Методи:

1. Инициализиране.
2. Запазване или промяна на потребител.
3. Изобразяване на информацията за вече съществуващ потребител ( за промяна).

#### Room

##### Room\_ViewController

Изглед стая с клиенти и техните използвани услуги.

Методи:

1. Инициализиране.
2. Плащане на услуга.
3. Изтриване на услуга( от мениджър).
4. Добавяне на клиенти към стаята.
5. Добавяне на услуга към клиент.
6. Зареждане на данни.

##### Room\_HomeController

Главен изглед за стаи.

Методи:

1. Инициализиране.
2. Отваряне на селектирана стая.
3. Добавяне на стаи.
4. Зареждане на информация.
5. При натискане на клавиш от клавиатурата.
   1. От мръсна към свободна (чиста).
   2. От свободна към ремонт ( от ремонт към мръсна).

##### Add\_RoomController

Изглед за добавяне на стая.

Методи:

1. Инициализиране.
2. Запазване.

#### Reservations

##### Reservations\_HomeController

Главен изглед за резервации.

Методи:

1. Инициализиране.
2. Изтриване на резервация.
3. Промяна на резервация.
4. Смяна на датата.
5. Добавяне на резервация.

##### Add\_To\_Room

##### editreservation\_Controller

##### chooseRoom\_controller

##### AddReservation\_Controller

#### References

##### Receptionist\_ReserenceController

Изглед справка за заетостта (за рецепционист). По зададени начална и крайна дата.

Методи:

1. Инициализиране.
2. Търсене (нова справка).

##### ReferencesHomeController

Изглед справки. По зададени начална и крайна дата.

Методи:

1. Инициализиране.
2. Рейтинг на стаите.
3. Информация за клиенти.
4. Рейтинг на клиенти.
5. Създадени резервации.
6. Изтриване на TableView.
7. Изтриване на VBox.
8. Създадени резервации от рецепционист.
9. Използвани услуги.

#### Notification

##### NotificationController

Изглед известия.

Методи:

1. Инициализиране.
2. Изчистване на списъка.
3. Get/set методи за списъка с известия.

#### Login

##### LoginController

Изглед за вход в системата. Метод за изобразяване на отказан вход заедно с известие.

#### Hotel

##### Hotel\_HomeCotroller

Главен изглед за хотели.

Методи:

1. Инициализиране.
2. Зареждане на данни.
3. Добавяне на хотел.
4. Редактиране на хотел.
5. Изтриване на хотел.

##### Add\_HotelController

Изглед за добавяне на хотел.

Методи:

1. Инициализиране.
2. Set метод за хотел.
3. Запазване.

#### Clients

##### AddCust\_Controller

Изглед за добавяне на клиенти.

Методи:

1. Инициализиране.
2. Добавяне на клиент.
3. Get/Set методи за променлива клиент.

#### Additional\_services

##### ADS\_HomeController

Главен изглед за предлагани услуги.

Методи:

1. Инициализиране.
2. Добавяне на услуга.

##### services\_add\_controller

##### services\_category\_add\_controller

##### services\_to\_room\_controller

## Регистриране на събития в системата – log4j 2

Събитията се записват във файл за съответния ден в папка logs.

1. Ниво trace се записва във файл debug.log.
2. Ниво debug се записва във файл debug.log.
3. Ниво info се записва във файл app-logs.log.
4. Ниво error се записва във файл app-logs.log.

### Конфигурация на лог системата

<?xml version="1.0" encoding="UTF-8"?>

<Configuration>

    <Properties>

        <Property name="basePath">../logs</Property>

    </Properties>

    <Appenders>

        <RollingFile name="fileLogger" fileName="${basePath}/app-logs.log" filePattern="${basePath}/app-logs-%d{yyyy-MM-dd}.log">

            <PatternLayout pattern="[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{36} - %msg%n" />

            <Policies>

                <TimeBasedTriggeringPolicy interval="1" modulate="true" />

            </Policies>

        </RollingFile>

        <RollingFile name="debugFile" fileName="${basePath}/debug.log" filePattern="${basePath}/debug-%d{yyyy-MM-dd}.log">

            <PatternLayout pattern="[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{36} - %msg%n" />

            <Policies>

                <TimeBasedTriggeringPolicy interval="1" modulate="true" />

            </Policies>

        </RollingFile>

        <Console name="console" target="SYSTEM\_OUT">

            <PatternLayout pattern="[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{36} - %msg%n" />

        </Console>

    </Appenders>

    <Loggers>

        <Logger name="org.hibernate" level="error" additivity="false">

            <AppenderRef ref="debugFile" />

            <AppenderRef ref="fileLogger" />

        </Logger>

        <Logger name="org.hibernate.SQL" level="debug" additivity="false">

            <AppenderRef ref="debugFile" />

        </Logger>

        <Logger name="org.hibernate.type.descriptor.sql" level="trace" additivity="false">

            <AppenderRef ref="debugFile" />

        </Logger>

        <Logger name="base\_classes" level="debug" additivity="false">

            <AppenderRef ref="debugFile" />

        </Logger>

        <Logger name="logic" level="debug" additivity="false">

            <AppenderRef ref="debugFile" />

        </Logger>

        <Logger name="gui" level="debug" additivity="false">

            <AppenderRef ref="debugFile" />

        </Logger>

        <Logger name="gui" level="info" additivity="false">

            <AppenderRef ref="fileLogger" />

        </Logger>

        <Root level="info" additivity="false">

            <appender-ref ref="console" />

            <AppenderRef ref="fileLogger" />

        </Root>

    </Loggers>

</Configuration>

# Тестови резултати