МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

**КУРСОВОЙ ПРОЕКТ**

по дисциплине: «Моделирование и оптимизация проектирования техническим систем»

на тему: «***Профилирование backend Java приложений, взаимодействующих с БД***»

Выполнил: М.А. Лосев

студент группы 10701118

Руководитель: А.А. Прихожий

проф. д.т.н.

Минск 2021

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе**

по дисциплине: «Моделирование и оптимизация проектирования техническим систем»

на тему: «***Профилирование backend Java приложений, взаимодействующих с БД***»

Выполнил: М.А. Лосев

студент группы 10701118

Руководитель: А.А. Прихожий

проф. д.т.н.

Минск 2021

**ОГЛАВЛЕНИЕ**

ВВЕДЕНИЕ ...…………………………………………………………………….……....4

1 Постановка задачи ……………………………………………………………………..5

2 Модель решения задачи………………………………………………………………..6

3 Метод решения задачи ………………………………………………………………...7

4 Алгоритм решения задачи …………………………………………………………….9

5 Информационная модель ………………………………...…………………………..14

6 Архитектура программы ……………………………………………………………..16

6.1 Слой данных ………………………………………………………..…………….18

6.2 Слой сервиса ……………………………………………………………………...19

6.3 Слой контроллера ………………………………………………………………...20

6.4 Слой View …………………………………………………………………………20

7 Графический интерфейс …………………………………………………………….. 21

7.1 Графический интерфейс веб-приложения ………………………………………21

7.2 Графический интерфейс отчета профилирования ……………………………...23

8 Тестовый пример. Тестирование программы ………………………………………24

9 Демонстрация работы программы …………………………………………………..27

ЗАКЛЮЧЕНИЕ ………………………………………………………………………...29

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ ………………………………………30

ПРИЛОЖЕНИЕ А ……………………………………………………………………...31

# **ВВЕДЕНИЕ**

Профилирование — сбор характеристик работы программы, таких как время выполнения отдельных фрагментов (подпрограмм/методов), а также ресурсы системы. Инструмент, используемый для анализа работы, называют профилировщиком или профайлером (англ. *profiler*). Обычно выполняется совместно с оптимизацией программы.

Инструментальные средства анализа программы чрезвычайно важны для того, чтобы понять поведение программы. Проектировщики ПО нуждаются в таких инструментальных средствах, чтобы оценить, как хорошо выполнена работа. Программисты нуждаются в инструментальных средствах, чтобы проанализировать их программы и идентифицировать критические участки программы.

Это часто используется, чтобы определить, как долго выполняются определенные части программы. Затем эти участки могут быть оптимизированы, чтобы выполняться быстрее.

В качестве приложения для оптимизации было выбрано веб-приложение собственной разработки «Hotel Booking», представляющее собой систему для бронирования комнат в отеле. Приложение состоит из Java сервера и использует MySQL в качестве СУБД.

В системе предусмотрено три роли пользователей:

• Администратор – зарегистрированный пользователь, имеющий доступ к панели администратора.

• Клиент – зарегистрированный пользователь, имеющий доступ к использованию услуг приложения.

• Незарегистрированный пользователь, имеющий возможность зайти / зарегистрироваться в системе, а также имеющий доступ к главной странице.

В соответствии с этими ролями система делится на три части: главная страница, доступная для всех пользователей, сторона клиента отеля и сторона администратора.

# **1 Постановка задачи**

Необходимо провести тестирование скорости работы приложения, идентифицировать медленные участки и провести оптимизацию, если таковые имеются.

В случае наличия медленно выполняемого функционала необходимо получить точное время выполнения, при этом для проведения измерений исходный код бизнес-логики должен не меняться или меняться незначительно, то есть инструмент должен позволять получать необходимые данные с минимальной модификацией кода измеряемых участков приложения.

# **2 Модель решения задачи**

В поставленной задаче будет рассматриваться профилирование отдельных методов классов. Инструмент должен быть универсальным, в результате использования которого ожидается документ (отчет), содержащий следующую информацию:

* дату и время генерации отчета;
* время выполнения участка кода;
* сведения об операционной системе;
* другие возможные данные, представляемые средой выполнения.

Функционал веб-приложения состои из следующих частей:

* Неаутентифицированный пользователь имеет возможность войти, введя логин и пароль или зарегистрироваться.
* Пользователю (клиенту) системы доступен функционал бронирования комнат.
* Администратор просматривает поступившие заявки от пользователей и выставляет счет.

# **3 Метод решения задачи**

Для решения поставленной задачи было решено провести первичную идентификацию медленных участков путем ручного тестирования. Так как основными сущностями системы являются Пользователь (User) и Заказ (Reservation), преварительно были сгенерированы тестовые данные указанных классов и сохранены в базу данных.

Для получения данных о системе было использовано системное API, доступное в Java. Настройки генерации и профилирования были вынесены в конфигурационный файл. Таким образом, возможно изменение параметров без необходимости изменения исходного кода приложения.

В качестве формата отчета был выбран HTML, так как данный формат документа поддерживается всеми предустановленными браузерами в современных операционных системах.

Помимо этого, для профилирования методов было использовано аспектно-ориентированное программирование. Аспектно-ориентированное программирование (АОП) — это парадигма программирования, являющейся дальнейшим развитием процедурного и объектно-ориентированного программирования (ООП). Идея АОП заключается в выделении так называемой сквозной функциональности. Данный подход позволяет ввести доп. функциональность (в данном случае измерение времени выполнения и других характеристик), не имея прямого отношения к модулям и бизнес-логике.

В аспектно-ориентированном программировании используется понятие Join point. Join point – это точка наблюдения, присоединения к коду, где планируется введение функциональности.

Следующее понятие – Pointcut – это срез, запрос точек присоединения. Правила запросов точек очень разнообразные, например, таким образом можно получить методы (join point), имеющие необходимую аннотацию, и затем выполнить нужные операции над этими точками (join point).

Advice – это набор инструкций, выполняемых на точках среза (Pointcut). Инструкции можно выполнять по событию разных типов:

* Before – перед вызовом метода;
* After – после вызова метода;
* After returning – после возврата значения из функции;
* After throwing – в случае exception;
* After finally – после выполнения блока finally;
* Around – можно сделать пред-, пост-обработку метода, а также обойти вызов метода.

Aspect – модуль, в котором собраны описания Pointcut и Advice.

В результате был создан аспект, профилирующий программу путем измерения времени выполнения метода и остальных параметров, и сохраняющий полученные данные в HTML отчет.

# **4 Алгоритм решения задачи**

*Алгоритм профилировщика*

Профилировщик реализован с использованием аспектно-ориентированного программирования и представляет собой аспект, внедряемый с помощью фреймворка Spring AOP в приложение. Была создана аннотация @Profiled, которую можно указать над профилируемым методом. При этом стоит отметить, что аспекты могут быть применены только к объектам, находящимся под управлением Spring контейнера, то есть к объектам, отмеченными аннотациями @Component, @Service, @Repository, @Controller. При указании этих аннотаций контейнер Spring создает прокси помеченных классов и управляет их жизненным циклом, имея возможность применить функционал аспекта.

Для получения времени выполнения был использован алгоритм, блок-схема которого приведена на рисунке 4.1

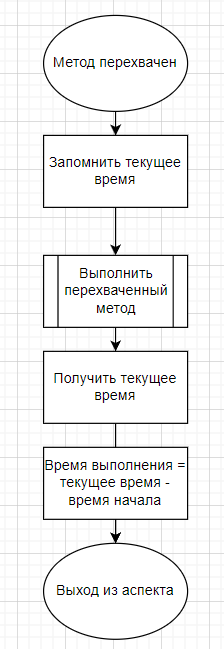


Рисунок 4.1 – Алгоритм получения времени выполнения

Для измерения используемой оперативной памяти в текущий момент используется формула 4.1

M = Mt – Mf, (4.1)

где M – используемая память, Mt – вся выделенная память, Mf – свободная память.

Java предоставляет API для получения памяти посредством методов Runtime.getRuntime().totalMemory() и Runtime.getRuntime().freeMemory соответственно.

Для измерения в течение выполнения метода был выбран следующий алгоритм: расчет памяти происходит в отдельном потоке, что позволяет одновременно выполнять перехваченный метод и производить измерения. Когда метод выполнен, поток останавливается. Управление потоками происходит с помощью ExecutorService.

Остальные данные, такие как сведения о системе и виртуальной машине получаются с помощью созданного класса ProfilingUtils, который в свою очередь использует метод System.getProperty().

Затем полученные результаты отправляются в ReportService, который создает HTML отчет. Исходный код аспекта профилирования представлен на рисунке 4.2.

|  |
| --- |
| @Aspect @Component public class ProfilingAspect {   @Autowired  private ReportService reportService;   @Value("${profiling.memory.measure.step.millis}")  private long memoryMeasureStepMillis;   private ExecutorService executorService = Executors.*newCachedThreadPool*();   @Around("@annotation(Profiled)")  public Object profileMethod(ProceedingJoinPoint joinPoint) throws Throwable {  long start = System.*currentTimeMillis*();  List<MemoryUsage> memoryUsages = new ArrayList<>();  Future<?> punctuation = executorService.submit(() -> punctuateMemoryUsage(memoryUsages));  try {  return joinPoint.proceed();  } finally {  long executionTime = System.*currentTimeMillis*() - start;  punctuation.cancel(true);  ExecutionInfo executionInfo = ExecutionInfo.*builder*()  .executionTimeMillis(executionTime)  .methodName(joinPoint.getSignature().toShortString())  .memoryUsageByTime(memoryUsages)  .build();  ProfilingReport report = ProfilingReport.*builder*()  .jreInfo(ProfilingUtils.*getJreInfo*())  .systemInfo(ProfilingUtils.*getSystemInfo*())  .executionInfo(executionInfo)  .generationTime(Instant.*now*())  .build();  reportService.generateReport(report);  }  }  @SneakyThrows  private void punctuateMemoryUsage(List<MemoryUsage> memoryUsages) {  while (true) {  Instant now = Instant.*now*();  double value = ProfilingUtils.*getMemoryUsageInMb*();  MemoryUsage memoryUsage = new MemoryUsage(now, value);  memoryUsages.add(memoryUsage);  TimeUnit.*MILLISECONDS*.sleep(memoryMeasureStepMillis);  }  } } |

Рисунок 4.2 – Аспект профилирования

*Оптимизируемый участок программы*

В ходе тестирования приложения была проверена скорость работы приложения, и идентифицирован медленный участок выполнения программы – получение списка всех заказов пользователей администратором. В классе ReservationServiceHelperImpl была обнаружена так называемая проблема N+1 запросов.

Проблема N+1 возникает, когда фреймворк доступа к данным выполняет N дополнительных SQL-запросов для получения тех же данных, которые можно получить при выполнении одного SQL-запроса. Чем больше значение N, тем больше запросов будет выполнено и тем больше влияние на производительность. Проблема заключается в выполнении множества дополнительных запросов, которые в сумме выполняются уже существенное время, влияющее на быстродействие.

В данном случае для получения списка заказов всех пользователей сперва делается запрос на получение списка всех пользователей, а затем для каждого пользователя выполняется запрос на получение заказов данного пользователя. Исходной код метода приведен на рисунке 4.3.

|  |
| --- |
| @Profiled @Override @Transactional public List<Reservation> findAllReservations() {  List<User> users = userRepository.findAll();  List<Reservation> reservations = new ArrayList<>();  for (User user : users) {  reservations.addAll(reservationRepository.findByUserId(user.getId()));  }  return reservations; } |

Рисунок 4.3 – Неоптимизированный метод findAllReservations

Для решения проблемы N+1 был создан класс NewServiceHelperImpl. В данной реализации запрос на получение всех заказов выполняется напрямую в reservationRepository. Таким образом, вместо множества запросов в БД выполняется только один запрос.

Но так как сущность Заказ (Reservation) имеет отношение ManyToOne к пользователю (User), то фреймворк также выполняет запросы на получение пользователей, так как при данном отношении стратегия FetchType по умолчанию — Eager. Для решения этой проблемы стратегия получения связанных сущностей была изменена на Lazy. Таким образом, запрос на получение пользователя заказа будет выполнен только в тот момент, когда понадобятся данные этого пользователя, а не сразу при чтении самого заказа. Результаты оптимизации приведены на рисунке 4.4.

|  |
| --- |
| @Profiled @Override public List<Reservation> findAllReservations() {  return repository.findAll(); }  @Entity @Table(name = "reservation") public class Reservation implements Identifiable, Serializable {   private static final long *serialVersionUID* = 4646050451695840394L;  @Id  @GeneratedValue(strategy = GenerationType.*AUTO*)  private Integer id;   @ManyToOne(fetch = FetchType.*LAZY*)  @JoinColumn(name = "user\_id")  private User user;  // other fields  } |

Рисунок 4.4 – Оптимизированная версия программы

# **5 Информационная модель**

*Входные данные.*

* Файл конфигурации application.properties:

*# Data set*  
test.data.generation.enabled=false  
test.data.generation.users=2000  
test.data.generation.reservations.per.user=5  
  
*# Profiling*  
profiling.report.base.folder=target/reports  
profiling.memory.measure.step.millis=1000

*test.data.generation.enabled* – свойство, отвечающее за включение или выключение генерации тестовых данных.

*test.data.generation.users* – количество пользователей, которые будут записаны в базу данных.

*test.data.generation.reservations.per.user* – количество заказов каждого пользователя, которые также будут записаны в базу данных.

profiling.report.base.folder – директория, в которой будут храниться полученные отчеты.

*profiling.memory.measure.step.millis* – промежутки времени, через которые будут браться текущие значения оперативной памяти, используемой виртуальной машиной Java.

* Аннотация @Profiled – данная аннотация является маркером над методом, необходимым для измерения. Методы, отмеченные данной аннотацией будут найдены профилировщиком и будут получены данные об их выполении.

*Выходные данные.*

Отчет, содержащий:

* дату генерации;
* название и версию операционной системы;
* архитектуру системы;
* версию и производителя виртуальной машины;
* имя выполняемого метода;
* время выполнения в секундах.

*Промежуточные данные.*

* Список пар время-значение memoryUsageByTime, содержащий время и используемую память, и заполняемый постепенно и используемый для построения графика.
* ProfilingReport – объект, содержащий измеренные значения, из которого генерируется HTML файл. Данный класс, а также содержащиеся в нем объекты приведены на рисунке 5.1.

|  |
| --- |
| @Builder @AllArgsConstructor @Getter public class ProfilingReport {  private JreInfo jreInfo;  private SystemInfo systemInfo;  private ExecutionInfo executionInfo;  private Instant generationTime; }  @Data public class ExecutionInfo {  private String methodName;  private final long executionTimeMillis;  private List<MemoryUsage> memoryUsageByTime; }  @Data public class JreInfo {  private final String vendor;  private final String version; }  @Data public class MemoryUsage {  private final Instant instant;  private final double usageInMb; }  @Data public class SystemInfo {  private final String osArchitecture;  private final String osName;  private final String osVersion; } |

Рисунок 5.1 – Промежуточные модели системы

# **6 Архитектура программы**

Web-приложение было написано с использованием языка Java 8. Разработка велась в интегрированной среде Intellij IDEA Ultimate Edition x64.

Backend часть реализована с помощью технологии Java EE и состоит из следующих компонентов:

* Spring MVC – для обработки HTTP-запросов;
* Spring Boot, Core – основной фреймворк для внедрения зависимостей и объединения модулей;
* SL4J – для логгирования;
* Maven – для сборки приложения;
* MySQL – в качестве СУБД;
* Spring Data JPA – для взаимодействия с базой данных.
* Tomcat (Spring) – в качестве веб-сервера.
* JSTL + JSP – шаблонные движки, которые транслируются в Java-код, выполняются на стороне сервера и превращаются в сгенерированные HTML-страницы для дальнейшей отправки клиенту. Данные страницы стилизованы с использованием CSS. Также применяется JavaScript для динамических компонентов.

Данные приложения хранятся в базе данных MySQL. Для взаимодействия с ней использовался инструмент MySQL Workbench, а также интерфейс командной строки “mysql”. Для работы с данными из БД в проекте используется технология JPA, позволяющая описать сущности базы данных программно и обращаться к ним с помощью интерфейсов-репозиториев.

Для валидации backend части были написаны отдельные классы, а для валидации frontend части используются средства HTML-валидации.

Сборка проекта осуществляется инструментом Maven в формате JAR, который затем запускается в веб-сервере Tomcat, встроенным в Spring Boot приложение.

Архитектура проекта построена по шаблону Model-View-Controller и состоит из контроллеров, принимающих запросы от пользователя и возвращающего ответы в виде HTML-страниц; сервис-классов, содержащих бизнес-логику; слоя данных, который представляет собой СУБД, классы-репозитории и сущности.

Общая архитектура приложения представлена на рисунке 6.1*.*



Рисунок 6.1 – Общая архитектура приложения

Исходный код, находящийся в папке src, состоит из двух пакетов: main и test. В пакете main содержатся классы приложения, файлы ресурсов, компоненты веб-приложения. Структура проекта представлена на рисунке 6.2.

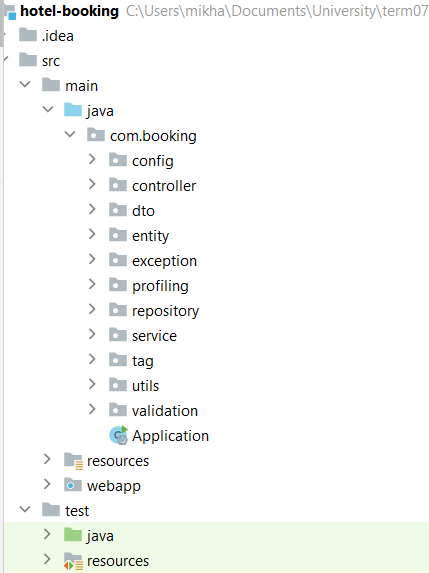


Рисунок 6.2 – Структура проекта

## **6**.1 Слой данных

Классы, реализующие слой данных находятся в подпакетах repository, entity. UML-диаграмма классов предметной области, расположенных в пакете entity, приведена на рисунке 6.3.

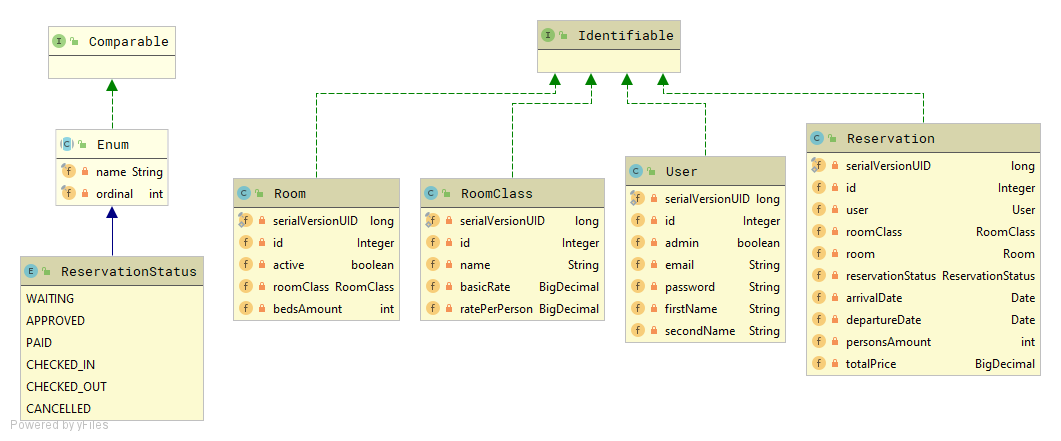


Рисунок 6.3 – UML-диаграмма классов-сущностей

Здесь в качестве маркерного интерфейса выступает интерфейс Identifiable, обозначающий, что реализующая его сущность имеет поле id, соответствующее первичному ключу, и может быть сохранена в базу данных. Помимо полей все сущности имеют методы-геттеры, сеттеры, переопределенные equals и hashCode.

На основе данных классов была спроектирована база данных, схема который представлена на рисунке 6.4.

В таблице заказа «reservation» помимо номера комнаты хранится внешний ключ на желаемый класс (тип апартаментов) комнаты. Это необходимо, поскольку в момент заказа комната, относящаяся к этому заказу не определена, но желаемый тип апартаментов известен заранее.

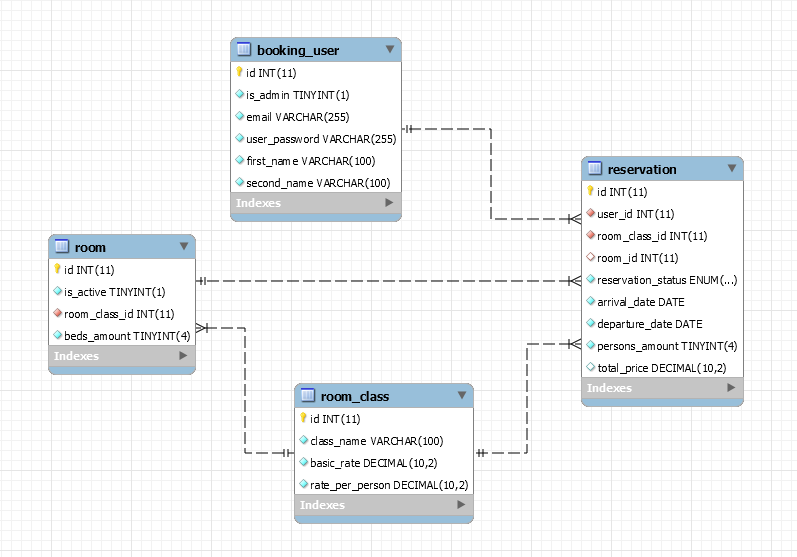


Рисунок 6.4 – Схема базы данных

Для программного взаимодействия с базой данных в классы сущности были добавлены JPA аннотации, а также созданы интерфейсы JPA репозиториев.

Данные классы объявляют сигнатуры методов, необходимых для базового взаимодействия с данными в хранилище: чтение, сохранение, модификация.

## **6**.2 Слой сервиса

Сервис в приложении необходим для выполнения бизнес-логики программы.

Были созданы следующие интерфейсы сервиса: ReservationService – для выполнения логики заказов, ReservationServiceHelper — для вспомогательной логики, RoomClassService – получения доступных классов и обновления цен, RoomService – получения списка комнат и обновления их статуса и UserService, который содержит логику аутентификации и регистрации.

## **6**.3 Слой контроллера

Данный слой состоит из классов, использующих технологию Spring MVC, и принимающих все запросы от пользователей и возвращающий ответы клиентской части. Поддерживаемые HTTP-методы: GET, POST.

Сама логика бизнес-логика в контроллере не выполняется, а вызывается нужный сервис.

После обработки запроса происходит redirect, то есть возврат ответа пользователю с изменением URL в браузере, либо forward, когда ответ возвращается, но пользователь остается на том же адресе.

## **6**.4 Слой View

Слой View является клиентской частью приложения. Здесь применены технологии Java EE, такие как JSP и JSTL. Они представляют собой шаблонизаторы, которые преобразуют шаблоны страниц в необходимые HTML-страницы, возвращаемые пользователю. Для оформления страниц используется CSS, а для динамических компонентов – JavaScript. С помощью HTML-валидации также происходит проверка данных на frontend части.

# **7 Графический интерфейс**

Графический интерфейс состоит из следующих частей: графический интерфейс веб-приложения, графический интерфейс отчета профилирования.

## **7**.**1** **Графический интерфейс веб-приложения**

Основными элементами веб-приложения являются следующие страницы:

* home page, демонстрируемая пользователю при открытии веб-приложения, рис. 7.1;
* страница заказов, просматриваемая пользователями (свои заказы) или администратором (все заказы), показанная на рисунке 7.2;
* страница оставления заказа, изображенная на рисунке 7.3.

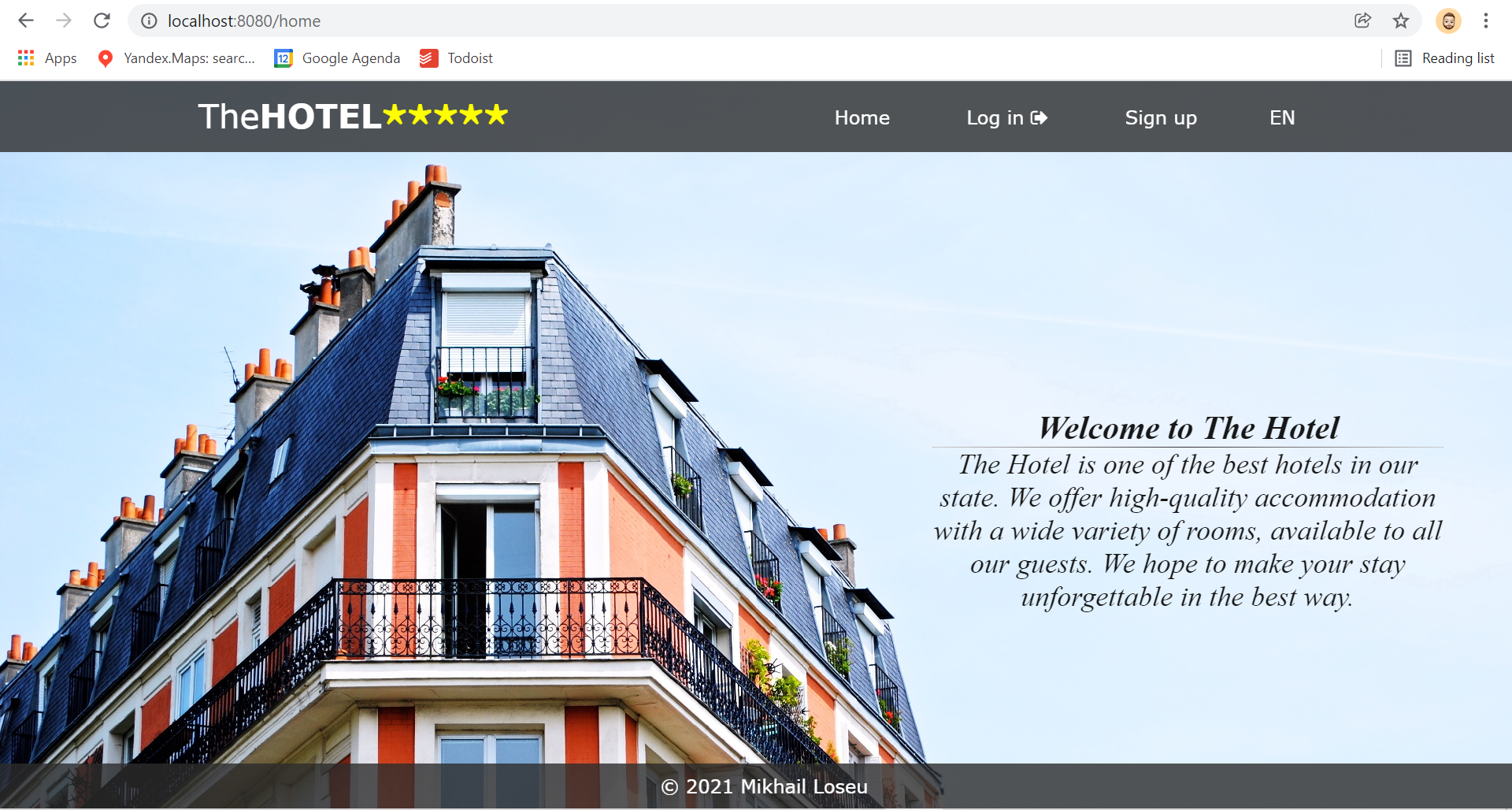
****

Рисунок 7.1 – Домашняя страница приложения

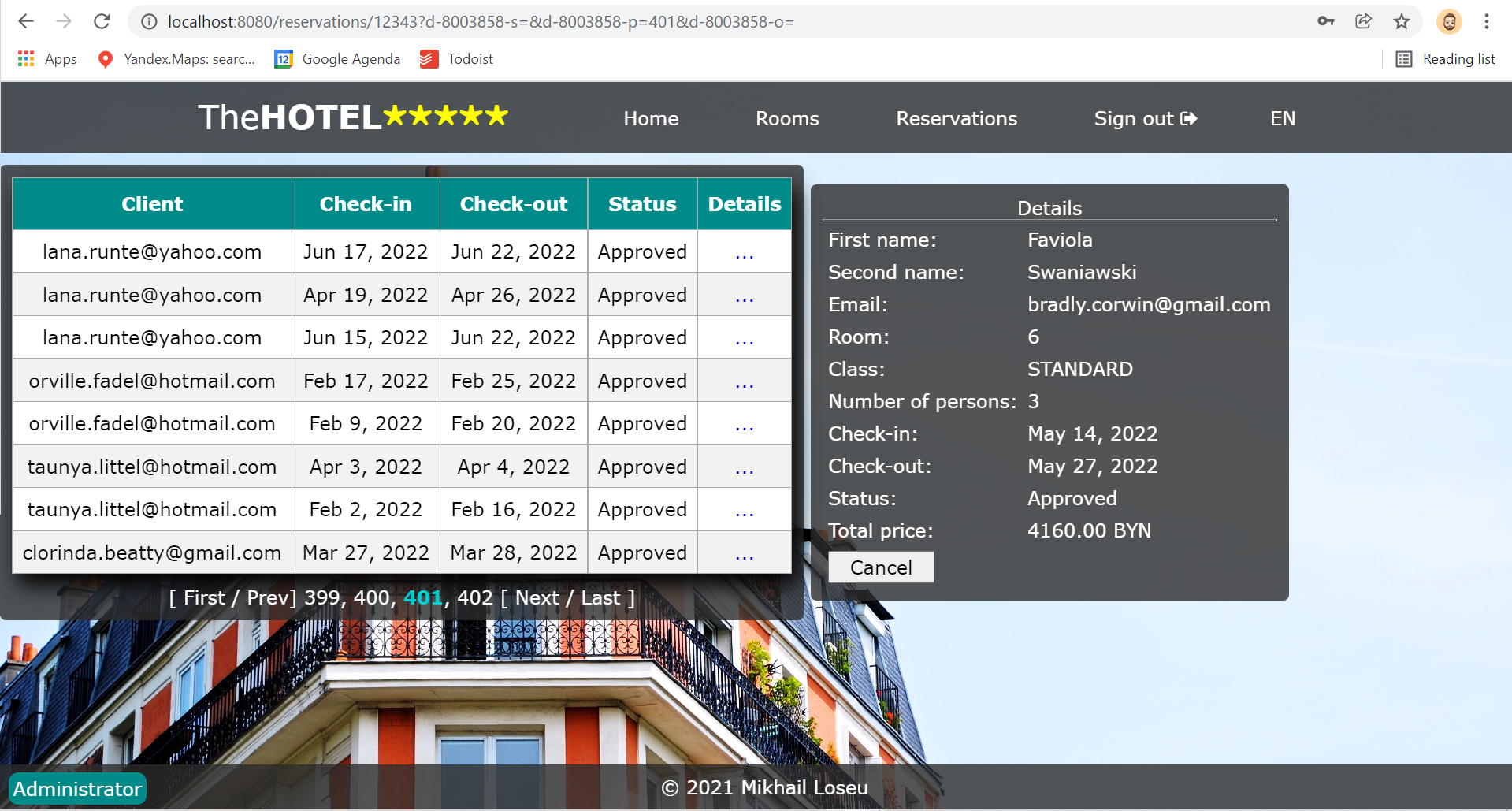


Рисунок 7.2 – Страница заказов (администратор)

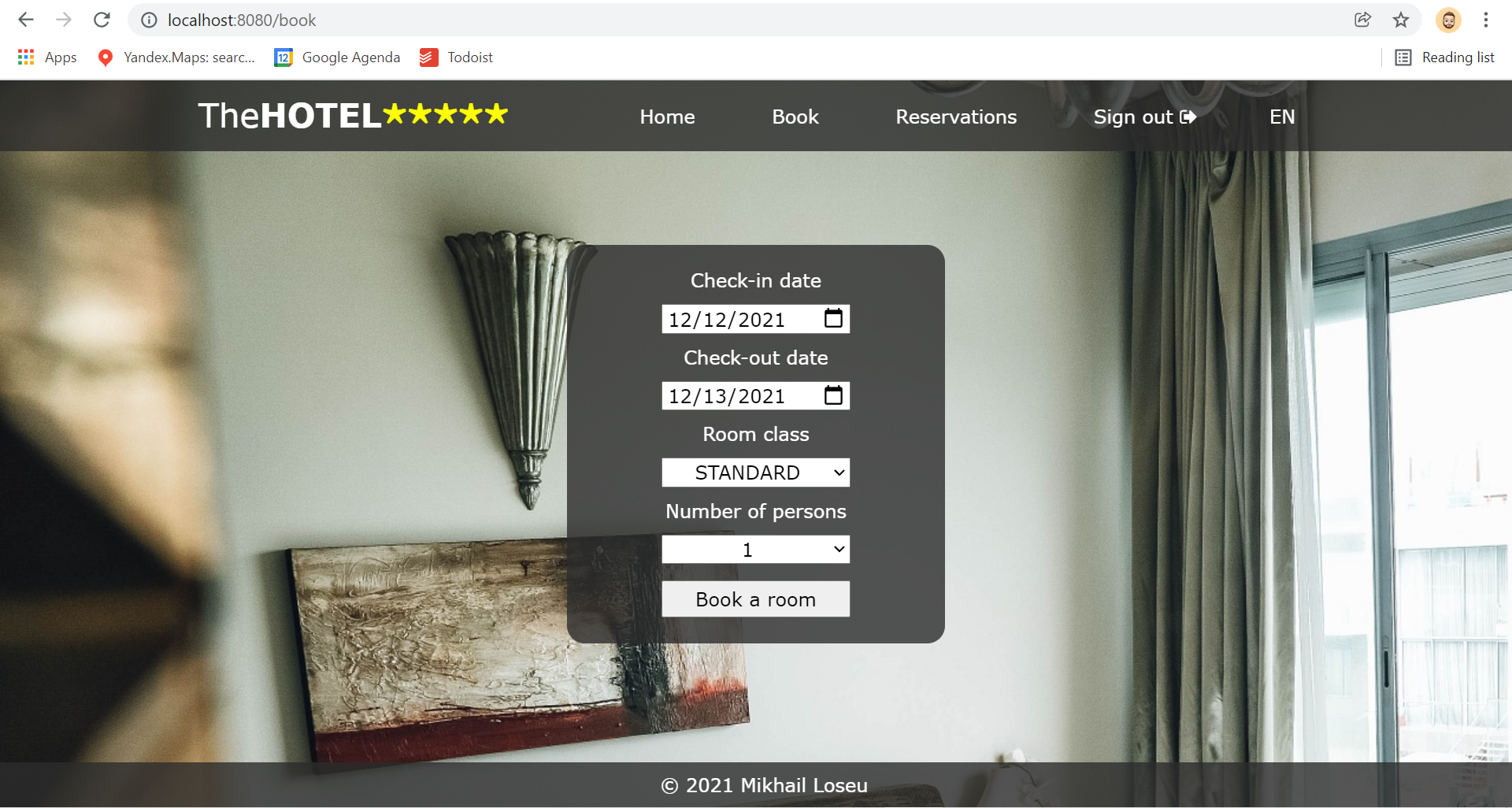


Рисунок 7.3 – Страница бронирования (пользователь)

## **7**.**2** **Графический интерфейс отчета профилирования**

На рисунке 7.4 представлен HTML файл отчета, получаемый в результате профилирования участка findAllReservations().

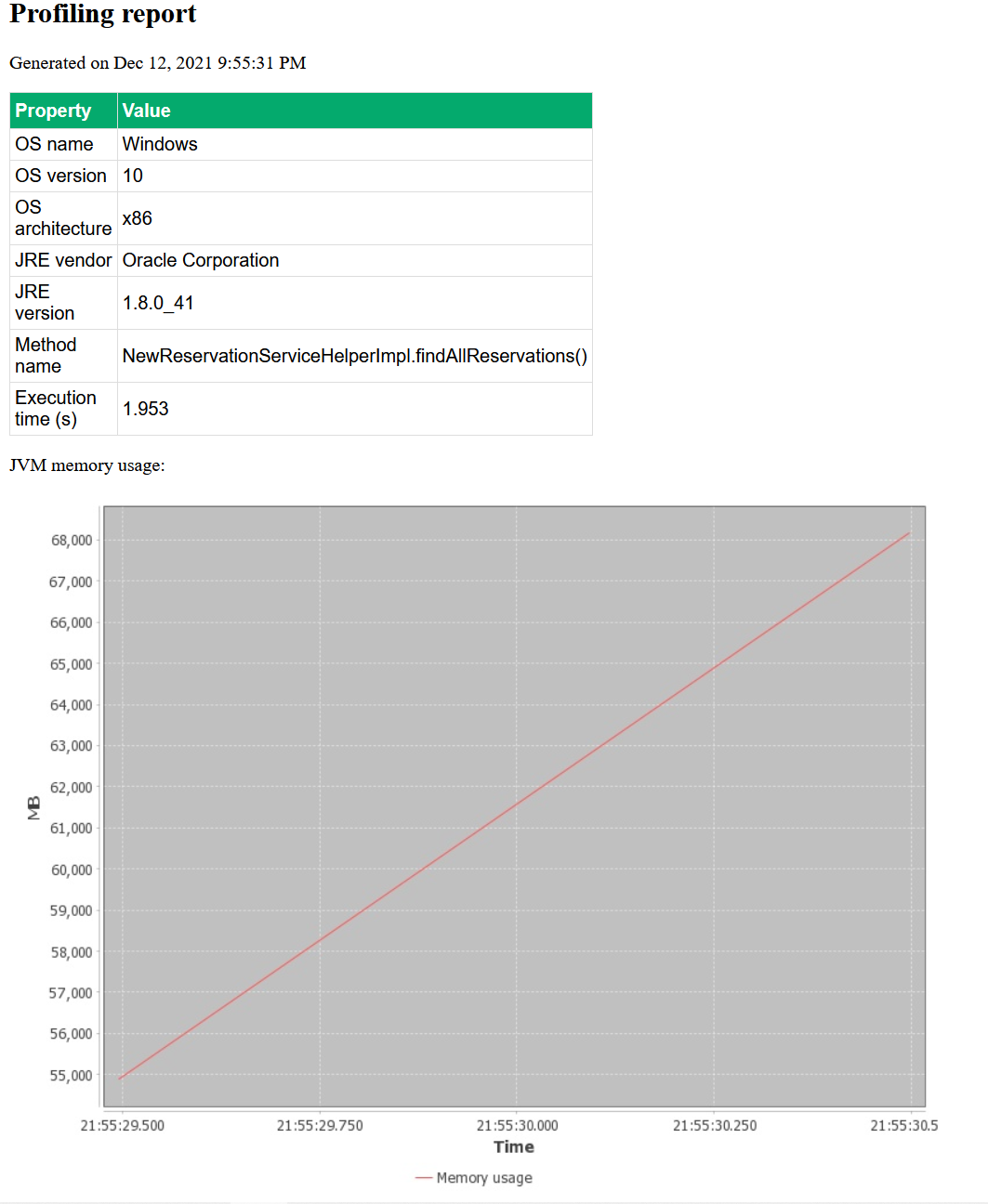
****

Рисунок 7.4 – Пример сгенерированного отчета профилирования

# **8 Тестовый пример. Тестирование программы**

Для тестирования были сгенерированы следующие данные:

1003 уникальных пользователей системы и 10000 заказов, которые были сохранены в базу данных. С помощью следующего запроса было проверено количество записанных данных, результаты которого представлены на рисунке 8.1:

SELECT 'User count', *COUNT*(*\**) FROM booking\_user  
UNION SELECT 'Reservation count', *COUNT*(*\**) FROM reservation;

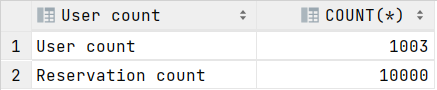


Рисунок 8.1 – Количество записей в системе

Тесты времени выполнения в изначальной реализации ReservationServiceHelperImpl показали время выполнения списка заказов – 19 секунд, отчет о профилировании представлен на рисунке 8.2. Разница между минимальным и максимальным значением использования оперативной памяти составила 16 Мб.

Затем класс реализации был изменен на NewReservationServiceHelperImpl. Время выполения данного участка программы составило менее 2 секунд, что показано на рисунке 8.3. Скачок памяти составил 13Мб.

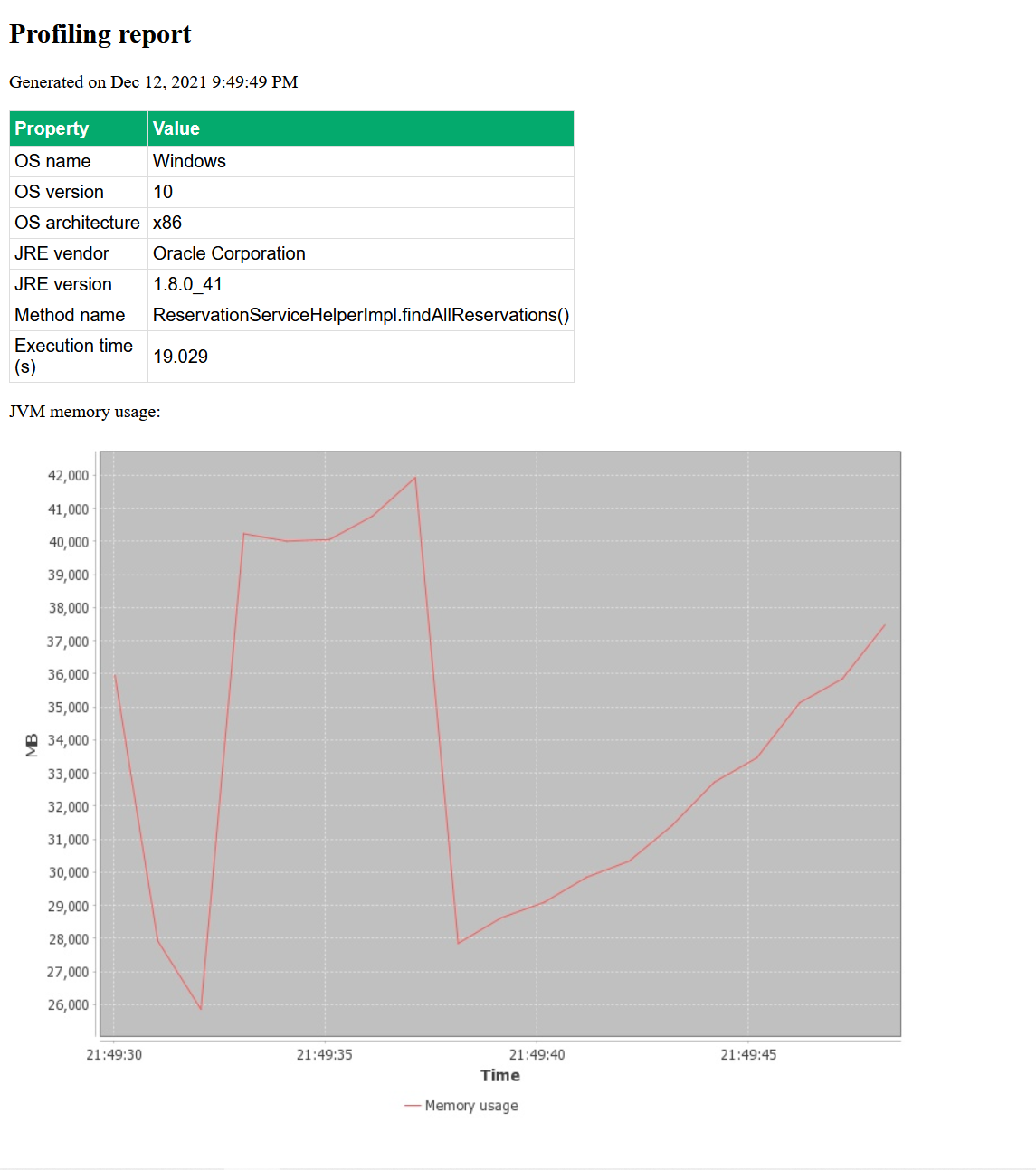


Рисунок 8.2 – Результат профилирования до оптимизации

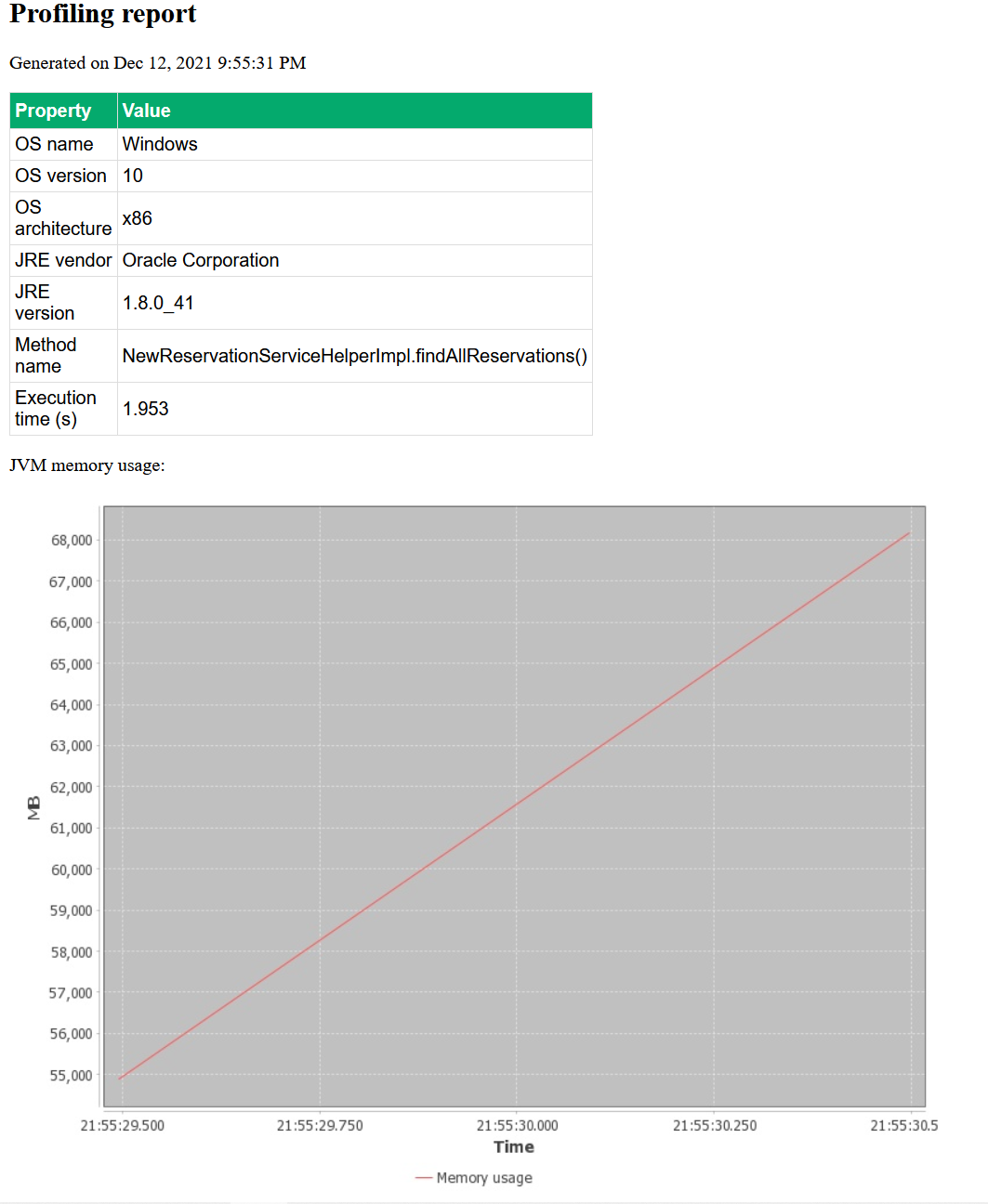


Рисунок 8.3 – Результат профилирования с частичной оптимизацией

Таким образом, избавление от проблемы N+1 значительно повышает скорость работы данного участка приложения.

# **9 Демонстрация работы программы**

Для демонстрации программы количество пользователей было увеличено до 1500, а количество заказов – 13000. На рисунке 9.1 продемонстирована скорость работы участка до оптимизации. На рисунке 9.2 показано время выполнения метода для получения списка заказов после полной оптимизации: избавление от проблемы N+1, а также изменение стратегии загрузки связанных сущностей с Eager на Lazy. Скорость выполнения составила 37.319s. и 0.592s. до и после оптимизации соответсвенно.

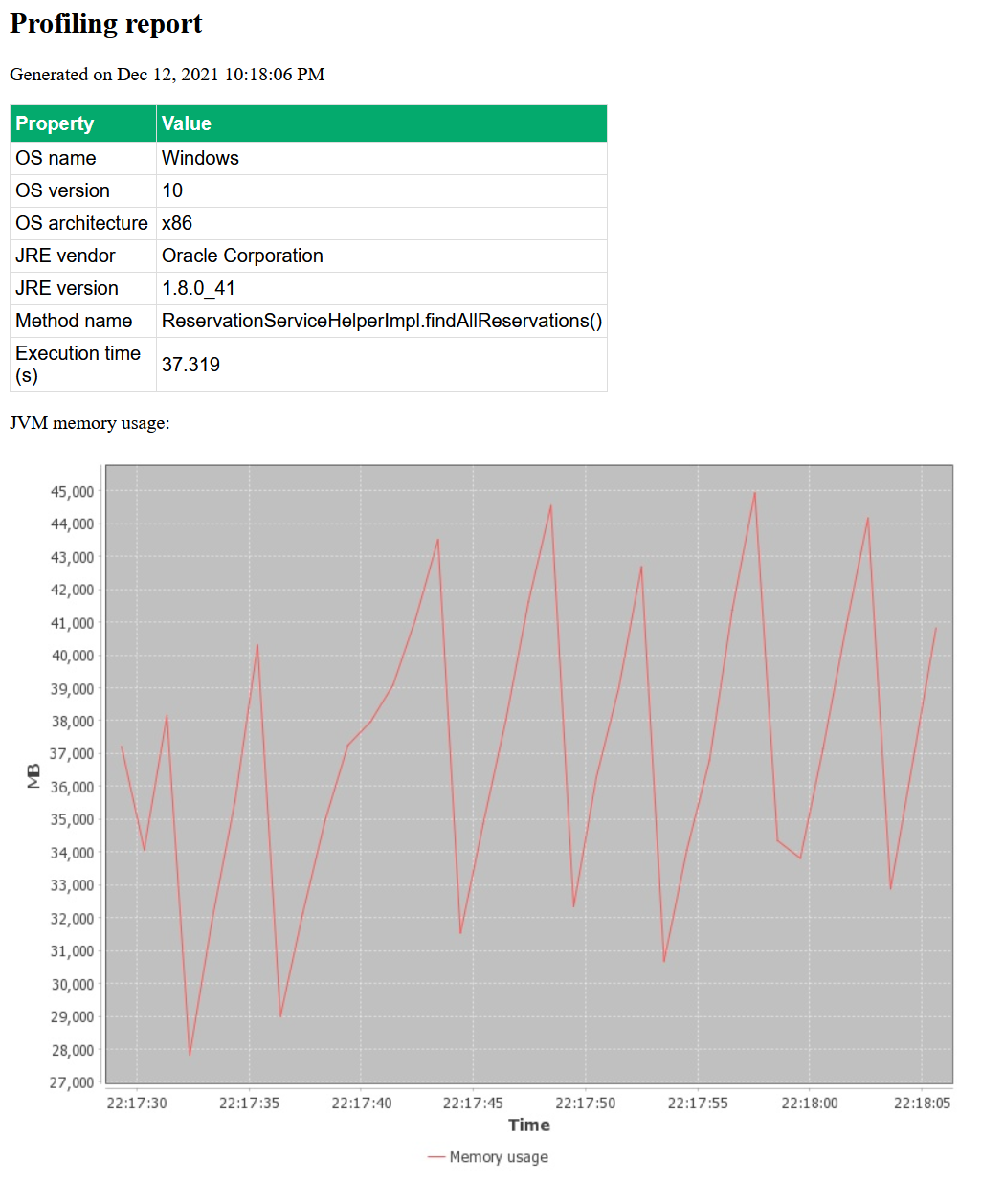


Рисунок 9.1 – Результат профилирования до оптимизации

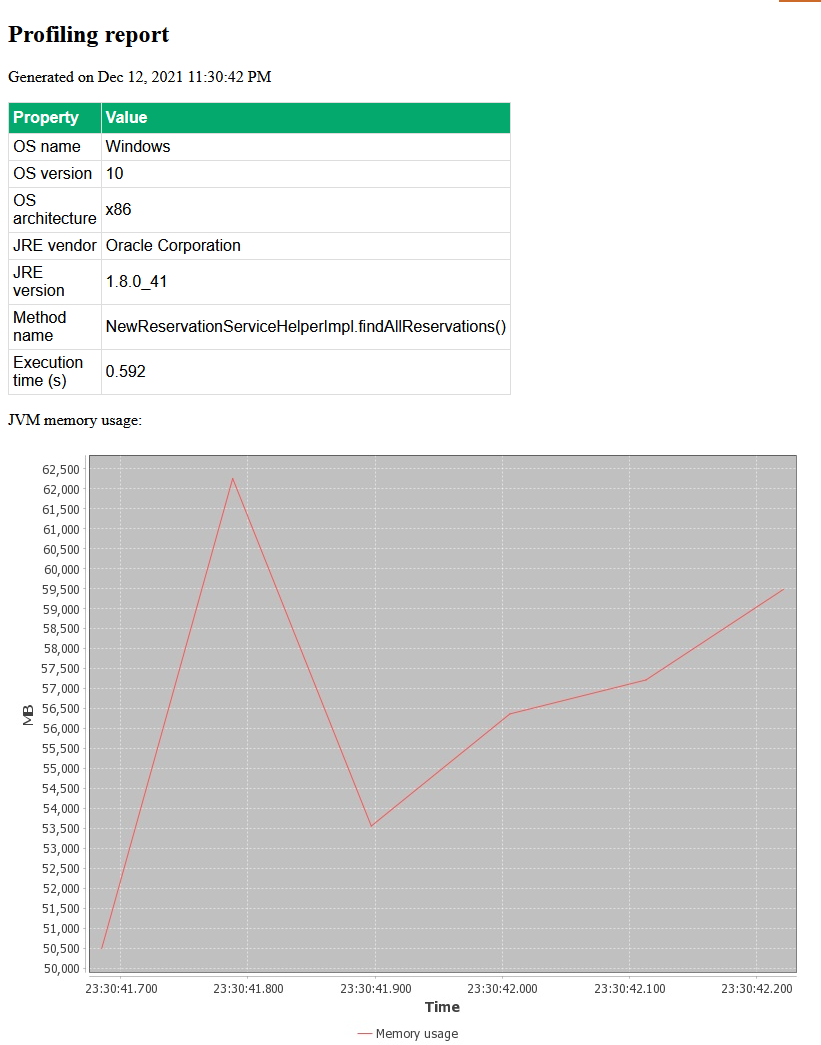


Рисунок 9.2 – Результат профилирования после полной оптимизации

# **ЗАКЛЮЧЕНИЕ**

В данной курсовой работе были использованы следующие средства:

* Язык программирования Java 8.
* Среда разработки IntelliJ IDEA x64.

Было рассмотрены основные понятия аспектно-ориентированного программирования, идентифицирован медленный участок работы приложения. Измерено время и параметры выполнения, а затем произведена оптимизация путем решения проблемы N+1 запросов, а также изменения стратегии получения связанных сущностей с Eager на Lazy.

Были использованы такие технологии и подходы, как аспектно-ориентированное программирование, управление потоками и обращение к стандартной библиотеке для получения информации о системе.

Разработан профилировщик, способный измерять данные о выполнении участков программы, генерировать отчет в формате HTML с построением графиков.

В результате тестирования, профилирования и оптимизации веб-приложение было доработано и готово к использованию, скорость выполнения оптимизируемого участка была увеличена более чем в 63 раза при наличии 14,500 записей в базе данных и составила 0.5с.

# **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. Java. Методы программирования / И. Н. Блинов, В. С. Романчик. – Минск: Четыре четверти, 2013. – 897 с.
2. Проблема с N+1 запросами в JPA и Hibernate [Электронный ресурс] – Режим доступа: <https://habr.com/ru/company/otus/blog/529692/> – Дата доступа 27.11.2021, свободный – Загл. с экрана. – Яз.рус.
3. Профилирование (информатика) [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/Профилирование\_(информатика)– Дата доступа 27.11.2021, свободный – Загл. с экрана. – Яз.рус.

# ПРИЛОЖЕНИЕ А

Листинг исходных кодов

*Файл Profiled.java*

@Retention(RetentionPolicy.*RUNTIME*)  
@Target(ElementType.*METHOD*)  
@Inherited  
public @interface Profiled {  
}

*Файл ProfilingAspect.java*

@Aspect  
@Component  
public class ProfilingAspect {  
  
 @Autowired  
 private ReportService reportService;  
  
 @Value("${profiling.memory.measure.step.millis}")  
 private long memoryMeasureStepMillis;  
  
 private ExecutorService executorService = Executors.*newCachedThreadPool*();  
  
 @Around("@annotation(Profiled)")  
 public Object profileMethod(ProceedingJoinPoint joinPoint) throws Throwable {  
  
 long start = System.*currentTimeMillis*();  
 List<MemoryUsage> memoryUsages = new ArrayList<>();  
 Future<?> punctuation = executorService.submit(() -> punctuateMemoryUsage(memoryUsages));  
 try {  
 return joinPoint.proceed();  
 } finally {  
 long executionTime = System.*currentTimeMillis*() - start;  
 punctuation.cancel(true);  
  
 ExecutionInfo executionInfo = ExecutionInfo.*builder*()  
 .executionTimeMillis(executionTime)  
 .methodName(joinPoint.getSignature().toShortString())  
 .memoryUsageByTime(memoryUsages)  
 .build();  
  
 ProfilingReport report = ProfilingReport.*builder*()  
 .jreInfo(ProfilingUtils.*getJreInfo*())  
 .systemInfo(ProfilingUtils.*getSystemInfo*())  
 .executionInfo(executionInfo)  
 .generationTime(Instant.*now*())  
 .build();  
 reportService.generateReport(report);  
 }  
 }  
  
 @SneakyThrows  
 private void punctuateMemoryUsage(List<MemoryUsage> memoryUsages) {  
 while (true) {  
 Instant now = Instant.*now*();  
 double value = ProfilingUtils.*getMemoryUsageInMb*();  
 MemoryUsage memoryUsage = new MemoryUsage(now, value);  
 memoryUsages.add(memoryUsage);  
 TimeUnit.*MILLISECONDS*.sleep(memoryMeasureStepMillis);  
 }  
 }  
  
}

*Файл ProfilingUtils.java*

public class ProfilingUtils {  
  
 private static final double *MB\_DIVISOR* = 1024.;  
  
 private static final String *OS\_ARCHITECTURE* = "system.arch";  
 private static final String *OS\_NAME* = "system.name";  
 private static final String *OS\_VERSION* = "system.version";  
 private static final String *JAVA\_VENDOR* = "java.vendor";  
 private static final String *JAVA\_VERSION* = "java.version";  
  
 public static double getMemoryUsageInMb() {  
 long totalBytes = Runtime.*getRuntime*().totalMemory();  
 long freeBytes = Runtime.*getRuntime*().freeMemory();  
 long diff = totalBytes - freeBytes;  
 return diff / *MB\_DIVISOR*;  
 }  
  
 @SneakyThrows  
 public static SystemInfo getSystemInfo() {  
 String architecture = System.*getProperty*(*OS\_ARCHITECTURE*);  
 String name = System.*getProperty*(*OS\_NAME*);  
 String version = System.*getProperty*(*OS\_VERSION*);  
 return SystemInfo.*builder*()  
 .osArchitecture(architecture)  
 .osName(name)  
 .osVersion(version)  
 .build();  
 }  
  
 public static JreInfo getJreInfo() {  
 String vendor = System.*getProperty*(*JAVA\_VENDOR*);  
 String version = System.*getProperty*(*JAVA\_VERSION*);  
 return JreInfo.*builder*()  
 .vendor(vendor)  
 .version(version)  
 .build();  
 }  
  
}

*Файл ExecutionInfo.java*

@Data  
@Builder  
public class ExecutionInfo {  
  
 private String methodName;  
  
 private final long executionTimeMillis;  
  
 private List<MemoryUsage> memoryUsageByTime;  
  
}

*Файл JreInfo.java*

@Data  
@Builder  
public class JreInfo {  
  
 private final String vendor;  
  
 private final String version;  
  
}

*Файл MemoryUsage.java*

@Data  
public class MemoryUsage {  
  
 private final Instant instant;  
  
 private final double usageInMb;  
  
}

*Файл SystemInfo.java*

@Data  
@Builder  
public class SystemInfo {  
  
 private final String osArchitecture;  
  
 private final String osName;  
  
 private final String osVersion;  
  
}

*Файл ProfilingReport.java*

@Builder  
@AllArgsConstructor  
@Getter  
public class ProfilingReport {  
  
 private JreInfo jreInfo;  
  
 private SystemInfo systemInfo;  
  
 private ExecutionInfo executionInfo;  
  
 private Instant generationTime;  
  
}

*Файл ReportService.java*

public interface ReportService {  
  
 void generateReport(ProfilingReport report);  
  
}

*Файл ReportServiceImpl.java*

@Service  
public class ReportServiceImpl implements ReportService {  
  
 private static final DateTimeFormatter *TIMESTAMP\_FORMATTER* = DateTimeFormatter.*ofPattern*("yyyy-MM-dd\_HH'h'mm'm'ss's'");  
 private static final DateTimeFormatter *DATE\_FORMATTER* = DateTimeFormatter.*ofLocalizedDateTime*(FormatStyle.*MEDIUM*);  
  
 private static final String *TEMPLATE* = *getTemplate*();  
  
 @Value("${profiling.report.base.folder}")  
 private String reportBaseFolder;  
  
 @SneakyThrows  
 @Override  
 public void generateReport(ProfilingReport report) {  
 String folder = reportBaseFolder + "/rep\_" + fromInstant(report.getGenerationTime()).format(*TIMESTAMP\_FORMATTER*);  
 Files.*createDirectories*(Paths.*get*(folder));  
 saveChart(report, folder);  
 saveHtmlReport(report, folder);  
 }  
  
 private static String getTemplate() {  
 try {  
 ClassLoader loader = Thread.*currentThread*().getContextClassLoader();  
 try (InputStream stream = loader.getResourceAsStream("reportTemplate.html")) {  
 try (InputStreamReader reader = new InputStreamReader(stream)) {  
 return new BufferedReader(reader).lines().collect(Collectors.*joining*(System.*lineSeparator*()));  
 }  
 }  
 } catch (IOException e) {  
 throw new IllegalStateException(e.getMessage(), e);  
 }  
 }  
  
 private void saveChart(ProfilingReport report, String path) throws IOException {  
 List<MemoryUsage> memoryUsageByTime = report.getExecutionInfo().getMemoryUsageByTime();  
 TimeSeries timeSeries = new TimeSeries("Memory usage");  
 memoryUsageByTime.forEach(usage -> timeSeries.add(  
 new Millisecond(Date.*from*(usage.getInstant())),  
 (int) usage.getUsageInMb())  
 );  
 JFreeChart chart = ChartFactory.*createTimeSeriesChart*("", "Time", "MB", new TimeSeriesCollection(timeSeries));  
  
 final File file = new File(path + "/memory\_usage.jpg");  
 ChartUtils.*saveChartAsJPEG*(file, chart, 800, 600);  
 }  
  
 private void saveHtmlReport(ProfilingReport report, String folder) throws IOException {  
 String html = String.*format*(*TEMPLATE*,  
 fromInstant(report.getGenerationTime()).format(*DATE\_FORMATTER*),  
 *// System info*  
report.getSystemInfo().getOsName(),  
 report.getSystemInfo().getOsVersion(),  
 report.getSystemInfo().getOsArchitecture(),  
 *// JRE info*  
report.getJreInfo().getVendor(),  
 report.getJreInfo().getVersion(),  
 *// Execution info*  
report.getExecutionInfo().getMethodName(),  
 report.getExecutionInfo().getExecutionTimeMillis() / 1000.  
 );  
 Files.*write*(Paths.*get*(folder + "/report.html"), html.getBytes(StandardCharsets.*UTF\_8*));  
 }  
  
 private LocalDateTime fromInstant(Instant instant) {  
 return LocalDateTime.*ofInstant*(instant, ZoneId.*systemDefault*());  
 }  
  
}

*Файл TestDataGenerator.java*

@Component  
@ConditionalOnProperty(prefix = "test.data.generation", name = "enabled", havingValue = "true")  
@Slf4j  
public class TestDataGenerator implements ApplicationListener<ApplicationStartedEvent> {  
  
 private static final String *PASSWORD* = DigestUtils.*md5Hex*("12345");  
  
 @Value("${test.data.generation.users}")  
 private int userCount;  
  
 @Value("${test.data.generation.reservations.per.user}")  
 private int reservationsPerUser;  
  
 @Autowired  
 private UserRepository userRepository;  
 @Autowired  
 private ReservationRepository reservationRepository;  
 @Autowired  
 private RoomRepository roomRepository;  
  
 private Faker faker = new Faker();  
 private RandomService random = faker.random();  
  
 private List<Room> rooms;  
  
 @PostConstruct  
 private void init() {  
 rooms = roomRepository.findAll();  
 }  
  
 @Override  
 public void onApplicationEvent(ApplicationStartedEvent event) {  
 for (int i = 0; i < userCount; i++) {  
 try {  
 User user = userRepository.save(generateUser());  
 for (int j = 0; j < reservationsPerUser; j++) {  
 Reservation reservation = generateReservation(user);  
 reservationRepository.save(reservation);  
 }  
 } catch (RuntimeException e) {  
 *log*.error(e.getMessage(), e);  
 }  
 }  
 }  
  
 private User generateUser() {  
 User user = new User();  
 Name name = faker.name();  
 user.setFirstName(name.firstName());  
 user.setSecondName(name.lastName());  
 user.setEmail(faker.internet().emailAddress());  
 user.setPassword(*PASSWORD*);  
 return user;  
 }  
  
 private Reservation generateReservation(User user) {  
 Reservation reservation = new Reservation();  
 reservation.setUser(user);  
 Room room = randomItem(rooms);  
 reservation.setRoom(room);  
 reservation.setPersonsAmount(random.nextInt(1, room.getBedsAmount()));  
  
 ZonedDateTime arrivalDate = ZonedDateTime.*now*().plusDays(random.nextInt(1, 200));  
 ZonedDateTime departureDate = arrivalDate.plusDays(random.nextInt(1, 14));  
  
 reservation.setArrivalDate(Date.*from*(arrivalDate.toInstant()));  
 reservation.setDepartureDate(Date.*from*(departureDate.toInstant()));  
 reservation.setRoomClass(room.getRoomClass());  
 reservation.setReservationStatus(randomItem(Arrays.*asList*(ReservationStatus.*values*())));  
  
 reservation.setTotalPrice(ReservationPriceCalculator.*calculateReservationPrice*(reservation));  
  
 return reservation;  
 }  
  
 private <T> T randomItem(List<T> list) {  
 return list.get(random.nextInt(list.size()));  
 }  
}

*Файл UserRepository.java*

@Repository  
public interface UserRepository extends JpaRepository<User, Integer> {  
  
 Optional<User> findByEmailAndPassword(String email, String password);  
  
 Optional<User> findByEmail(String email);  
  
}

*Файл ReservationRepository.java*

@Repository  
public interface ReservationRepository extends JpaRepository<Reservation, Integer> {  
  
 List<Reservation> findByUserId(Integer id);  
  
}

*Файл ReservationService.java*

public interface ReservationService {  
  
 void loadReservations(Integer id, Model model) throws ServiceException;  
  
 void cancelReservation(Integer id) throws ServiceException;  
  
 void approve(Integer reservationId, Integer roomId) throws ServiceException;  
  
 void pay(Integer id, String cardNumber, String cvv, String validThru) throws ServiceException;  
  
 void checkIn(Integer id) throws ServiceException;  
  
 void checkOut(Integer id) throws ServiceException;  
  
}

*Файл ReservationServiceImpl.java*

@Service  
public class ReservationServiceImpl implements ReservationService {  
  
 private static final String *RESERVATIONS\_ATTRIBUTE* = "reservations";  
 private static final String *RESERVATION\_DETAILS\_ATTRIBUTE* = "reservation\_details";  
 private static final String *SUITABLE\_ROOMS\_ATTRIBUTE* = "rooms";  
  
 private final RoleService roleService;  
 private final ReservationRepository reservationRepository;  
 private final RoomUtils roomUtils;  
 private final RoomRepository roomRepository;  
 private final PaymentValidator paymentValidator;  
 private final ReservationServiceHelper helper;  
  
 public ReservationServiceImpl(RoleService roleService, ReservationRepository reservationRepository, RoomUtils roomUtils, RoomRepository roomRepository, PaymentValidator paymentValidator, ReservationServiceHelper helper) {  
 this.roleService = roleService;  
 this.reservationRepository = reservationRepository;  
 this.roomUtils = roomUtils;  
 this.roomRepository = roomRepository;  
 this.paymentValidator = paymentValidator;  
 this.helper = helper;  
 }  
  
 @Override  
 public void loadReservations(Integer id, Model model) throws ServiceException {  
 User user = roleService.getCurrentUser();  
 loadReservations(model, user);  
 if (shouldShowDetails(id)) {  
 loadDetails(model, id, user);  
 }  
 }  
  
 @Override  
 public void cancelReservation(Integer id) throws ServiceException {  
 checkPermissions(id);  
 reservationRepository.findById(id).ifPresent(reservation -> {  
 reservation.setReservationStatus(ReservationStatus.*CANCELLED*);  
 reservationRepository.save(reservation);  
 });  
 }  
  
 @Override  
 public void approve(Integer reservationId, Integer roomId) throws ServiceException {  
 Reservation reservation = reservationRepository.getById(reservationId);  
 Room room = roomRepository.getById(roomId);  
 reservation.setRoom(room);  
 reservation.setReservationStatus(ReservationStatus.*APPROVED*);  
 reservation.setTotalPrice(ReservationPriceCalculator.*calculateReservationPrice*(reservation));  
 reservationRepository.save(reservation);  
 }  
  
 @Override  
 public void pay(Integer id, String cardNumber, String cvvNumber, String validThru) throws ServiceException {  
 if (!paymentValidator.isCardNumberValid(cardNumber)) {  
 throw new ServiceException("Invalid card number: " + cardNumber);  
 }  
 if (!paymentValidator.isExpirationDateValid(validThru)) {  
 throw new ServiceException("Invalid expiration date: " + validThru);  
 }  
 if (!paymentValidator.isCvvNumberValid(cvvNumber)) {  
 throw new ServiceException("Invalid cvv number: " + cvvNumber);  
 }  
 Reservation reservation = reservationRepository.getById(id);  
 reservation.setReservationStatus(ReservationStatus.*PAID*);  
 reservationRepository.save(reservation);  
 }  
  
 @Override  
 public void checkIn(Integer id) throws ServiceException {  
 setStatus(id, ReservationStatus.*CHECKED\_IN*);  
 }  
  
 @Override  
 public void checkOut(Integer id) throws ServiceException {  
 setStatus(id, ReservationStatus.*CHECKED\_OUT*);  
 }  
  
 private void setStatus(Integer id, ReservationStatus status) {  
 Reservation reservation = reservationRepository.getById(id);  
 reservation.setReservationStatus(status);  
 reservationRepository.save(reservation);  
 }  
  
 private void loadReservations(Model model, User user) {  
 List<Reservation> reservations;  
 if (user.isAdmin()) {  
 reservations = helper.findAllReservations();  
 } else {  
 int id = user.getId();  
 reservations = reservationRepository.findByUserId(id);  
 }  
 model.addAttribute(*RESERVATIONS\_ATTRIBUTE*, reservations);  
 }  
  
 private void loadDetails(Model model, Integer id, User user) throws ServiceException {  
 checkPermissions(id);  
 Reservation reservation = reservationRepository.getById(id);  
 ReservationStatus reservationStatus = reservation.getReservationStatus();  
 if (user.isAdmin() && reservationStatus == ReservationStatus.*WAITING*) {  
 BigDecimal totalPrice = ReservationPriceCalculator.*calculateReservationPrice*(reservation);  
 reservation.setTotalPrice(totalPrice);  
 List<Room> rooms = roomRepository.findAll();  
 List<Reservation> reservations = helper.findAllReservations();  
 List<Room> suitableRooms = roomUtils.getAvailableRooms(rooms, reservations, reservation);  
 model.addAttribute(*SUITABLE\_ROOMS\_ATTRIBUTE*, suitableRooms);  
 }  
 model.addAttribute(*RESERVATION\_DETAILS\_ATTRIBUTE*, reservation);  
 }  
  
 private boolean shouldShowDetails(Integer id) {  
 return id != null;  
 }  
  
 private void checkPermissions(Integer reservationId) throws ServiceException {  
 Reservation reservation = reservationRepository.getById(reservationId);  
 User user = roleService.getCurrentUser();;  
 if (!user.isAdmin() && !reservation.getUser().equals(user)) {  
 throw new ServiceException("Not authorized");  
 }  
 }  
  
}

*Файл ReservationServiceHelper.java*

public interface ReservationServiceHelper {  
  
 List<Reservation> findAllReservations();  
  
}

*Файл ReservationServiceHelperImpl.java*

@Component  
*//@Primary*  
public class ReservationServiceHelperImpl implements ReservationServiceHelper {  
  
 private final UserRepository userRepository;  
 private final ReservationRepository reservationRepository;  
  
 public ReservationServiceHelperImpl(UserRepository userRepository, ReservationRepository reservationRepository) {  
 this.userRepository = userRepository;  
 this.reservationRepository = reservationRepository;  
 }  
  
 @Profiled  
 @Override  
 @Transactional  
 public List<Reservation> findAllReservations() {  
 List<User> users = userRepository.findAll();  
 List<Reservation> reservations = new ArrayList<>();  
 for (User user : users) {  
 reservations.addAll(reservationRepository.findByUserId(user.getId()));  
 }  
 return reservations;  
 }  
  
}

*Файл NewReservationServiceHelperImpl.java*

@Component  
@Primary  
public class NewReservationServiceHelperImpl implements ReservationServiceHelper {  
  
 private final ReservationRepository repository;  
  
 public NewReservationServiceHelperImpl(ReservationRepository repository) {  
 this.repository = repository;  
 }  
  
 @Profiled  
 @Override  
 public List<Reservation> findAllReservations() {  
 return repository.findAll();  
 }  
  
}

*Файл ReservationController.java*

@Controller  
public class ReservationController {  
  
 private final ReservationService service;  
  
 public ReservationController(ReservationService service) {  
 this.service = service;  
 }  
  
 @GetMapping({"/reservations", "/reservations/{id}"})  
 @PreAuthorize("@roleService.isUser()")  
 public String showReservationsPage(  
 @PathVariable(name = "id", required = false) Integer id,  
 Model model  
 ) throws ServiceException {  
 service.loadReservations(id, model);  
 return "reservations";  
 }  
  
 @PostMapping("/reservations/{id}/cancel")  
 @PreAuthorize("@roleService.isUser()")  
 public RedirectView cancelReservation(@PathVariable("id") Integer id, HttpServletRequest request) throws ServiceException {  
 service.cancelReservation(id);  
 return redirectToCurrentPage(request);  
 }  
  
 @PostMapping("/reservations/{id}/approve")  
 @PreAuthorize("@roleService.isAdmin()")  
 public RedirectView approveReservation(  
 @PathVariable("id") Integer reservationId,  
 @RequestParam("roomId") Integer roomId,  
 HttpServletRequest request  
 ) throws ServiceException {  
 service.approve(reservationId, roomId);  
 return redirectToCurrentPage(request);  
 }  
  
 @PostMapping("/reservations/{id}/pay")  
 @PreAuthorize("@roleService.isUser()")  
 public RedirectView payForReservation(  
 @PathVariable("id") Integer id,  
 @RequestParam("cardNumber") String cardNumber,  
 @RequestParam("cvvNumber") String cvvNumber,  
 @RequestParam("validThru") String validThru,  
 HttpServletRequest request  
 ) throws ServiceException {  
 service.pay(id, cardNumber, cvvNumber, validThru);  
 return redirectToCurrentPage(request);  
 }  
  
 @PostMapping("/reservations/{id}/check\_in")  
 @PreAuthorize("@roleService.isAdmin()")  
 public RedirectView checkIn(  
 @PathVariable("id") Integer id,  
 HttpServletRequest request  
 ) throws ServiceException {  
 service.checkIn(id);  
 return redirectToCurrentPage(request);  
 }  
  
 @PostMapping("/reservations/{id}/check\_out")  
 @PreAuthorize("@roleService.isAdmin()")  
 public RedirectView checkOut(  
 @PathVariable("id") Integer id,  
 HttpServletRequest request  
 ) throws ServiceException {  
 service.checkOut(id);  
 return redirectToCurrentPage(request);  
 }  
  
 private RedirectView redirectToCurrentPage(HttpServletRequest request) {  
 String page = CurrentPageGetter.*getCurrentPage*(request);  
 return new RedirectView(page);  
 }  
  
}

*Файл User.java*

@Entity  
@Table(name = "booking\_user")  
public class User implements Identifiable, Serializable {  
  
 private static final long *serialVersionUID* = 5113246202178236722L;  
  
 @Column(name = "id")  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Integer id;  
  
 @Column(name = "is\_admin")  
 private boolean admin;  
  
 @Column(name = "email")  
 private String email;  
  
 @Column(name = "user\_password")  
 private String password;  
  
 @Column(name = "first\_name")  
 private String firstName;  
  
 @Column(name = "second\_name")  
 private String secondName;  
  
 public User() {  
  
 }  
 // getters, setters, equals, hashCode  
}

*Файл Reservation.java*

@Entity  
@Table(name = "reservation")  
public class Reservation implements Identifiable, Serializable {  
  
 private static final long *serialVersionUID* = 4646050451695840394L;  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Integer id;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "user\_id")  
 private User user;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "room\_class\_id")  
 private RoomClass roomClass;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 @JoinColumn(name = "room\_id")  
 private Room room;  
  
 @Column(name = "reservation\_status")  
 @Enumerated(EnumType.*STRING*)  
 private ReservationStatus reservationStatus;  
  
 @Column(name = "arrival\_date")  
 private Date arrivalDate;  
  
 @Column(name = "departure\_date")  
 private Date departureDate;  
  
 @Column(name = "persons\_amount")  
 private int personsAmount;  
  
 @Column(name = "total\_price")  
 private BigDecimal totalPrice;  
// getters, setters, equals, hashCode  
}

*Файл Application.java*

@SpringBootApplication  
public class Application {  
 public static void main(String[] args) {  
 SpringApplication.*run*(Application.class, args);  
 }  
}