*Министерство образования и науки Российской Федерации*

*Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования*

*«Владимирский государственный университет*

*имени Александра Григорьевича и Николая Григорьевича Столетовых»*

*Кафедра информационных систем и программной инженерии*

***КУРСОВОЙ ПРОЕКТ***

*Прототип программной системы «GAR»*

*Выполнили: студенты гр. ПРИ-115*

*Лапыгина В. Д., Родионова А. В.,*

*Ханова А. А., Крутова А. С.*

*Принял: ст.преп. Тимофеев А.А.*

*Владимир, 2018*

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 3

1 ПОСТАНОВКА ЗАДАЧИ 4

1.1 Описание предметной области 4

1.2 Основные понятия предметной области 4

1.3 Функциональные требования к системе 6

1.4 Нефункциональные требования к системе 7

1.5 Сравнительный анализ аналогов 8

2 АНАЛИЗ ЗАДАЧИ И ФУНКЦИОНАЛЬНАЯ ДЕКОМПОЗИЦИЯ СИСТЕМЫ 11

2.1 Анализ бизнес-процесса "Достижение цели" 11

2.2 Сценарий взаимодействия пользователя с системой 16

2.3 Авторизация пользователя в системе 17

2.4 Регистрация пользователя в системе 17

2.5 Изменение данных пользователя 17

2.6 Добавление дефолтной цели 18

2.7 Добавление своей цели 18

2.8. Просмотр своих целей 19

2.9.Отметка о прохождении цели 19

2.10. Просмотр рейтинга пользователей 20

2.11. Создание темы на форуме 20

2.12. Создание сообщения на форуме 20

3 СТРУКТУРНАЯ ОРГАНИЗАЦИЯ СИСТЕМЫ 22

3.1 Общие принципы организации системы 22

3.2 Взаимодействие с внешними сервисами/удалённым модулем для оплаты 22

3.3 Организация доступа к данным 23

3.4 Организация бизнес-логики 27

3.5 Организация веб-интерфейса 28

3.6 Взаимодействие компонентов системы для цели 33

3.7 Организация обработки ошибок 35

3.8 Организация управления доступом 36

4 РЕАЛИЗАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ 37

4.1 EJB-компоненты/компоненты бизнес-логики 37

4.2 Веб-компоненты 39

Здесь рассматриваются компоненты, которые были включены в war-модуль приложения. 39

4.3 Физическая структура баз данных 44

4.4 Обеспечение целостности данных 45

4.5 Реализация взаимодействия с внешним сервисом 47

5 ИНСТРУКЦИЯ ПО УСТАНОВКЕ ПРИЛОЖЕНИЯ 49

**5.1 Установка приложения** 49

**5.2 Установка Web-сервиса** 55

6 ОРГАНИЗАЦИЯ УПРАВЛЕНИЯ ПРОЕКТОМ 59

6.1 Общие принципы взаимодействия в команде 59

6.2 Распределение ролей в команде/зоны ответственности 60

6.3 Календарный план работ 60

Рисунок 40. Диаграмма Ганта. 61

7 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ 62

ЗАКЛЮЧЕНИЕ 63

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 64

ПРИЛОЖЕНИЕ А Листинг компонентов системы 65

ПРИЛОЖЕНИЕ Б Скрипт создания базы данных 66

**ВВЕДЕНИЕ**

Каждый человек самосовершенствуется, приобретает новые навыки, развивает уже имеющиеся умения. Было решено создать веб-приложение для того, чтобы помочь людям в этом.

Для того, чтобы добиваться результатов, нужно ставить для себя цели и уверенно идти к их достижению, то есть совершать маленькие шаги, приводящие к положительному итогу. Задумка нашего приложения именно в этом: когда цель слишком объёмная и требует для своего достижения много времени и ресурсов, человеку тяжело перебороть себя и заставить делать что-то полезное, а разбивая большую цель на маленькие подцели, каждый видит, что выполнить задуманное не так уж и сложно.

В наши дни людям особенно сложно найти время для себя и достаточную мотивацию для самосовершенствования. GAR (Goal Action Result – расшифровка аббревиатуры приложения) придуман как раз для решения данной проблемы. Пользователю предоставится возможность создавать или выбирать для себя своеобразный TODO-list, при выполнении мини-задач из которого, он будет получать рейтинговые баллы, позволяющие ему занимать определённую позицию в рейтинге с другими пользователями, и персонажа, который будет развиваться прямо пропорционально с конкретным навыком, в процессе осваивания его пользователем.

**1 ПОСТАНОВКА ЗАДАЧИ**

**1.1 Описание предметной области**

В данном проекте мы создали веб-приложение мотивирующего характера с элементами игры. Его основная задача – помощь людям в достижении целей. Любой, кому не хватает мотивации, может зарегистрироваться, выбрать цель или создать свою и встать на путь её выполнения. Каждой цели соответствует свой персонаж, который развивается по мере выполнения подцелей. Для цели, которую пользователь создаёт сам, он может выбрать персонажа: бесплатного или более интересного за деньги. Если у пользователя возникают какие-либо вопросы, он всегда может зайти на форум и задать их или же посмотреть, нет ли уже ранее созданной интересующей его темы.

Мы решили сделать именно веб-приложение, чтобы каждый желающий мог получить доступ к приложению практически с любого устройства, с любого браузера. Проект состоит из трёх компонентов: веб-сервиса, базы данных и веб-сайта.

**1.2 Основные понятия предметной области**

MIME-тип изображения – стандарт Интернет, является частью протокола HTTP. Задача MIME это идентификация типа содержимого файла по его заголовку.

Аватарка – публичное графическое представление пользователя, созданное самим пользователем.

Автор темы – создатель темы на форуме.

Администратор – пользователь, зарегистрированный в системе осуществляющий управление списком пользователей, а также управление списком целей пользователей. Администратор имеет следующие характеристики: логин и пароль, ФИО, номер телефона, адрес электронной почты.

Бан – статус, который даётся пользователю за непотребное поведение на форуме.

Гость – пользователь, не зарегистрированный или не авторизованный в системе. Гость может авторизоваться или зарегистрироваться и получить дополнительные функции, доступные зарегистрированным пользователям.

Картинка – иллюстрация, фотография или рисунок в формате \*.jpg или \*.png хранящая автаркии пользователей, картинки персонажей и картинки дефолтных целей.

Клиент – пользователь, зарегистрированный в системе, желающий саморазвиваться в системе GAR посредством игры. Клиент имеет все характеристики, описанные у администратора, кроме даты рождения, интересов и многого другого. С основными функциями клиента можно ознакомиться по диаграмме прецедентов представленной …

Персонаж – это вымышленное существо, которое в системе имеет имя и стоимость, а также несколько стадий развития, отображённых картинками, которые изменяются по ходу выполнения цели пользователя.

Рейтинг – числовой или порядковый показатель, отображающий значимость определенного пользователя.

Роль пользователя – тип пользователя, предусмотренный в системе, в нашей их три: администратор и клиент.

Сообщение – информация в текстовой форме, которую пользователь хочет донести до других, сообщение пишется в определённой теме в форуме.

Тема – предмет (суть) какого-либо рассуждения или изложения имеющая дату создания и автора.

Уровень – степень отличия, ранг в какой-либо иерархии; отображает сколько пунктов из цели уже выполнено пользователем.

Форум – последовательный список тем, доступных для наполнения сообщениями.

Цель – осознанный образ предвосхищаемого результата, на достижение которого направлено действие человека. Цель имеет название и описание, а также персонажа, который будет развиваться в ходе ее выполнения. Цели разделяются по доступности: либо это цель, созданная пользователем, и видит ее только пользователь или же это цель общедоступная (дефолтная) для выполнения, то есть ее видят все клиенты системы и могут добавить в свой личный список.

Цель пользователя – это цель, добавленная в список целей пользователя; с момента добавления цели в личный список цель появляется у пользователя и начинает выполняться.

Рисунок 1. Диаграмма классов концептуального уровня.

**1.3 Функциональные требования к системе**

Система должна предоставлять пользователю обещанный функционал, а именно: наполнение личного профиля данными, добавление или создание цели, которую он собирается достигать, возможность отмечать выполненные промежуточные подцели, наблюдение за развивающимся вместе с ним персонажем, возможность общаться с другими пользователями системы, а также отслеживание своей позиции в общем рейтинге пользователей.

***Бизнес требования:***

Веб-приложение для саморазвития, направленное на аудиторию 14+.

***Пользовательские требования:***

* При первом посещении сайта пользователю должна быть доступна главная страница с информацией о приложении GAR и различным функционалом (который становится доступен после регистрации и входа)
* Для авторизованного пользователя доступно: управление личными данными, создание целей или выбор из готовых, возможность просматривать свой текущий прогресс.
* Также пользователь за дополнительную плату может приобрести персонажа.

**1.4 Нефункциональные требования к системе**

***Ограничения:***

* Разработка системы должна быть на языке Java, платформа Java EE.
* Система должна представлять из себя веб-приложение.
* Система должна реализовывать MVC паттерн.
* Для хранения данных используется СУБД MySQL.
* Пользовательский интерфейс должен быть удобным, чтобы каждый пользователь мог интуитивно понять на какую кнопку нажать, чтобы совершить нужное действие.
* Система должна быть надёжной, т. е. должна сохранять работоспособное состояние на протяжении длительного времени, иметь небольшое время реакции.

***Внешние интерфейсы:***

* При покупке персонажа использовать собственный сервис для оплаты (подробнее пункт 3.2)
* Для взаимодействия с веб-сервисом использовался архитектурный стиль REST (Representational state transfer).

**1.5 Сравнительный анализ аналогов**

Мы провели сравнительный анализ аналогов нашего приложения, результаты которого приведены ниже.

***Название:*** Lift

Интерфейс представлен на рисунке 2

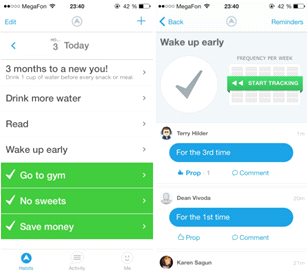


Рисунок 2. Интерфейс Lift.

***Доступность:*** Бесплатное (только английский язык)

***Как проходит процесс достижения цели:*** Пользователь добавляет цель, отмечает выполнение цели галочкой

***Игровая часть:*** Не ярко выражена

***Название:*** SmartProgress

Интерфейс представлен на рисунке 3

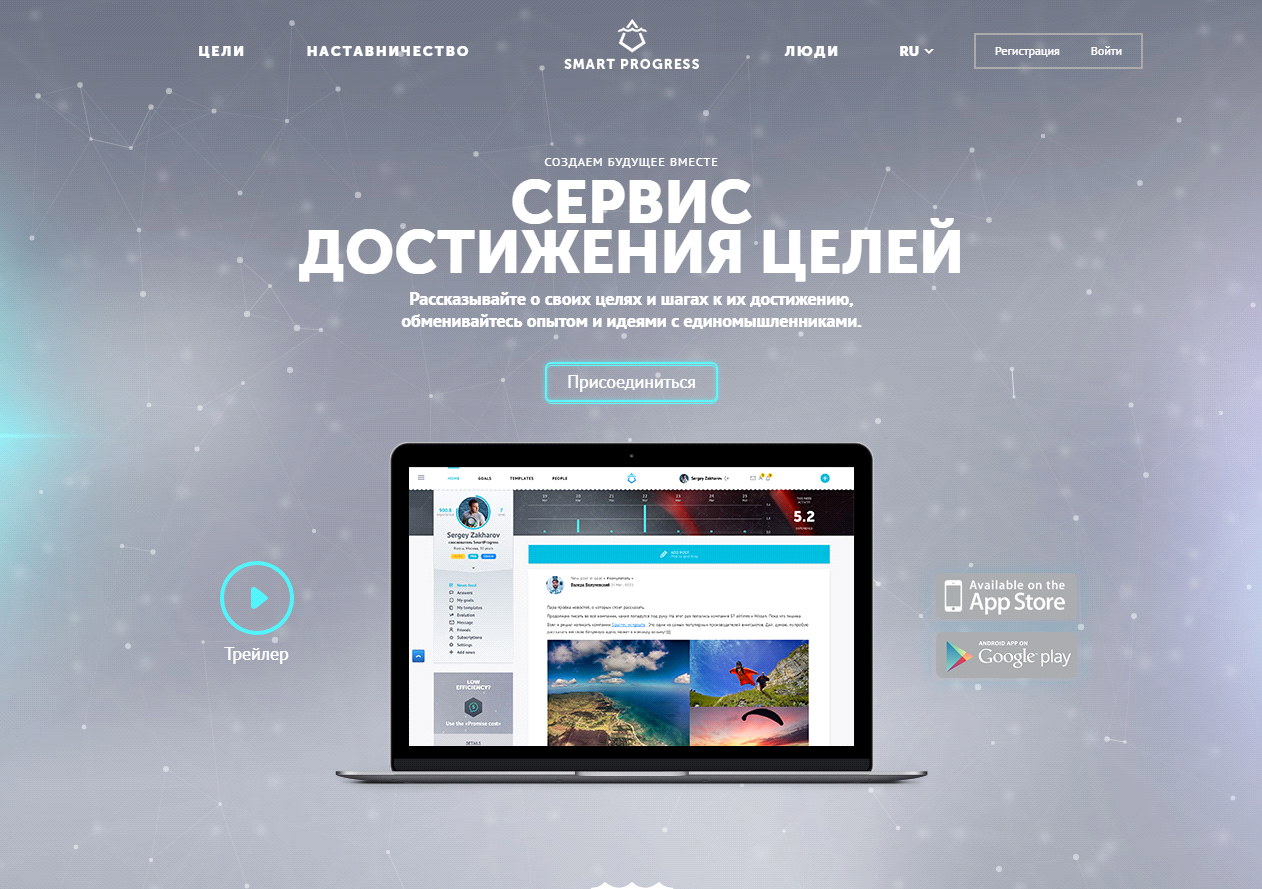


Рисунок 3. Интерфейс SmartProgress.

***Доступность:*** Бесплатное (английский, русский язык)

***Как проходит процесс достижения цели:*** Пользователь добавляет цель, пишет о ней отчеты, обменивается опытом

***Игровая часть:*** Не выражена

***Название:*** Трекер привычек Loop

Интерфейс представлен на рисунке 4

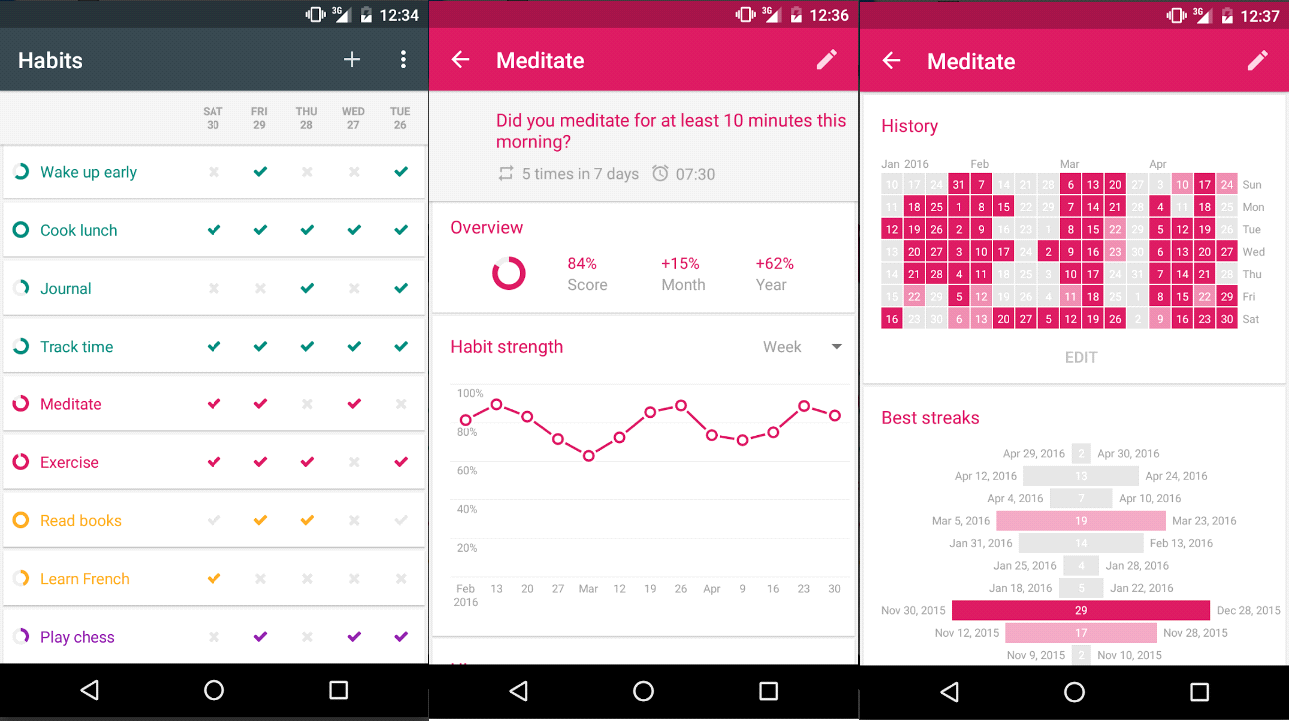


Рисунок 4. Интерфейс Трекера привычек Loop.

***Доступность:*** Бесплатное (английский, русский язык)

***Как проходит процесс достижения цели:*** Пользователь добавляет цель, в достижении цели помогают напоминания, графики прогресса

***Игровая часть:*** Не выражена

***Название:***GAR

Интерфейс представлен ниже в разделе 3.5 Организация интерфейса.

***Доступность:*** Бесплатное (только русский язык)

***Как проходит процесс достижения цели:*** Пользователь добавляет цель, отмечает галочками выполненные подцели, дополнительная мотивация: персонаж, рейтинговая система

***Игровая часть:*** Сильно выражена

**2 АНАЛИЗ ЗАДАЧИ И ФУНКЦИОНАЛЬНАЯ ДЕКОМПОЗИЦИЯ СИСТЕМЫ**

**2.1 Анализ бизнес-процесса "Достижение цели"**

Каждый человек хочет добиться успеха для этого ему нужно мотивировать себя на достижение цели. В повседневной жизни человек ставит конкретные задачи для достижения цели, но ссылаясь на человеческий фактор он либо забывает ли не хватает мотивации. Для этого мы и разработали наше веб-приложение.

Ниже представлена диаграмма прецедентов [4].

Рисунок 5. Диаграмма прецедентов.

Для большего понимания рассмотрим нашу систему в нотации IDEF0 [8]. Приведем два уровня декомпозиции.

Перейдем к диаграмме А-0 бизнес-процесса «Достижение цели» (рис. 6).



Рисунок 6. Диаграмма A-0 бизнес процесса «Достижение цели» в нотации «TO-BE».

Опишем бизнес-процесс «Достижение цели»:

* Вход – материал или информация, которые станут результатом на выходе. На данной диаграмме это «Выбранная цель для достижения».
* Управление – нормативные, регламентирующие и управляющие данные, которыми руководствуется процесс во время его выполнения. На данной диаграмме это «Лимит времени (21 день)» и «Правила пользования веб-приложение GAR».
* Механизм – ресурсы, необходимые для выполнения процесса. На данной диаграмме это «Веб-приложение GAR», «Клиент».
* Выход – информация или материал, которые являются результатом работы. На данной диаграмме это «Итоговый результат достижения цели».

Рассмотрим декомпозицию этого процесса (рис. 7).



Рисунок 7. Декомпозиция процесса «Достижение цели» в нотации «TO-BE».

Как видно из диаграммы, процесс разбился на три подпроцесса «Создать цель» – создание цели для дальнейшего ее выполнения. «Выполнить цель» – сделать отметку о достижении своих результатов. И «Анализ результатов достижения цели» – просмотр рейтинга.

Чтобы подробнее рассмотреть эти подпроцессы, проведем декомпозицию каждого. Начнем с процесса «Создать цель» (рис. 8).



Рисунок 8. Диаграмма A-1 декомпозиция процесса «Создать цель» в нотации «TO-BE».

Процесс разбился на три подпроцесса – «Выбрать цель», «Описать цель», «Выбрать персонажа».

Рассмотрим декомпозицию процесса «Выполнить цель» (рис. 9).



Рисунок 9. Диаграмма A-2 декомпозиция процесса «Выполнить цель» в нотации «TO-BE».

Процесс разбился на три подпроцесса – «Просмотреть цель», «Отметить выполнение», «Построение графика и развитие персонажа».

Мы рассмотрели систему с точки зрения нотации IDEF0. Рассмотрели основные механизмы управления эти процессом. Построили диаграмму осинового бизнес процесса и сделали его декомпозицию.

**2.2 Сценарий взаимодействия пользователя с системой**

Ниже на рисунке 10 представлена диаграмма состояния объекта процесса «Добавление цели» [4].

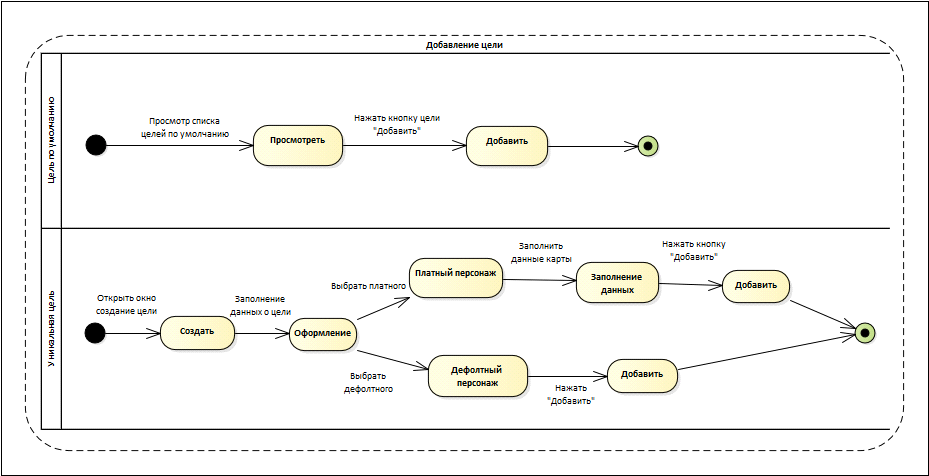


Рисунок 10. Диаграмма состояния объекта процесса «Добавление цели».

На диаграмме представлены несколько способов системы, с помощью которых пользователь может добавить себе цель: добавление дефолтных целей и добавление уникальных целей. Первое подразумевает выбор цели из готового списка целей с уже привязанным к ней персонажем. Второе дает больше возможностей, так как пользователь сам создает цель, описывает ее, а так же выбирает себе персонажа.

Выше бы рассмотрен сценарий одного из ключевых процессов описываемой системы – добавление цели. Далее рассмотрим и ряд других ее возможностей.

**2.3 Авторизация пользователя в системе**

Название: «Войти»

Действующее лицо: «Гость»

Предусловие: Гость должен быть зарегистрирован в системе.

Основной поток:

Пользователь открывает страницу, отображающую форму входа на сайт;

Пользователь заполняет поле логин и поле пароль и авторизуется в системе.

Альтернативный поток: данные заполнены некорректно.

Гость ввёл данные некорректного формата или заполнил не все обязательные поля;

Вывод сообщения об ошибке.

Постусловие: Гость авторизован в системе.

**2.4 Регистрация пользователя в системе**

Название: «Регистрация»

Действующее лицо: «Гость»

Основной поток:

1. Гость открывает страницу, отображающую форму регистрации;

2. Гость заполняет все необходимые данные и регистрируется в системе.

Альтернативный поток: данные заполнены некорректно.

1. Гость ввёл данные некорректного формата или заполнил не все обязательные поля;

2. Вывод сообщения об ошибке.

Постусловие: Пользователь зарегистрирован в системе.

**2.5 Изменение данных пользователя**

Название: «Изменить данные»

Действующее лицо: «Клиент»

Предусловие: Пользователь должен быть зарегистрирован и авторизован в системе.

Основной поток:

1. Клиент открывает модальное окно, отображающего форму изменения данных;

2. Клиент изменят необходимые поля и нажимают кнопку сохранить.

Альтернативный поток: данные заполнены некорректно.

1. Клиент ввёл данные некорректного формата или заполнил не все обязательные поля;

2. Вывод сообщения об ошибке.

Постусловие: Данные изменены в системе.

**2.6 Добавление дефолтной цели**

Название: «Добавить цель»

Действующее лицо: «Клиент»

Предусловие: Пользователь должен быть зарегистрирован и авторизован в системе.

Основной поток:

1. Клиент открывает страницу, с доступными целями для добавления.

2. Клиент выбирает подходящую цели и нажимает кнопку добавить.

Альтернативный поток: -

Постусловие: Цель добавлена.

**2.7 Добавление своей цели**

Название: «Создать цель»

Действующее лицо: «Клиент»

Предусловие: Пользователь должен быть зарегистрирован и авторизован в системе.

Основной поток:

1. Клиент открывает страницу, с доступными целями для добавления. Если он не нашел подходящую цель, то может создать свою нажав кнопку создать цель.

2. Клиент заполняет все необходимые поля и нажимает кнопку добавить.

Альтернативный поток: данные заполнены некорректно.

1. Клиент ввёл данные некорректного формата или заполнил не все обязательные поля;

2. Вывод сообщения об ошибке.

Постусловие: Цель добавлена.

**2.8. Просмотр своих целей**

Название: «Просмотр добавленных целей»

Действующее лицо: «Клиент»

Предусловие: Пользователь должен быть зарегистрирован и авторизован в системе.

Основной поток:

1. Клиент открывает страницу, с добавленными целями.

Альтернативный поток: -

Постусловие: Цели просмотрены.

**2.9.Отметка о прохождении цели**

Название: «Ежедневное выполнение цели»

Действующее лицо: «Клиент»

Предусловие: Пользователь должен быть зарегистрирован и авторизован в системе.

Основной поток:

1. Клиент открывает страницу, отображающую форму выполнения цели;

2. Клиент нажимает кнопку выполнить цель на сегодня, данные обновляются, и кнопка становить недоступной.

Альтернативный поток: -

Постусловие: Ежедневная цель выполнена.

**2.10. Просмотр рейтинга пользователей**

Название: «Просмотреть рейтинг»

Действующее лицо: «Клиент»

Предусловие: Пользователь должен быть зарегистрирован и авторизован в системе.

Основной поток:

1. Клиент открывает страницу, отображающую список рейтинга;

2. Пользователь просматривает список рейтинга.

Альтернативный поток: -

Постусловие: Клиент просмотрел список рейтинга.

**2.11. Создание темы на форуме**

Название: «Добавление темы»

Действующее лицо: «Клиент»

Предусловие: Пользователь должен быть зарегистрирован и авторизован в системе.

Основной поток:

1. Клиент открывает страницу форум, и нажимает добавить тему;

2. Клиент вводи название темы и нажимает кнопку добавить.

Альтернативный поток: данные заполнены некорректно.

1. Клиент ввёл данные некорректного формата или заполнил не все обязательные поля;

2. Вывод сообщения об ошибке.

Постусловие: Тема добавлена.

**2.12. Создание сообщения на форуме**

Название: «Написать сообщение»

Действующее лицо: «Клиент»

Предусловие: Пользователь должен быть зарегистрирован и авторизован в системе.

Основной поток:

1. Клиент открывает страницу, отображающую форум отправки сообщения на форум и нажимает добавить тему;

2. Клиент заполняет поле ввод сообщения и нажимает кнопку отправить.

Альтернативный поток: пользователь заблокирован.

1. Клиент заблокирован для отправки сообщения на форум;

2. Вывод сообщения об ошибке.

Постусловие: Сообщение отправлено.

**3 СТРУКТУРНАЯ ОРГАНИЗАЦИЯ СИСТЕМЫ**

**3.1 Общие принципы организации системы**

Система включает в себя веб-приложение GAR и веб-сервис для осуществления платежей. Система поддерживает две роли: администратор (admin) и клиент (client). Клиент является активным пользователем приложения, именно ему доступен основной функционал системы. Так как монетизация системы осуществляется с помощью веб-сервиса, то взаимодействие с ним доступно клиенту при покупке персонажа.

Здесь еще общая схема организации системы

Рисунок 11. Общая схема организации системы.

Таким образом, наша система состоит из веб-приложения и веб-сервиса. Далее рассмотрим их взаимодействие.

**3.2 Взаимодействие с внешними сервисами/удалённым модулем для оплаты**

В разрабатываемом нами приложении осуществляется взаимодействие с сервисом оплаты как говорилось в пункте выше. Для этого в рамках выполнения курсового проектирования заранее нами был разработан не хитрый REST сервис в среде IntelliJ IDEA 2017.3.4. В данном пункте кратко расскажем о разработке данного сервиса.

Для того чтобы ускорить процесс написания сервиса нами было принято решение реализовать данный Maven Project при помощи Spring Boot [7]. С помощью: <http://start.spring.io/> мы быстро добавили все необходимые нам зависимости и создали пустой рабочий проект Maven на Spring Boot (рис. 12)

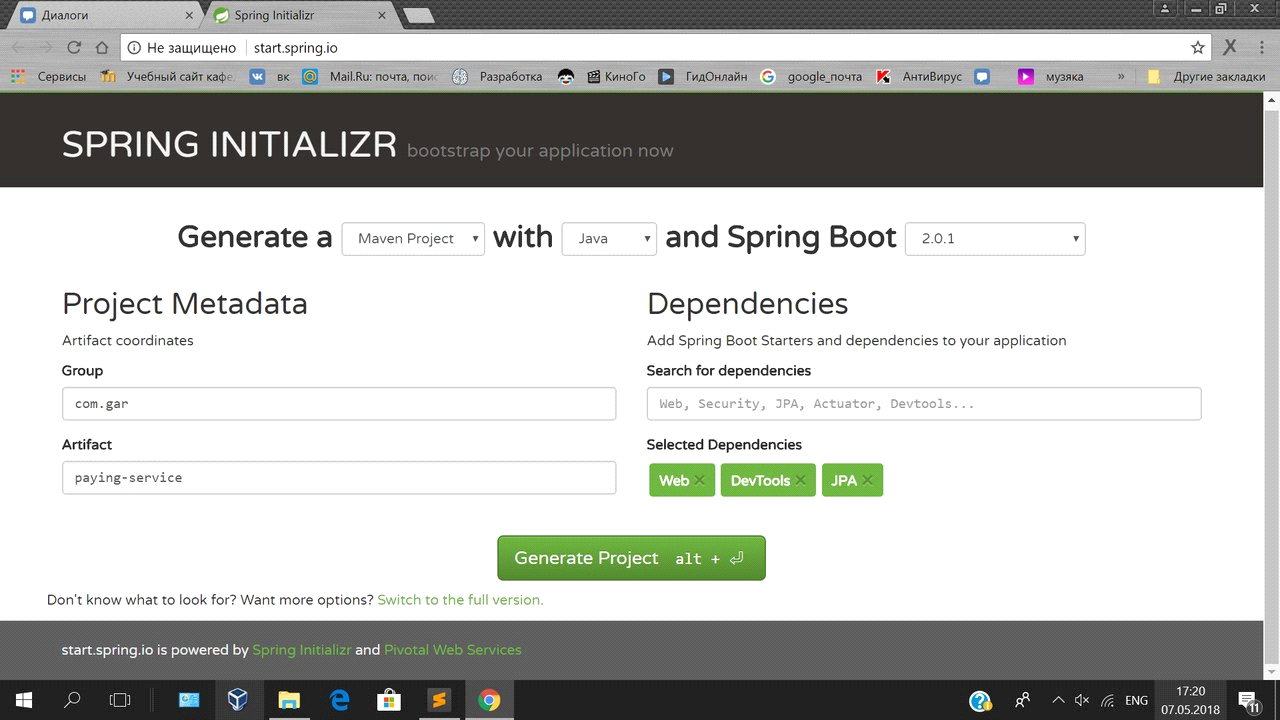


Рисунок 12. Создание Maven Project.

По данному пустому проекте мы буквально за пару часов реализовали наш сервис. Исходные коды представлены в приложении …, а ознакомиться с структурой вы можете на рисунке 13

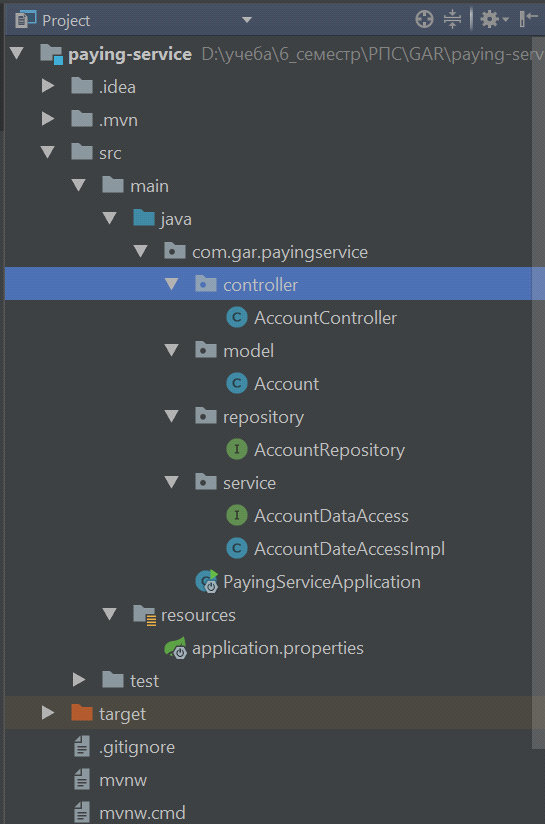


Рисунок 13. Структура проекта.

Вот так быстро нами был заработан простенький веб-сервис с методом оплаты.

**3.3 Организация доступа к данным**

Диаграмма классов уровня доступа к данным представлена на рисунке 14 [4].



Рисунок 14. Диаграмма доступа к данным.

На данной диаграмме представлены все, содержащиеся в системе классы Façade, локальные интерфейсы, которые они реализуют, а также абстрактный класс AbstractFacade, который эти классы наследуют. Так как при создании проекта была использована технология EJB, доступ к данным организован в специфичной для этой технологии форме (более подробно пункт 5.1) [2].

На рисунке ниже (рис. 15) представлена диаграмма классов сущностей [4]. На ней подробно видно, какие классы сущностей представлены в системе, их атрибуты, методы, отношения между классами.



Рисунок 15. Диаграмма классов-сущностей.

Диаграммы, представленные выше, дают четкое представление об организации доступа к данным в разработанном приложении. Далее стоит рассмотреть организацию бизнес-логики проекта.

**3.4 Организация бизнес-логики**

Диаграмма классов уровня бизнес-логики представлена на рисунках далее (рис. 16-17) [4].



Рисунок 16. Диаграмма уровня бизнес-логики. Часть 1.



Рисунок 17. Диаграмма уровня бизнес-логики. Часть 2.

Диаграмма разделена на несколько частей с целью улучшения ее понимания. На ней можно увидеть реализацию правил и ограничений, по которым работает описываемая система, иначе говоря – это методы управляемых бинов приложения.

**3.5 Организация веб-интерфейса**

На рисунках 18-19 представлены схемы перехода между страницами для двух ролей: клиента и администратора. На этих схемах можно подробно увидеть полную навигацию веб-приложения, условия перехода между страницами, а также наличие страниц ошибок, которые предусмотрены при возникновении исключительных ситуаций. На рисунке 18 представлена схема перехода между страницами для пользователя с ролью клиент.



Рисунок 18. Схема перехода между страницами для клиента.

Жирным выделены страницы, доступные с главной панели навигации. Также подразумевается, что на каждой странице может возникнуть исключительная ситуация, в связи с чем будет вызвана страница ошибки. Часть схемы со страницей ошибки вынесена отдельно, чтобы не загромождать схему.

На рисунке 19 представлена схема перехода между страницами для пользователя с ролью администратор.



Рисунок 19. Схема перехода между страницами для администратора.

Аналогично, как и для клиента, жирным выделены страницы, доступные с главной панели навигации, и также подразумевается наличие исключительных ситуаций, в связи с чем будет вызвана страница ошибки.

Дизайн проекта создавался долгое время, так как было предложено множество проектных решений. Макет главной страницы прототипа веб-приложения представлен на рисунке 20.



Рисунок 20. Макет прототипа веб-приложения.

В ходе проекта дизайн претерпел ряд изменений. Итоговый вариант главной страницы веб-приложения представлен на рисунке 21 [5] [6].

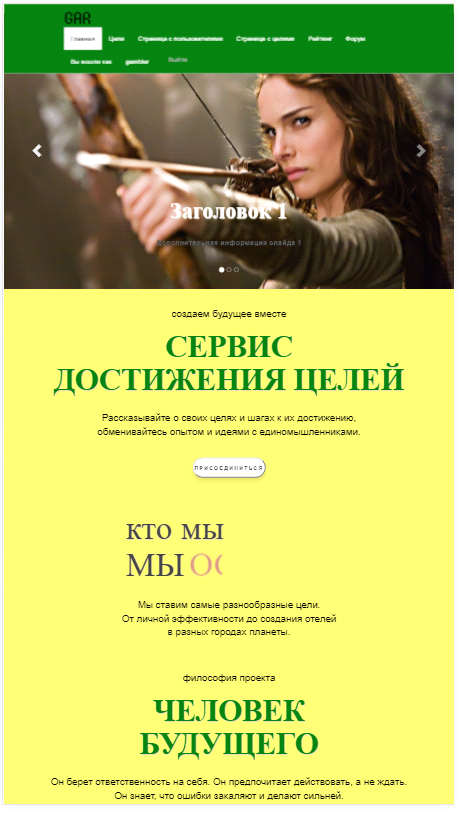


Рисунок 21. Макет веб-приложения.

Навигация по веб-приложению представлена в верхней части окна. Основной контент представлен в центре страницы. Также на каждой странице имеется подвал, содержащий основную контактную информацию и ссылки.

Таким образом, в результате коллективного решения, был разработан окончательный дизайн веб-приложения, организована четкая навигация.

**3.6 Взаимодействие компонентов системы для цели**

Взаимодействие компонентов системы отражено с помощью диаграммы последовательностей [4]. На рисунке 22 представлен процесс добавления цели пользователем. Клиент открывает страницу с целями, нажимает кнопку «Добавить цель», после чего появляется форма создания цели. В этой данные пользователь заполняет данные, а также выбирает персонажа. Если клиент выбирает персонажа по умолчанию, после этого ему нужно нажать кнопку «Добавить цель», и цель будет создана. Если клиент хочет добавить себе платного персонажа, ему нужно выбрать его из выпадающего списка, заполнить реквизиты карты и нажать кнопку «Добавить цель». Об успехе или неудаче совершения данного процесса пользователя информируют всплывающие окна с соответствующими сообщениями.



Рисунок 22. Диаграмма последовательностей для процесса «Добавления цели»

На рисунке 23 представлен процесс удаления цели. Все созданные клиентом цели находятся в его профиле, напротив каждой цели находится кнопка «Удалить “название цели”». При нажатии на эту кнопку всплывает окно подтверждения действия. Если клиент подтверждает действие, цель удаляется, если нет, цель остается.

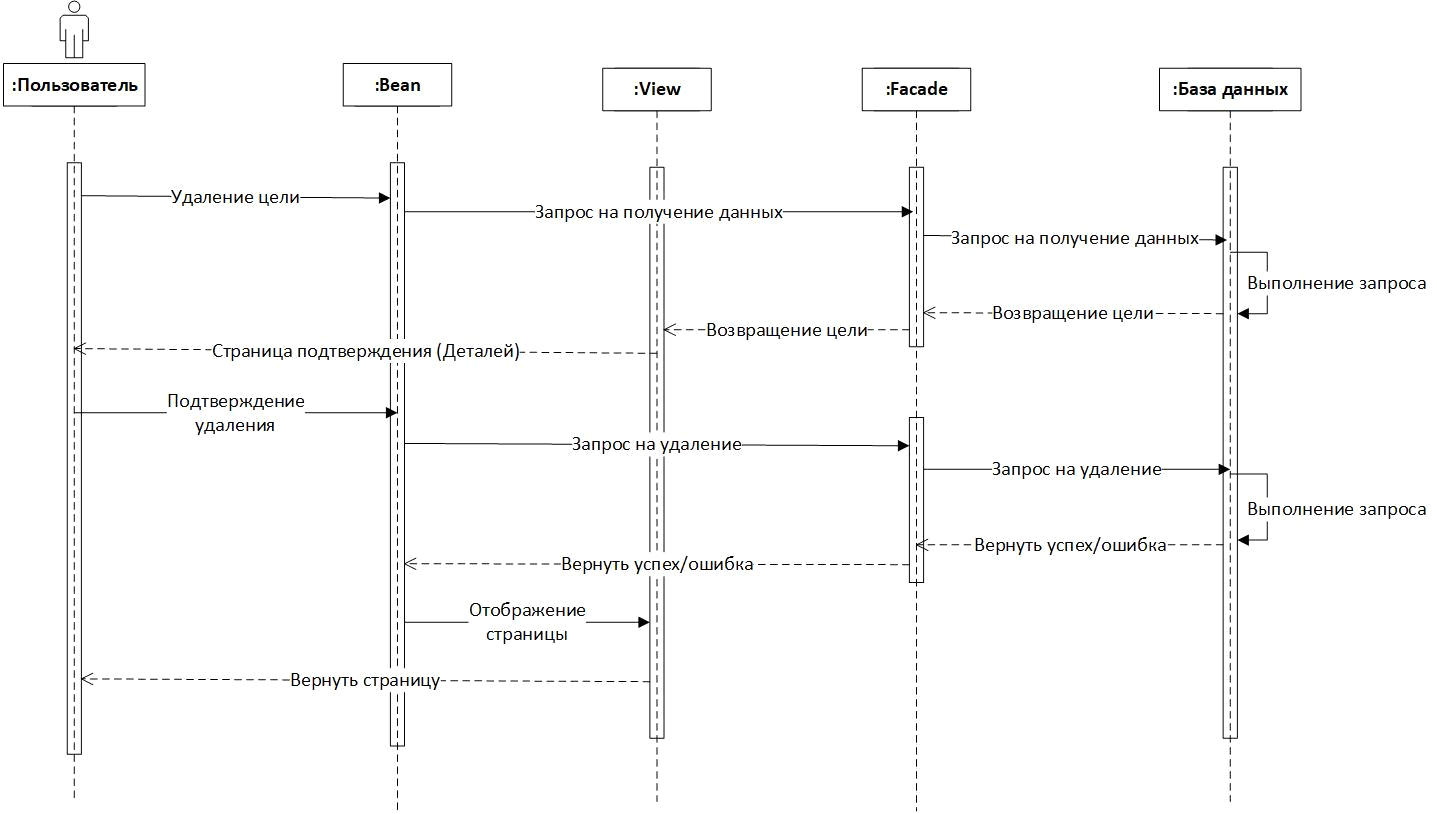


Рисунок 23. Диаграмма последовательностей для процесса «Удаление цели»

Диаграммы, представленные выше, описывают взаимодействие компонентов для одной из главных составляющих единиц системы – цели. На них можно подробно увидеть, как происходит процесс добавления и удаления цели.

**3.7 Организация обработки ошибок**

Для обработки ошибок были было предусмотрено создание страниц для некоторых кодов ошибок и исключений, было реализовано логирование с помощью библиотеки log4j [1]. Созданы следующие страницы ошибок:

* ошибка 404 – не найдено;
* ошибка 500 – ошибка сервера;
* ошибка 512 – слишком длинная формула;
* исключение EJBException;
* страница с сообщением о неправильном вводе логина или пароля.

Таким образом, при возникновении ситуаций, которые описаны выше, будут появляется соответствующие страницы ошибок, а при возникновении ошибок, не входящих в этот перечень, выводиться соответствующее сообщение. Также информация об ошибке будет заносится в файл логирования.

**3.8 Организация управления доступом**

В системе предусмотрено два вида пользователей:

* клиент;
* администратор.

У каждого пользователя есть свои строго определенные права, которые регулируют доступ к функциям системы. Например, клиенту доступны основные операции работы с целью – создание цели, просмотр информации о ней, удаление. Администратор может назначить пользователю с ролью клиент роль администратора, после чего данному пользователю будет доступен функционал администратора.

Итак, в системе поддерживается два вида пользователей. Более подробно список прав, доступный каждому виду пользователей, можно посмотреть на диаграмме прецедентов (рисунок 5).

В пунктах выше была описана организационная структура системы, рассмотрена организация бизнес-логики, доступа к данным, организация веб-интерфейса, описаны принципы обработки ошибок и управления доступом. Данная глава в полной мере описывает структурную организацию системы.

**4 РЕАЛИЗАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ**

**4.1 EJB-компоненты/компоненты бизнес-логики**

Проект разрабатывался на платформе Java EE, в рамках которой использовали технологию EJB, ориентированную на создание серверной бизнес-логики [1]. Подробнее о реализации компонентов системы написано ниже.

**4.1.1 EJB-модуль**

В проекте использовался механизм работы с персистентностью, который предоставляет удобный стандарт для CRUD операций над сущностями, а также поддерживает язык запросов JPQL для поиска и получения данных приложения [2] [3].

Для разработки проекта использовалось обратное проектирование – сначала были созданы классы сущностей, которые потом были отражены в виде таблиц в базе данных. Ниже для примера приведен листинг сущности UserGoal (клиентская цель).

Листинг №. Пример Entity:

@Entity

@Table(name = "goal\_user")

@XmlRootElement

public class GoalUser implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Basic(optional = false)

@Column(name = "ID\_Goal\_user")

private Integer iDGoaluser;

@JoinColumn(name = "ID\_Client", referencedColumnName = "ID\_Client")

@ManyToOne(optional = false)

private Client iDClient;

@JoinColumn(name = "ID\_Goal", referencedColumnName = "ID\_Goal")

@ManyToOne(optional = false)

private Goal iDGoal;

@OneToMany(cascade = CascadeType.ALL, mappedBy = "iDGoaluser")

private Collection<Level> levelCollection;

public GoalUser() {

}

…

Также были сгенерированы сеансовые компоненты для сущностных классов. Пример сеансового компонента GoalUserFacade приведен в листинге №

Листинг №. Пример Façade:

@Stateless

public class GoalUserFacade extends AbstractFacade<GoalUser> implements GoalUserFacadeLocal {

@PersistenceContext(unitName = "GAR-ejbPU")

private EntityManager em;

@Override

protected EntityManager getEntityManager() {

return em;

}

public GoalUserFacade() {

super(GoalUser.class);

}

@Override

public List<GoalUser> findGoalCurrentClient(Integer iDClient){

Query q = em.createNamedQuery("GoalUser.findAllCurrentClient");

q.setParameter("iDClient", iDClient);

return (List<GoalUser>) q.getResultList();

}

}

Каждый сеансовый компонент реализует локальный интерфейс, например, GoalUserFacadeLocal – интерфейс, который реализует сеансовый компонент GoalUserFacade (листинг№).

Листинг №. Пример LocalFaçade:

@Local

public interface GoalUserFacadeLocal {

void create(GoalUser goalUser);

void edit(GoalUser goalUser);

void remove(GoalUser goalUser);

GoalUser find(Object id);

List<GoalUser> findAll();

List<GoalUser> findRange(int[] range);

int count();

public List<GoalUser> findGoalCurrentClient(Integer iDClient);

}

Другие компоненты и сущности из базы данных реализованны аналогично.

Итак, EJB модуль состоит классов сущностей(Entity), сеансовых компонентов для сущностей (Facade) и локальных интерфейсов для сеансовых компонентов (FacadeLocal).

**4.2 Веб-компоненты**

Здесь рассматриваются компоненты, которые были включены в war-модуль приложения.

**4.2.1 Классы управляемых бинов**

В проекте для организации взаимодействия страниц и персистентных объектов были использованы стандартные Java Bean классы. В них прописаны методы, которые обращаются к ресурсам и передают данные на jsf-страницы [2] [3]. Пример такого бина, а именно бин, созданный для взаимодействия с сущностью User (пользователь), представлен в листинге №.

Листинг №. Пример Bean (UserBean):

@ManagedBean(name = "userBean")

@RequestScoped

public class UserBean {

public UserBean() {

}

static final Logger LOGGER = Logger.getLogger(UserBean.class);

public static final String USER\_KEY = "CurrentUser";

@EJB

private UserFacadeLocal userFacade;

private User user = new User();

private UserRole userRole = new UserRole();

private UserRolePK rolePK = new UserRolePK();

private String USER\_ROLE = "client";

@EJB

private UserRoleFacadeLocal userRoleFacadeLocal;

@EJB

private ClientFacadeLocal clientFacadeLocal;

private Client client = new Client();

@EJB

private ImageFacadeLocal facadeLocal;

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public String createUser(){

try{

user.setMessageCollection(null);

user.setTopicCollection(null);

//клиента по id

user.setClient(null);

userFacade.create(user);

rolePK.setLogin(user.getLogin());

rolePK.setRole(USER\_ROLE);

userRole.setUserRolePK(rolePK);

userRole.setUser(user);

userRoleFacadeLocal.create(userRole);

client.setBan(false);

client.setIDUser(user);

client.setGoalUserCollection(null);

client.setiDImage(facadeLocal.find(1));

clientFacadeLocal.create(client);

} catch (EJBException ejbe){

FacesContext.getCurrentInstance().addMessage("regForm:login", new FacesMessage("Пользователь с таким логином уже существует!!!")); LOGGER.error("Ошибка при добавлении пользователя:", ejbe);

}

return "authorization";

}

public User getCurrentUserObject(){

try {

user = userFacade.findLogin(getCurrentUser());

} catch (EJBException ejbe) {

LOGGER.error("Ошибка при получении текущего пользователя:", ejbe);

}

return user;

}

public String getCurrentUser(){

String login = "";

try {

FacesContext fc = FacesContext.getCurrentInstance();

Map<String, Object> params = fc.getExternalContext().getSessionMap();

login = (String) params.get(UserBean.USER\_KEY);

} catch (EJBException ejbe) {

LOGGER.error("Ошибка при получении логина текущего пользователя:", ejbe);

} catch (NumberFormatException nfe){

LOGGER.error("Ошибка при получении логина текущего пользователя. Пришли не верные параметры:", nfe);

}

return login;

}

public String logout() {

HttpSession session = (HttpSession) FacesContext.getCurrentInstance().getExternalContext().getSession(true);

session.invalidate();

return "/index?faces-redirect=true";

}

public boolean isUserLoggedIn() {

String login = getCurrentUser();

boolean result = !((login == null) || login.isEmpty());

return result;

}

}

Таким образом, для создания и манипулирования объектами, а также получения доступа к свойствам страниц jsf были использованы специальные компоненты – Manage Beans.

**4.2.2 JSF-страница authorization.xhtml**

Авторизация пользователя осуществляется на странице authorization.xhtml. Настройка метода входа для приложения выполняется посредством настройки файла web.xml, где мы и указываем страницу для авторизации. Ниже приведена часть кода из файла web.xml, иллюстрирующая настройку страницы для авторизации (листинг №), а также листинг страницы authorization.jsf (листинг №).

Листинг №. Файл web.xml:

<login-config>

<auth-method>FORM</auth-method>

<realm-name>jdbcRealmGAR</realm-name>

<form-login-config>

<form-login-page>/authorization.xhtml</form-login-page>

<form-error-page>/registrationError.html</form-error-page>

</form-login-config>

</login-config>

Листинг №. Страница authorization.xhtml:

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "<http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd>">

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://xmlns.jcp.org/jsf/html">

<h:head>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.3.7/css/bootstrap.min.css"></link>

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/3.3.7/js/bootstrap.min.js"></script>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css"></link>

<link href="css/style-authorization.css" rel="stylesheet"></link>

<title>Авторизация</title>

</h:head>

<h:body>

<div class="container">

<div class="row">

<div class="col-md-offset-3 col-md-6">

<form class="form-horizontal" method="post" action="j\_security\_check">

<span class="heading">АВТОРИЗАЦИЯ</span>

<div class="form-group">

<input type="text" class="form-control" id="inputLogin" placeholder="Login" name="j\_username"></input>

<i class="fa fa-user"></i>

</div>

<div class="form-group help">

<input type="password" class="form-control" id="inputPassword" placeholder="Password" name="j\_password"></input>

<i class="fa fa-lock"></i>

<a href="#" class="fa fa-question-circle"></a>

</div>

<div class="form-group">

<div class="main-checkbox">

<input type="checkbox" value="none" id="checkbox1" name="check"></input>

<label for="checkbox1"></label>

</div>

<span class="text">Запомнить</span>

<input type="submit" value="войти" class="btn btn-default"/>

<a href="/GAR-war/faces/registration.xhtml" class="btn btn-default">Регистрация</a>

</div>

</form>

</div>

</div><!-- /.row -->

</div><!-- /.container -->

</h:body>

</html>

В данном разделе иллюстрируется способ организации доступа к данным веб-приложения, а именно способ осуществления авторизации пользователя.

**4.2.3 JSF-страница registration.xhtml**

Регистрация пользователя осуществляется на странице registration.xhtml. Регистрация выполняется вызовом метода createUser(), который находится в UserBean. Листинг данного метода представлен ниже.

Листинг №. Метод createUser():

public String createUser(){

try{

user.setMessageCollection(null);

user.setTopicCollection(null);

user.setClient(null);

userFacade.create(user);

rolePK.setLogin(user.getLogin());

rolePK.setRole(USER\_ROLE);

userRole.setUserRolePK(rolePK);

userRole.setUser(user);

userRoleFacadeLocal.create(userRole);

client.setBan(false);

client.setIDUser(user);

client.setGoalUserCollection(null);

client.setiDImage(facadeLocal.find(1));

clientFacadeLocal.create(client);

} catch (EJBException ejbe){

FacesContext.getCurrentInstance().addMessage("regForm:login", new FacesMessage("Пользователь с таким логином уже существует!!!"));// не уверена...ооочень не уверена

LOGGER.error("Ошибка при добавлении пользователя:", ejbe);

}

return "authorization";

}

В методе реализована запись в базу данных информации о новом пользователе, а также валидация данных и логирование.

**4.2.4 Конфигурирование JSF-приложения**

Так как конфигурирование Java WEB приложения с помощью файла faces-config.xml [2] накладывает определенные сложности на процесс разработки, требует дополнительного контроля и подвержено ошибкам. В связи с этим было решено делегировать процесс конфигурации инфраструктуре, что и было реализовано с помощью аннотаций к JavaBean компонентам. Например, вместо кода XML-конфигурации для JSF была использована аннотация @ManagedBean для обеспечения управления бинами. Также с помощью аннотаций значительно упрощаются механизмы контроля навигации, управления ресурсами.

Исходя из вышесказанного, для упрощения процесса создания и поддержки приложения лучше всего использовать аннотации, что и было сделано.

**4.3 Физическая структура баз данных**

По сути дела, физическое проектирование базы данных подразумевает конструирование таблиц в СУБД. И как вы уже знаете из функциональных требований нами была выбрана СУБД MySQL. Причина ее выбора лежит на поверхности - это ряд преимуществ: простота использования, гибкость, низкую стоимость владения (относительно платных СУБД), а также масштабируемость и производительность.

В данной СУБД на основании модели классов был написан скрипт создания таблиц, представленный в приложении …. После чего по средствам инструмента для визуального проектирования MySQL Workbench получена физическая модель БД представленная на рисунке 24 и в приложении ….

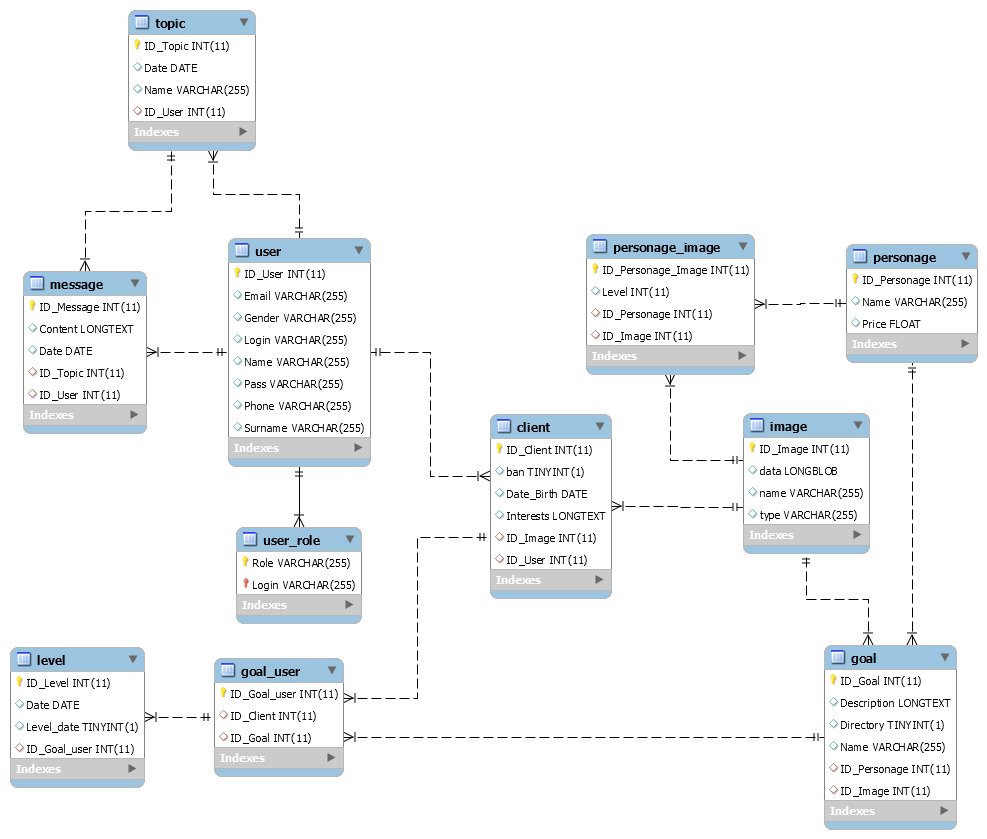


Рисунок 24.Физическая модель БД.

В данном пункте мы разработали физическую модель базы данных необходимую для СУБД MySQL содержащую имена сущностей и полей, а также для каждого атрибута тап данный. Подробнее с проектированием структуры базы банных можно ознакомиться в следующем разделе.

**4.4 Обеспечение целостности данных**

Для начала разберемся что означает целостность данных. Она означает систему правил, используемых для поддержания связей между записями в связанных таблицах, а также для обеспечения защиты от случайного удаления или изменения связанных данных. В нашей ИС достаточно было обеспечить целостность данных на уровне БД для этого мы воспользовались нормализацией.

Нормализация — это процесс организации данных в базе данных, включающий создание таблиц и установление отношений между ними в соответствии с правилами, которые обеспечивают защиту данных и делают базу данных более гибкой, устраняя избыточность и несогласованные зависимости.

Существует несколько правил нормализации баз данных. Каждое правило называется «нормальной формой». Рассмотрим проведение БД к первой, второй и третей нормальной форме.

1. Первая нормальная форма

* Устраняем повторяющиеся группы в отдельных таблицах.
* Создаем отдельную таблицу для каждого набора связанных данных.
* Идентифицируем каждый набор связанных данных с помощью первичного ключа.

2. Вторая нормальная форма

* Создаем отдельные таблицы для наборов значений, относящихся к нескольким записям.
* Связываем эти таблицы с помощью внешнего ключа.

3. Третья нормальная форма

* Устраняем поля, не зависящие от ключа.

Изучив правила нормализации БД мы предварительно при анализе системы, выявили сущности («User», «Client», «Goal», «Personage», «Image»…). После чего устраняя повторяющиеся группы в таблицах создали отдельную таблицу для каждого набора связанных данных («PersonageImage»…). Идентифицировали каждый набор связанных данных с помощью первичного ключа (ID\_User, ID\_Client …). Создали отдельные таблицы для наборов значений, относящихся к нескольким записям («Image»…). Связали эти таблицы с помощью внешнего ключа. После чего устранили поля, не зависящие от ключа («Topic»…).

В результате приведения БД к 3 нормальной форме мы получили структура БД представленную в пункте 4.3.

**4.5 Реализация взаимодействия с внешним сервисом**

В пункте 3.2 мы рассказываем о созданном нами веб-сервисе, а в данном пункте мы разберем как же реализовывается взаимодействие.

Как уже говорилось ранее нами использовался архитектурный стиль REST []. Он был выбран как быстрое и простое решение для создания сервиса.

Взаимодействие с данным сервисом осуществляется с помощь библиотек:

* httpclient-4.5.5.jar
* httpcore-4.4.9.jar

Для взаимодействия был создан метод:

//оплата персонажа

//исключение обробатываем чуть выше

public boolean payment(String holder, String codeCard, String codeSecurity, String expirationDate, String purchaseValue) throws IOException {

boolean result = false;

HttpClient clientHttp = new DefaultHttpClient();

HttpPost post = new HttpPost("http://localhost:8081/account/purchase");

List nameValuePairs = new ArrayList(1);

nameValuePairs.add(new BasicNameValuePair("name", "value")); //you can as many name value pair as you want in the list.

nameValuePairs.add(new BasicNameValuePair("expirationDate", expirationDate));//2018-05-31

nameValuePairs.add(new BasicNameValuePair("holder", holder));//Lapygina Vasilisa

nameValuePairs.add(new BasicNameValuePair("codeSecurity", codeSecurity));//321

nameValuePairs.add(new BasicNameValuePair("codeCard", codeCard));//1232353424

nameValuePairs.add(new BasicNameValuePair("purchaseValue", purchaseValue));//

post.setEntity(new UrlEncodedFormEntity(nameValuePairs));

HttpResponse response = clientHttp.execute(post);

BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));

String line = "";

while ((line = rd.readLine()) != null) {

result = line.equals("true");

}

return result;

}

По средствам данного метода мы отправляем GET запрос на REST сервис и получаем подтверждение если оплата успешно произведена.

В данном пункте мы показали, как в нашем приложении осуществляется взаимодействие с сервисом оплаты.

**5 ИНСТРУКЦИЯ ПО УСТАНОВКЕ ПРИЛОЖЕНИЯ**

В данном разделе представим пошаговую инструкцию по установке нашего приложения, начиная с этапа скачивания исходного кода и заканчивая получением полнофункционального приложения с тестовыми данными.

**5.1 Установка приложения**

Шаг 1: Убедитесь, что у вас установлены обязательные компоненты такие как:

* NetBeans последней версии - [https HYPERLINK "https://netbeans.org/":// HYPERLINK "https://netbeans.org/"netbeans HYPERLINK "https://netbeans.org/". HYPERLINK "https://netbeans.org/"org HYPERLINK "https://netbeans.org/"/](https://netbeans.org/)
* СУБД MySQL (из-за особенностей с кодировкой желательна версия 5.7)
* Проверьте установленный GlassFish Server хотя он и обычно включен в IDE NetBeans. Можете так же выбрать любой удобный вам сервер, но в дальнейшем на нем вам необходимо будет самостоятельно выполнять настройку подключения, а также выполнить настройку домена безопасности.

Шаг 2: Выполните скачивания репозитория архивом (рис. 25):

<https://github.com/nans1996/GAR>

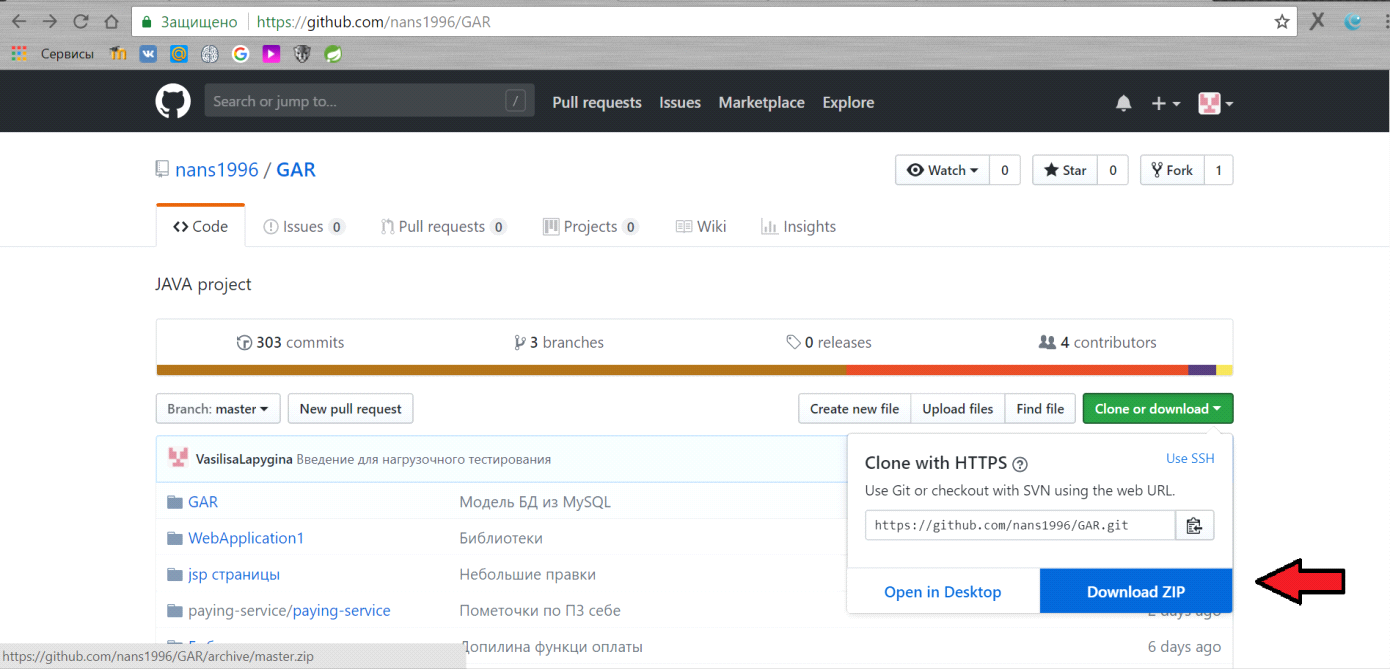


Рисунок 25. Скачивание ZIP архива с проектом.

Шаг 3: Распакуем скачанный архив:

Чтобы извлечь все содержимое сжатой папки, нажмите и удерживайте ее (или щелкните ее правой кнопкой мыши), выберите команду *Извлечь* все, а затем следуйте указаниям.

Шаг 6: Выполним создание базы данных:

* Зайдем в MySQL: mysql –uUSERNAME –pPASSWORD
* Выполним создание БД: create database GAR\_BD CHARACTER SET utf8 COLLATE utf8\_general\_ci;

Шаг 5: Выполним настройку подключения и домена безопасности (рис. 26):

* Откройте командную строку (cmd)
* Перейдите в директорию с установленным GlassFish сервером и в папку bin чаще всего это путь: C:\Program Files\glassfish-4.1.1\bin
* Поочередно выполните команды изменив пароль и пользователя на установленные в MySQL:

asadmin create-jdbc-connection-pool --datasourceclassname com.mysql.jdbc.jdbc2.optional.MysqlDataSource --restype javax.sql.DataSource --property portNumber=3306:password=1997:user=root:serverName=localhost:databaseName=GAR\_BD GARPool

asadmin create-jdbc-resource --connectionpoolid GARPool jdbc/GARperson

asadmin create-auth-realm --classname com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm --property datasource-jndi=jdbc/GARperson:user-table=user:user-name-column=Login:password-column=Pass:group-table=User\_Role:group-name-column=Role:jaas-context=jdbcRealm:digest-algorithm=none jdbcRealmGAR

* Если вы совершили ошибки в командах можете выполнить удаление:

asadmin delete-auth-realm jdbcRealmGAR

asadmin delete-jdbc-resource jdbc/GARperson

asadmin delete-jdbc-connection-pool GARPool

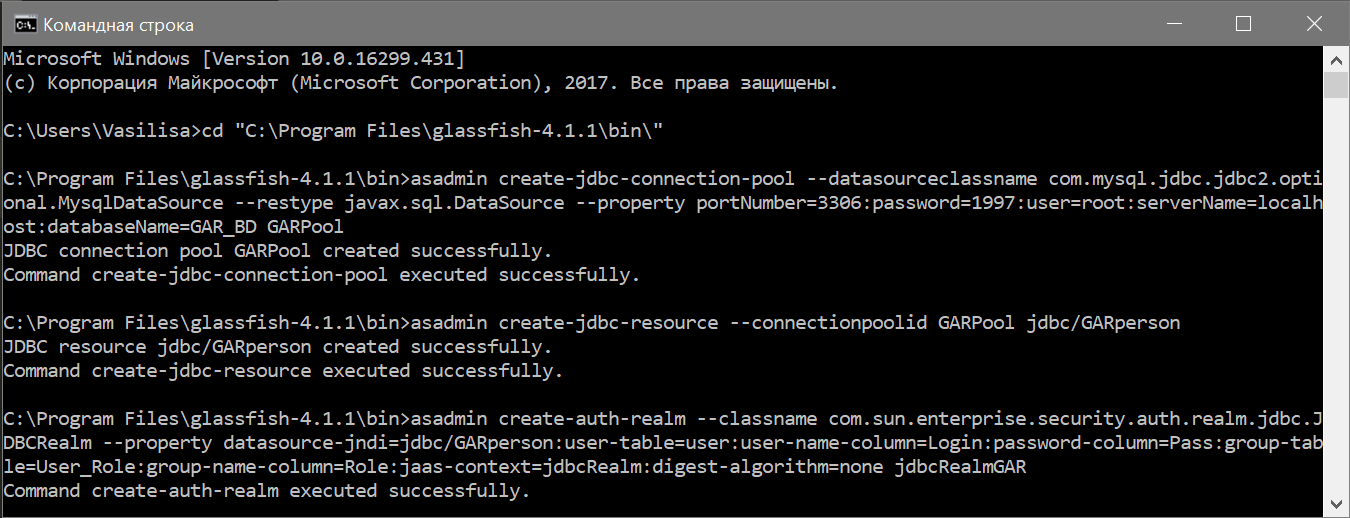


Рисунок 26. Создание подключения и настройка домена безопасности.

* Убедимся в успешности выполненных действий.

Для этого … (рис. 27)

В адресную строку браузера введем: localhost:4848 (рис. 27)

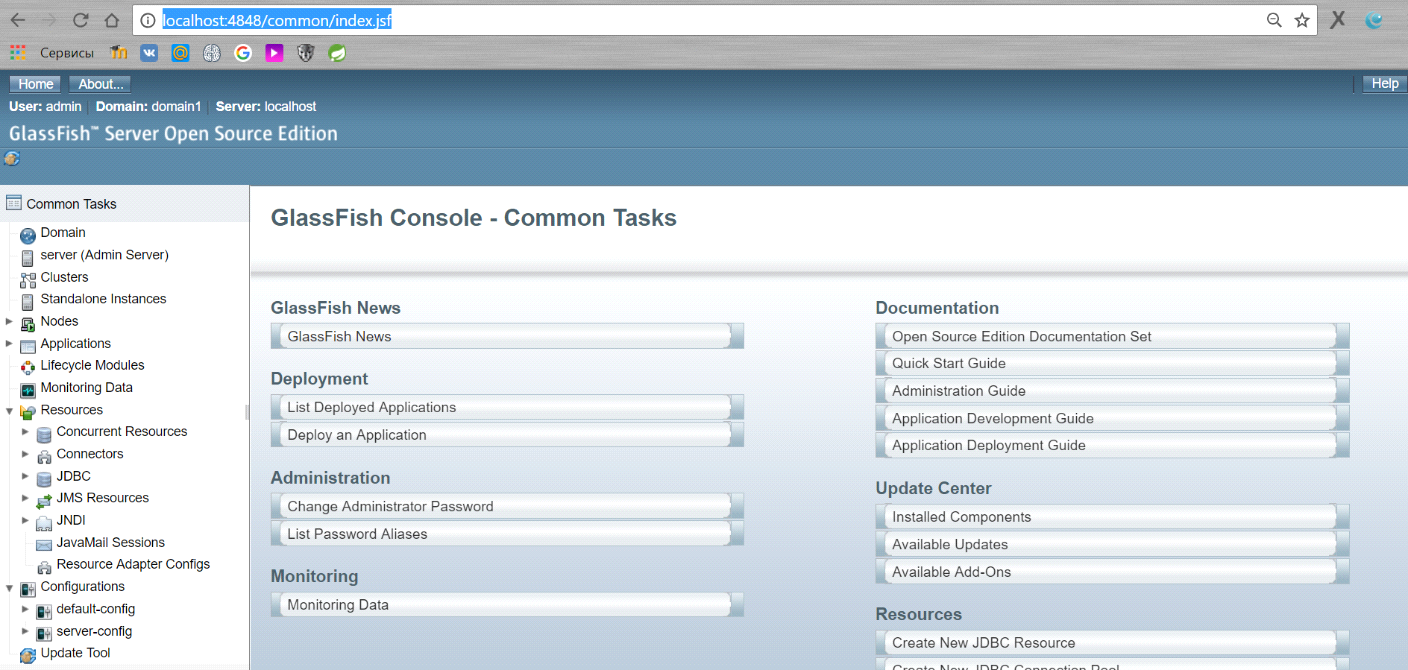


Рисунок 27. Админка сервера.

На вкладке JDBC прейдем в JDBC Connection Pools и зайдем в созданное нами подключение нажмем кнопку Ping. Получим сообщение «Ping Succeeded» (рис. 28)

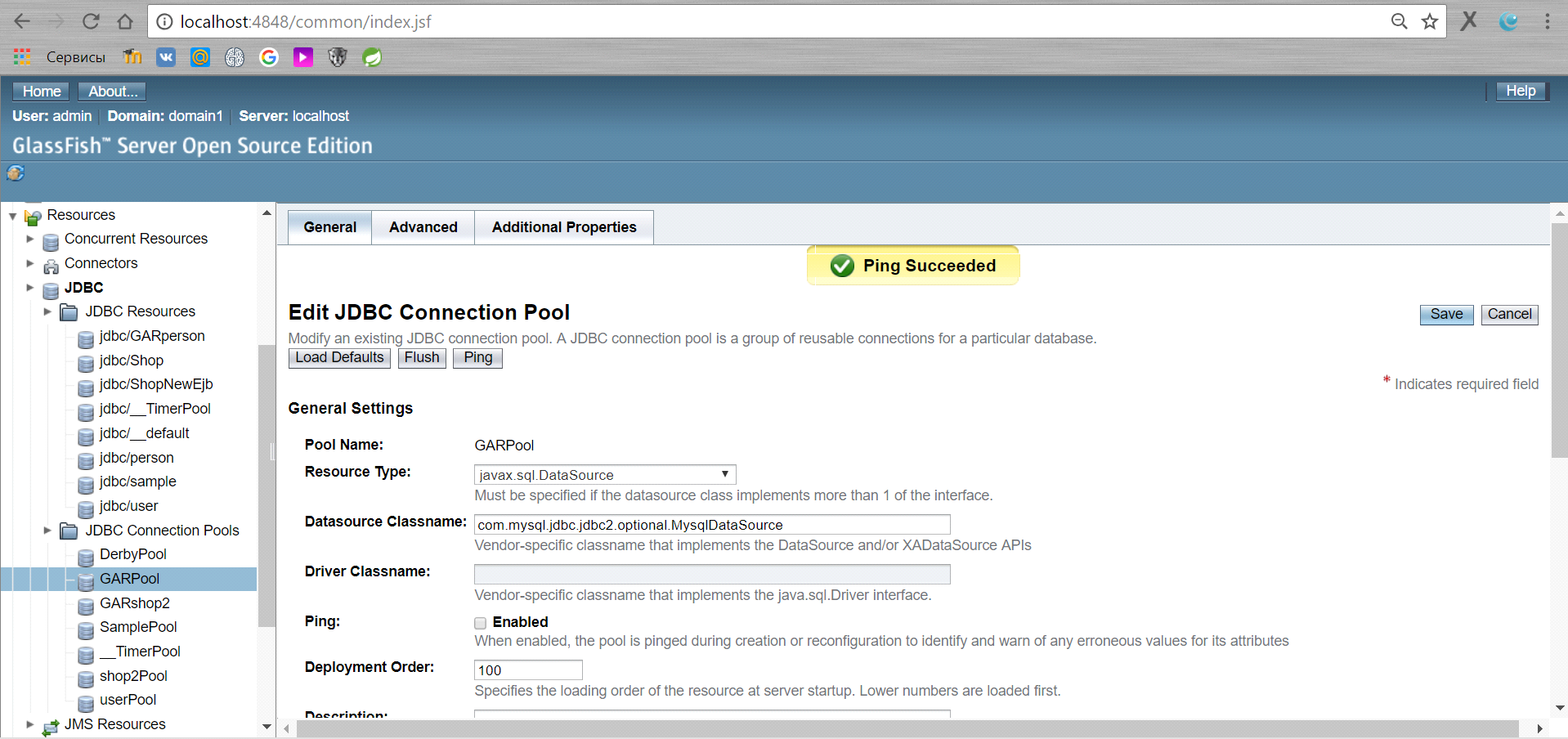


Рисунок 28. Установленное подключение.

Шаг 7: Открытие проекта

* Откроем среду разработки IDE NetBeans
* На панели выберем открыть проект после чего откроем разархивированный в пункте 3 проект (рис. 29)

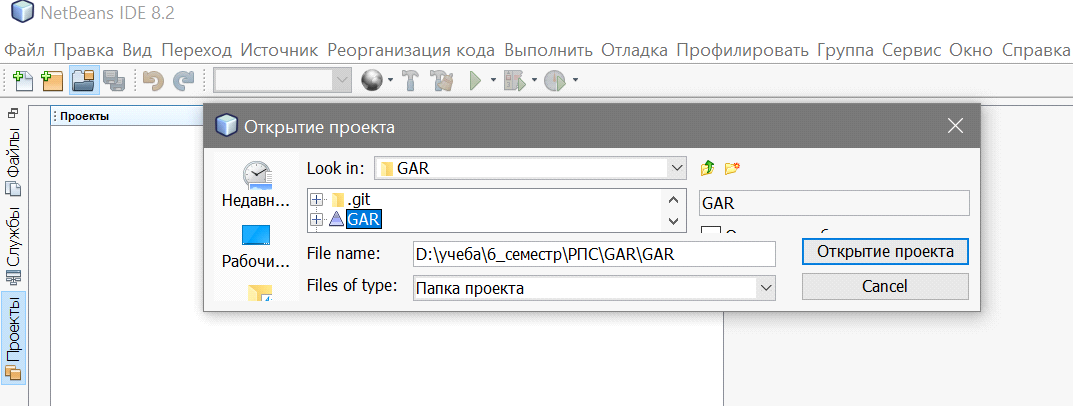


Рисунок 29. Открытие проекта.

* Откроем необходимые модули (рис. 30)

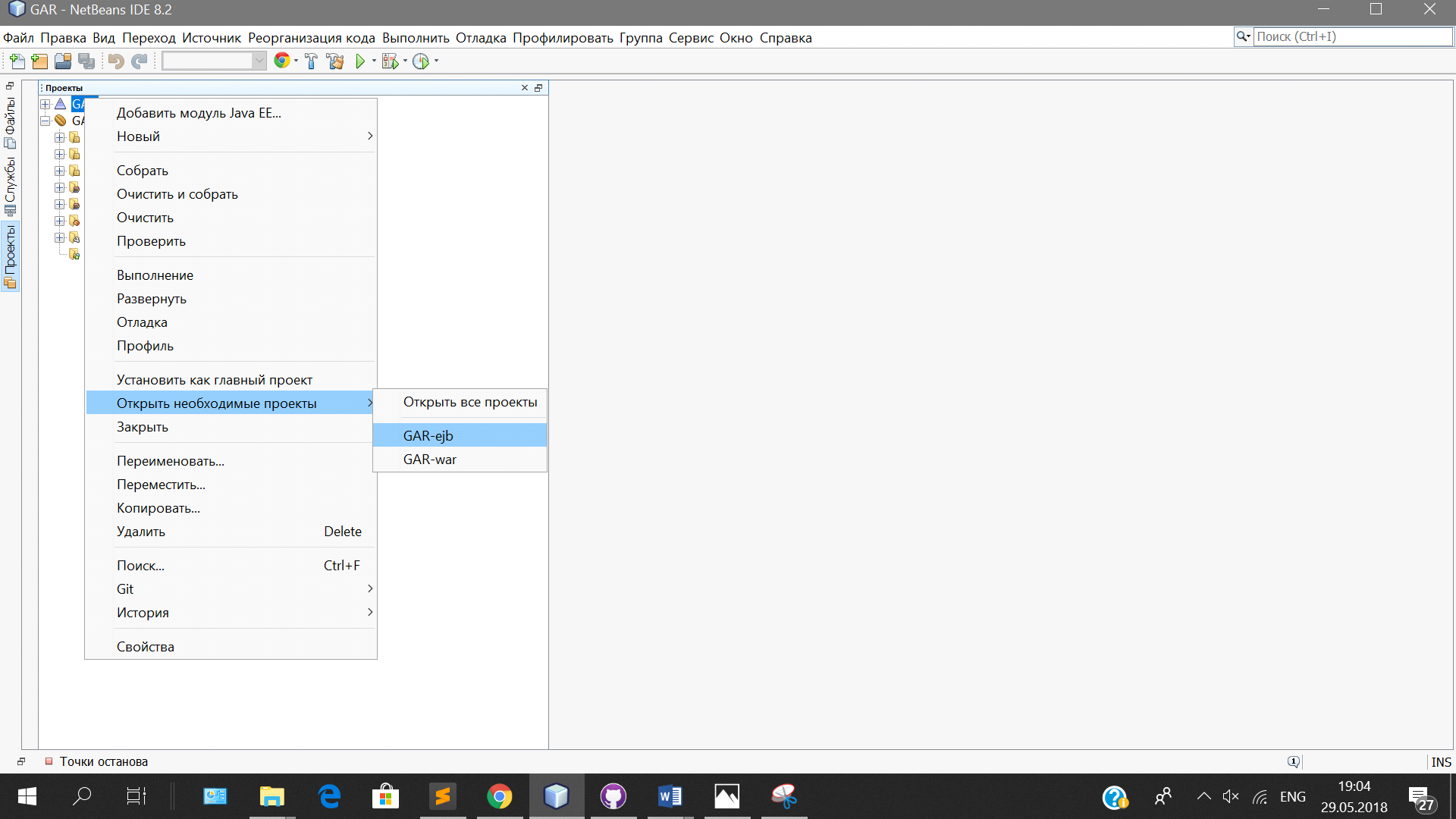


Рисунок 30. Открытие необходимых модулей.

Шаг 8: Деплой проекта

* В модуле persistence.xml укажем стратегию создания таблиц: Create.

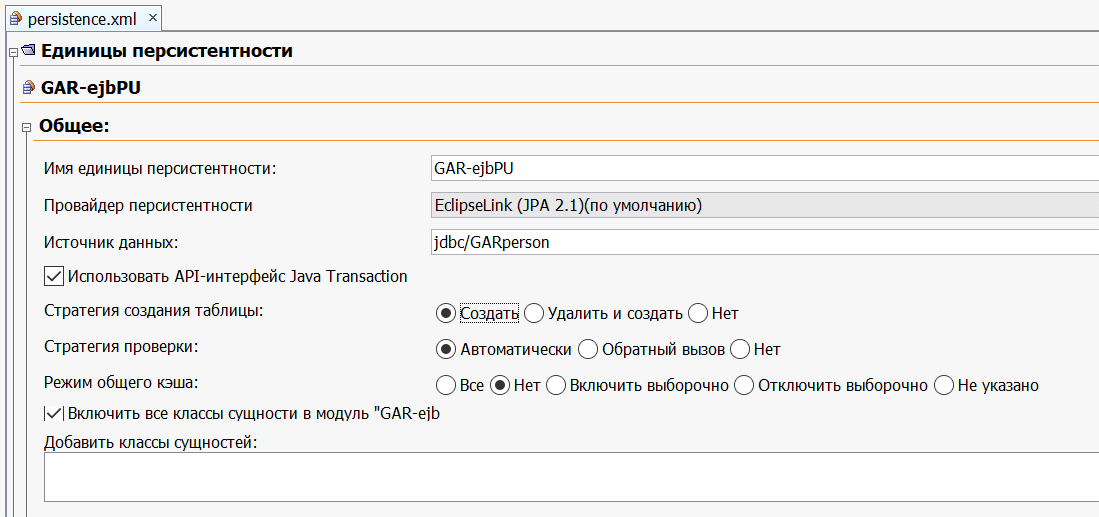


Рисунок 31. Модуль персистенстности.

* Развернём наш проект как показано на рисунке 32

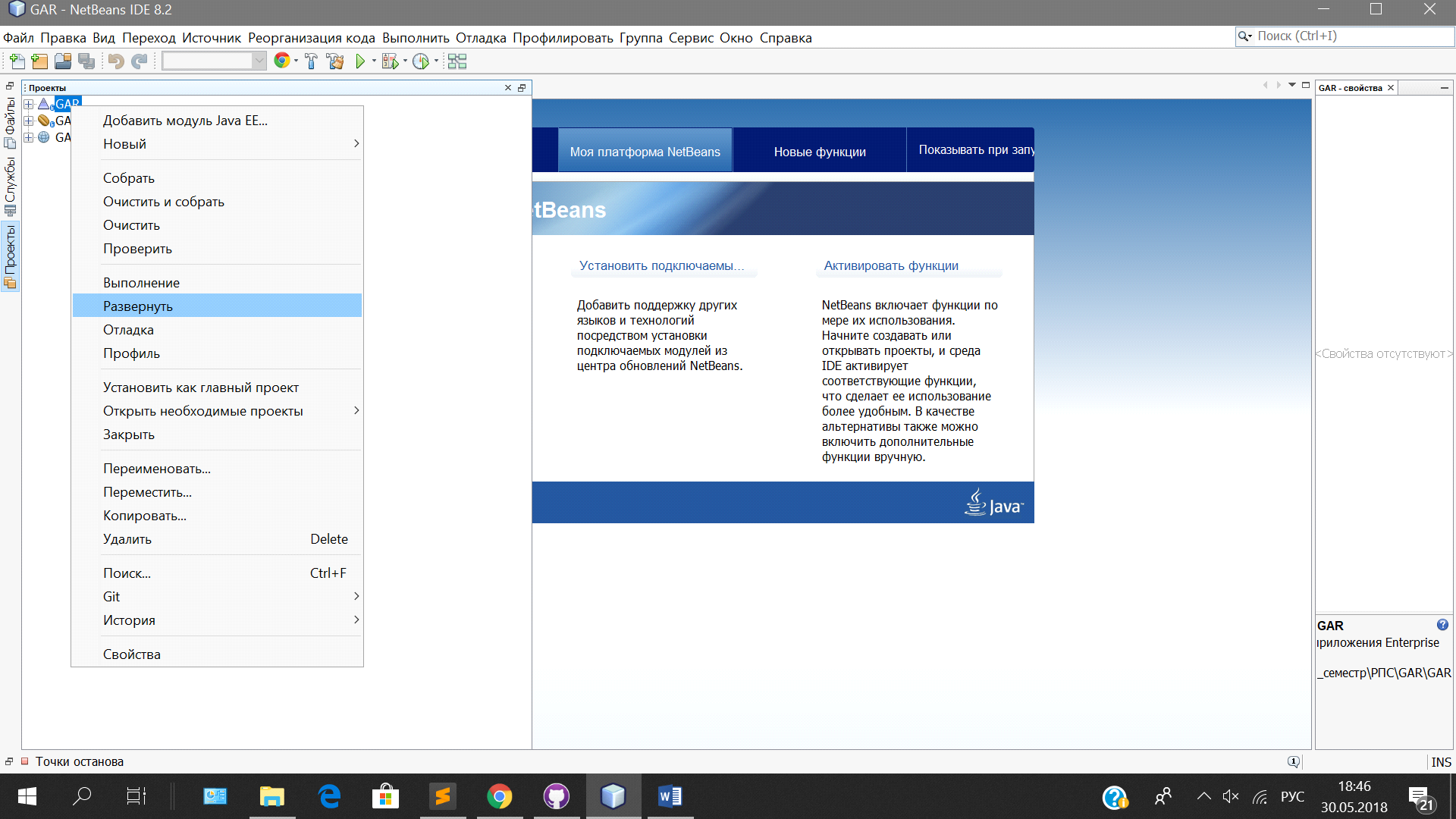


Рисунок 32. Разворачиваем Java EE приложение.

Шаг 9: Открытие проекта (рис. 33)

* В браузере в адресном сроке введите: <http://localhost:8080/GAR-war/>



Рисунок 33. Запущенное Web-приложение.

Шаг 10: Заполнение тестовыми данными (данный шаг не обязателен для функционирования приложения)

* Совет для заполнения лучше использовать: SQL Workbench так как, там не только удобный визуальный интерфейс, но есть возможность добавления изображений.
* Так же можете добавить тестовые данные по средствам скрипта:

use GAR\_BD; #открытие бд

#Добавление тестовых пользователей

Insert Into User(Login, Pass, Surname, Name, Phone, Email,Gender) Values('Boss', 'password', 'Boss','Boss','0000000000','adman@inbox.ru','женщина');

Insert Into User(Login, Pass, Surname, Name, Phone, Email,Gender) Values('gambler', 'password', 'Ivanov','Ivan','1111111111','gambler@inbox.ru','мужчина');

#Роли

Insert Into User\_Role(Login, Role) Values('Boss', 'admin');

Insert Into User\_Role(Login, Role) Values('gambler', 'client');

#Проверка

SELECT \* FROM User\_Role,User WHERE User\_Role.Login = User.Login;

#Заполнение дефолтного персонажа

Insert Into Personage(Name, Price) Values('Смурфик', 0);

#Заполнение дефолтного персонажа

Insert Into Personage(Name, Price) Values('Смурфик2', 10);

#Заполнение дефолтной цели 0-это дефолтная цель!!

Insert Into Goal(Name,Directory,Description,ID\_Personage) Values('Спать',0,'Чтобы выглядить красиво нужно высыпаться.',1);

#Дефолтный Игрок

Insert Into Client(Date\_Birth,Interests,ID\_User,ban) Values('1997-07-07','Люблю программировать,рисовать и спать.',1,0);

#Темы для форума

Insert Into Topic(Name,Date,ID\_User) Values('Персонаж','2018-05-04',1);

#Сообщения пользователя

Insert Into Message(Date,Content,ID\_User,ID\_Topic) Values('2018-05-04','Добавте нового персонажа.',1,1);

* Необходимо добавить 1 картинку в БД эта картинка по дефолту будет являться аватаркой регистрируемых пользователей. Этот пункт можно выполнить с помощью SQL Workbench или же в самом приложении.

Если вы без ошибочно выполнили все пункт, поздравляем Web – приложения «GAR» готово к использованию.

**5.2 Установка Web-сервиса**

Для использования функции покупки персонажа необходимо поднять Web-сервис.

Шаг 1: Убедитесь, что у вас установлены обязательные компоненты такие как:

* IntelliJ IDEA 2017.3.4 x64 и позднее (например учебная версия)
* СУБД MySQL (Можно использовать и другую СУБД)

Шаг 2: Скачайте и разархивируйте репозиторий (если вы еще этого не сделали). Подробная инструкция представлена в пункте 5.1.

Шаг 3: Открытие проекта.

* При запуске IntelliJ IDEA выбираем пункт Import Project (рис. 34).

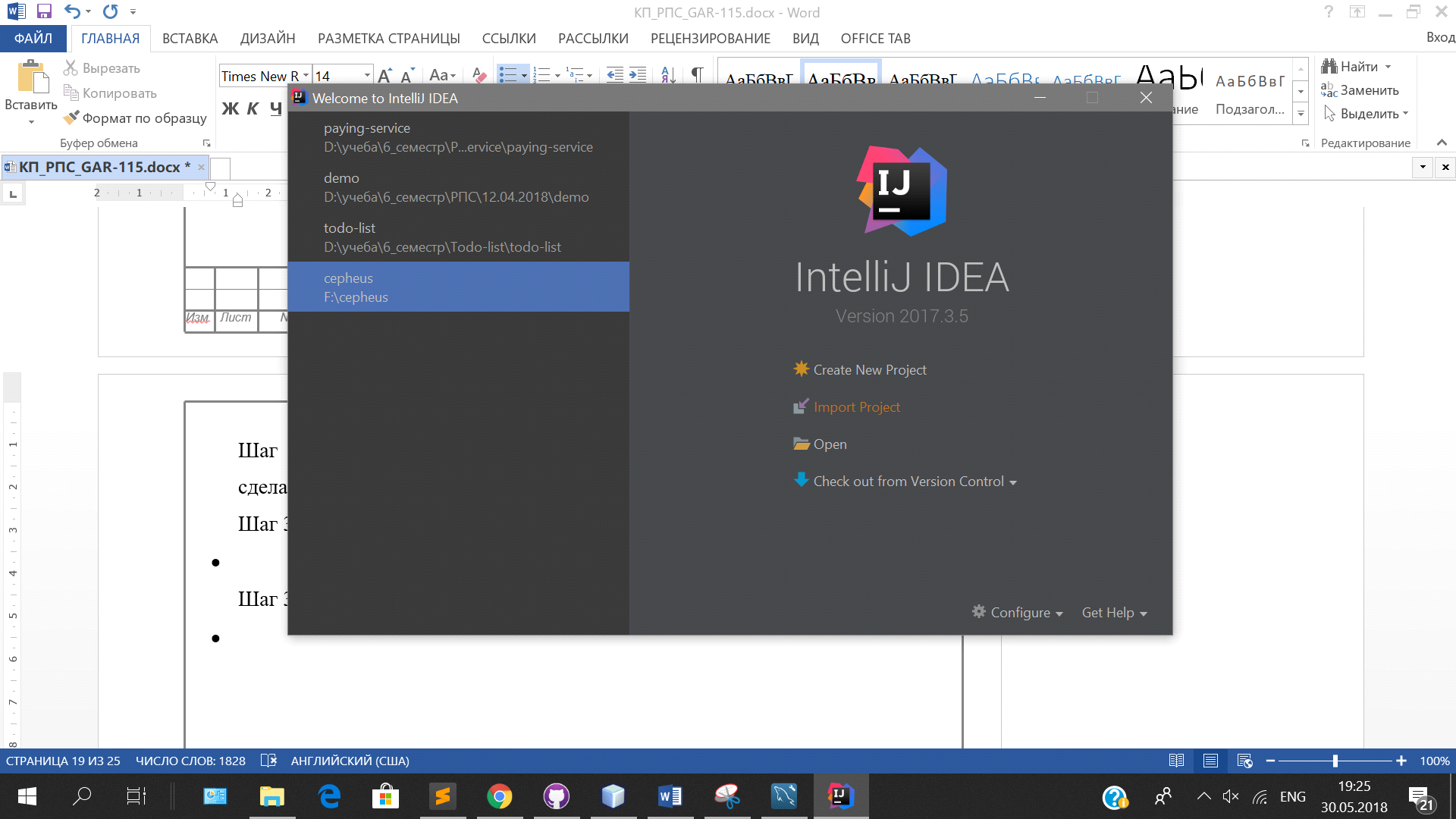


Рисунок 34. Импортируем проект.

* В проекте сервиса выбираем pom.properties ()

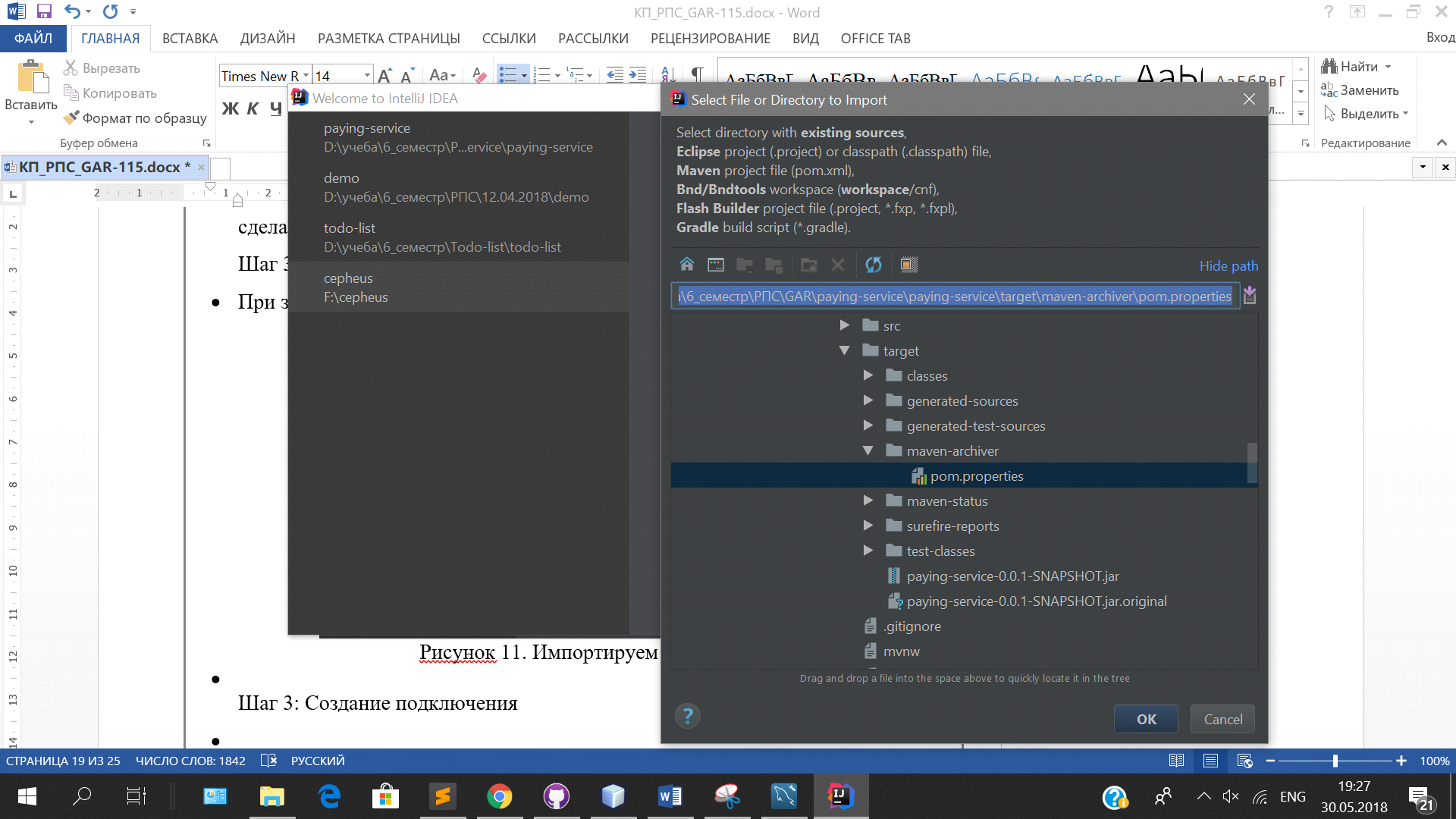


Рисунок 35. Импорт проекта.

* Выполняем настройки экспорта (в последнем пункте желательно добавить понятное вам имя проекта)

Шаг 4: Создание подключения

* Заходим в MySQL и создаем базу данных
* Выбираем вкладку Database после чего выполняем добавление нового подключения (зеленый плюсик) и выберем пункт создать подключение к MySQL.

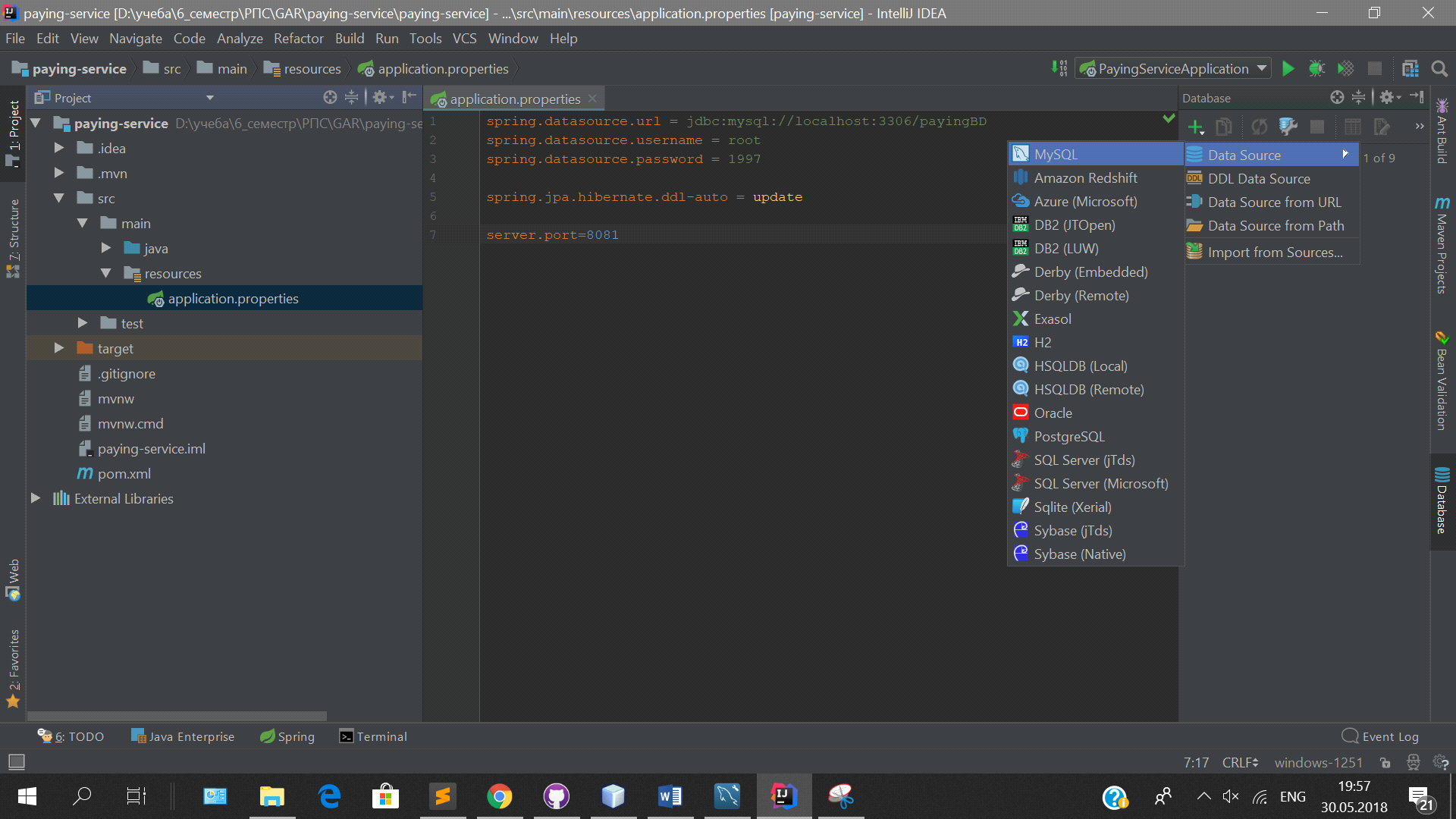


Рисунок 36. Создание подключения.

* Заполним необходимые поля и пропингуем подключение (рис. 37).

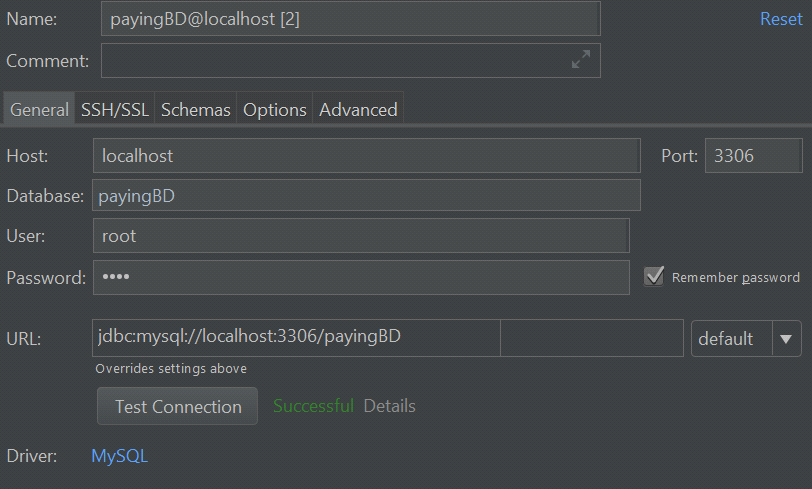


Рисунок 37. Создание подключения.

Шаг 5: Настройка информации о хранилище. Откроем файл application.properties и отредактируем в соответствии с этапами, выполненными ранее (рис. 38)

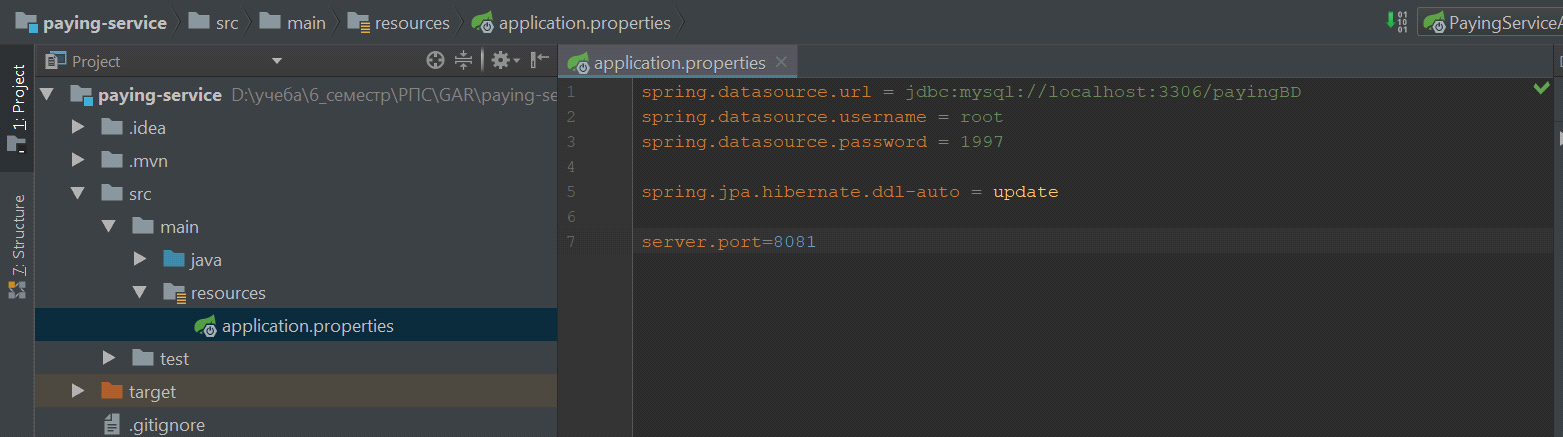


Рисунок 38. Application.properties.

Шаг 6: Сборка и разворачивание проекта

* На вкладке Maven Projects выполним команду «собрать» , после чего запустим наш проект.

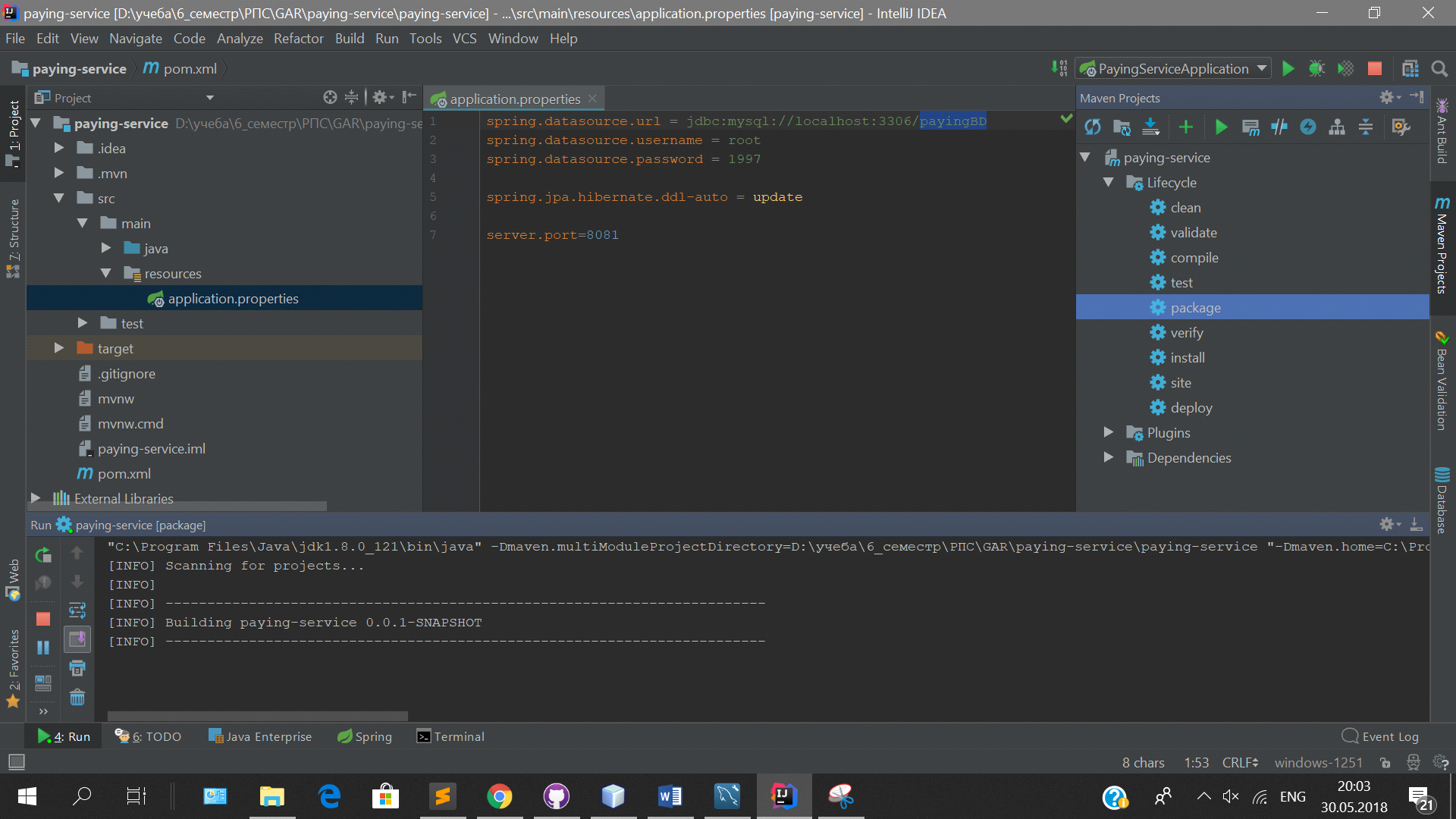


Рисунок 39. Сборка проекта.

Если все пункты выполнены верно, проект задиплоется без ошибок и наш сервис готов к использованию.

**6 ОРГАНИЗАЦИЯ УПРАВЛЕНИЯ ПРОЕКТОМ**

**6.1 Общие принципы взаимодействия в команде**

Для того чтобы добиться определенной эффективности коллективных взаимодействий, необходимо создать команду. Не каждый коллектив может стать командой единомышленников. Что подразумевается под термином «команда»? Это коллектив специалистов, которые дополняют друг друга своими навыками, знаниями и квалификацией, объединенных одним стремлением к решению коллективных задач, и при этом несущих личную ответственность за результат.

Для организации эффективной работы в нашей команде мы придерживались некоторых принципов:

• Позитивное отношение, ясность и общие цели для нас куда более сильные мотиваторы, чем дедлайны и бесконечные списки дел. Нет ничего плохого в стрессе как таковом, но без положительных эмоций он приносит лишь психологические страдания и снижает продуктивность.

• Мы ценим удобство правил и предписаний, но воображение и возможность меняться для нас куда важнее. Структурированная командная работа, правила, сам процесс — эффективны только при условии их гибкости. Структура является инструментом и должна меняться по ходу того, как развивается рабочий процесс.

• Мы ценим индивидуальные качества каждого, но способность работать в команде и разделять общие цели для нас куда важнее. Хорошая идея приходит к кому-то одному, но воплощает в жизнь её вся команда. Только находясь вместе, общаясь напрямую и оценивая работу друг друга, мы создали веб-приложение GAR.

Взаимодействие и общение команды происходит посредством общей конференции в социальной сети Вконтакте.

**6.2 Распределение ролей в команде/зоны ответственности**

Можно говорить о некоторых признаках организации хорошей команды. В ней участники считают себя частью рабочей группы. Соблюдается баланс между эффективной командной работой и сотрудничеством. Члены группы чувствуют себя компетентными, выполняют задания самостоятельно и несут ответственность за их выполнение. Между ними идет непрерывная дискуссия для улучшения сотрудничества и повышения эффективности командной работы. Каждый человек свободно предлагает свои идеи и критикует других.

Члены команды знают о заданиях друг друга и имеют представление о талантах и способностях каждого, что означает интерес и уважение друг к другу. Поэтому роли в нашей команде распределились следующим образом:

*Родионова Анастасия* - лидер команды, организатор разработки проекта, отвечает за Back-end.

*Лапыгина Василиса* - отвечает за Back-end разработку проекта и создания веб-сервиса оплаты.

*Ханова Анастасия* - отвечает за Front-end разработку, проектировщик UX интерфейса и ответственная за юзабилити проекта.

*Крутова Анастасия* - дизайнер сайта, отвечает за цветовую палитру и UI элементы, а также за прорисовку персонажей.

**6.3 Календарный план работ**

Календарный план нашей работы представлен на диаграмме Ганта (рис. 40). На данной диаграмме представлен план выполнения работ, связанных с организацией нашего проекта.

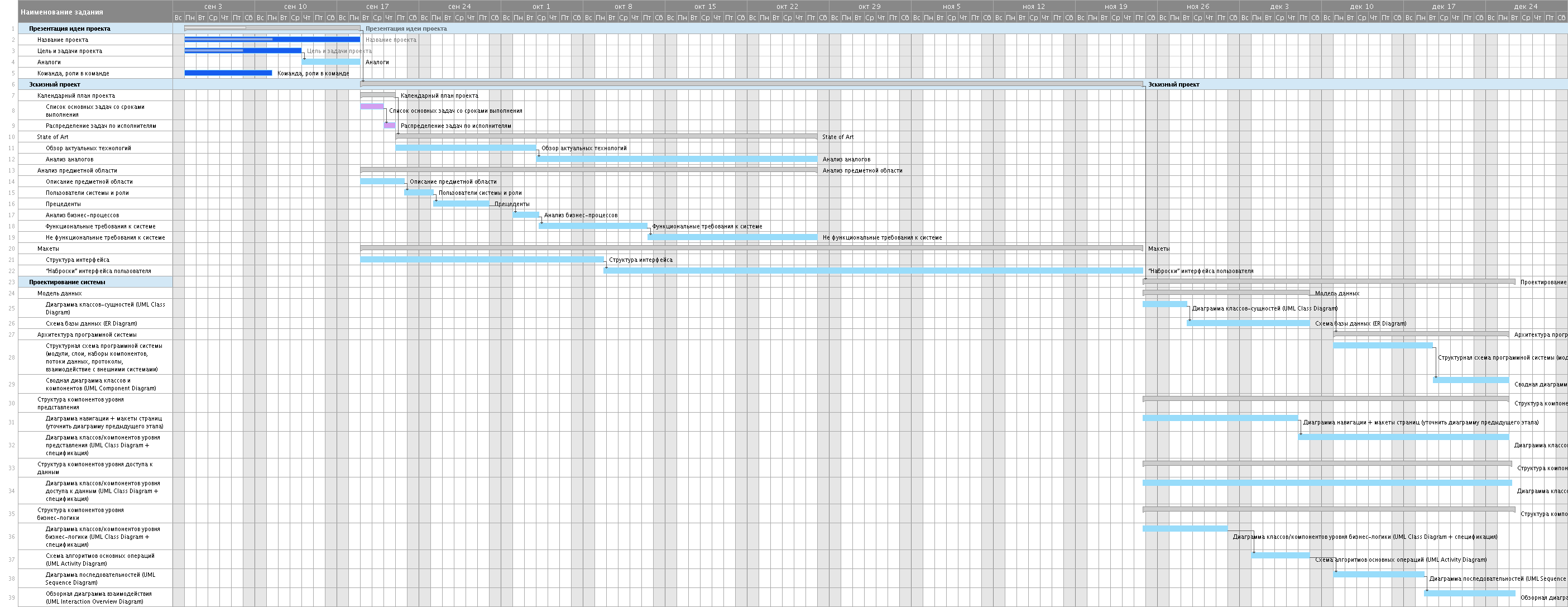


Рисунок 40. Диаграмма Ганта.

Подводя итог это главы, нужно отметить, что это была для нас первая крупная работа в команде. Были и разногласия, и споры по различным пунктам реализации системы, но мы всегда пытались найти компромисс. Безусловно, для нас это огромный опыт разработки в команде.

**7 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ**

Для начала разберем понятие нагрузочного тестирования и расскажем для чего мы его проводим.

Нагрузочное тестирование - вид тестирования производительности, проводимый с целью оценить поведение компонента или системы под увеличивающейся нагрузкой (число одновременно работающих пользователей и/или число транзакций) для определения максимально допустимого уровня нагрузки для исследуемого компонента или системы.  [глоссарий терминов тестирования стр 37]. Из определения становиться понятно, что нашей целью проведения нагрузочного тестирования будет являться оценка поведения системы под увеличивающейся нагрузкой.

Так как наше веб приложение предполагает регулярное (ежедневное) посещение сайта пользователями и целью нагрузочного тестирования является выявить максимальный уровень нагрузки, то нагрузка составит для начала 30 человек в минуту.

Определим сценарий на основании основных действий пользователя в нашем приложении для проведения нагрузочного тестирования:

* Зарегистрироваться – для данного действия будет использоваться файл \*.txt c тестовыми данными, так как есть необходимость в уникальном логине.
* Аутентифицироваться – только что зарегистрированным пользователем.
* Добавить цель.
* Выполнить добавленную целью.
* Написать сообщение на форуме.

Проводиться нагрузочное тестирование будет с пользователями в минуту: 30,60,120,240,480.

Первым делом были созданы все запросы и протестирована их работоспособности (рисунок 41).

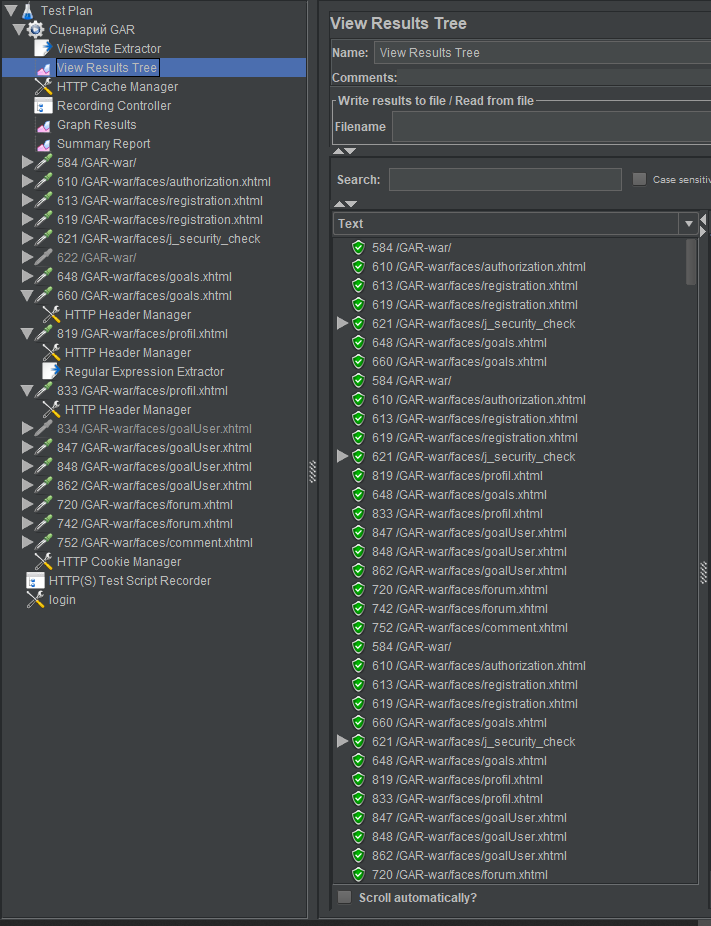


Рисунок 41. Структура.

Результаты тестирования по сценарию с данными нагрузками представлено в виде графиков на рисунках 42 – 46.

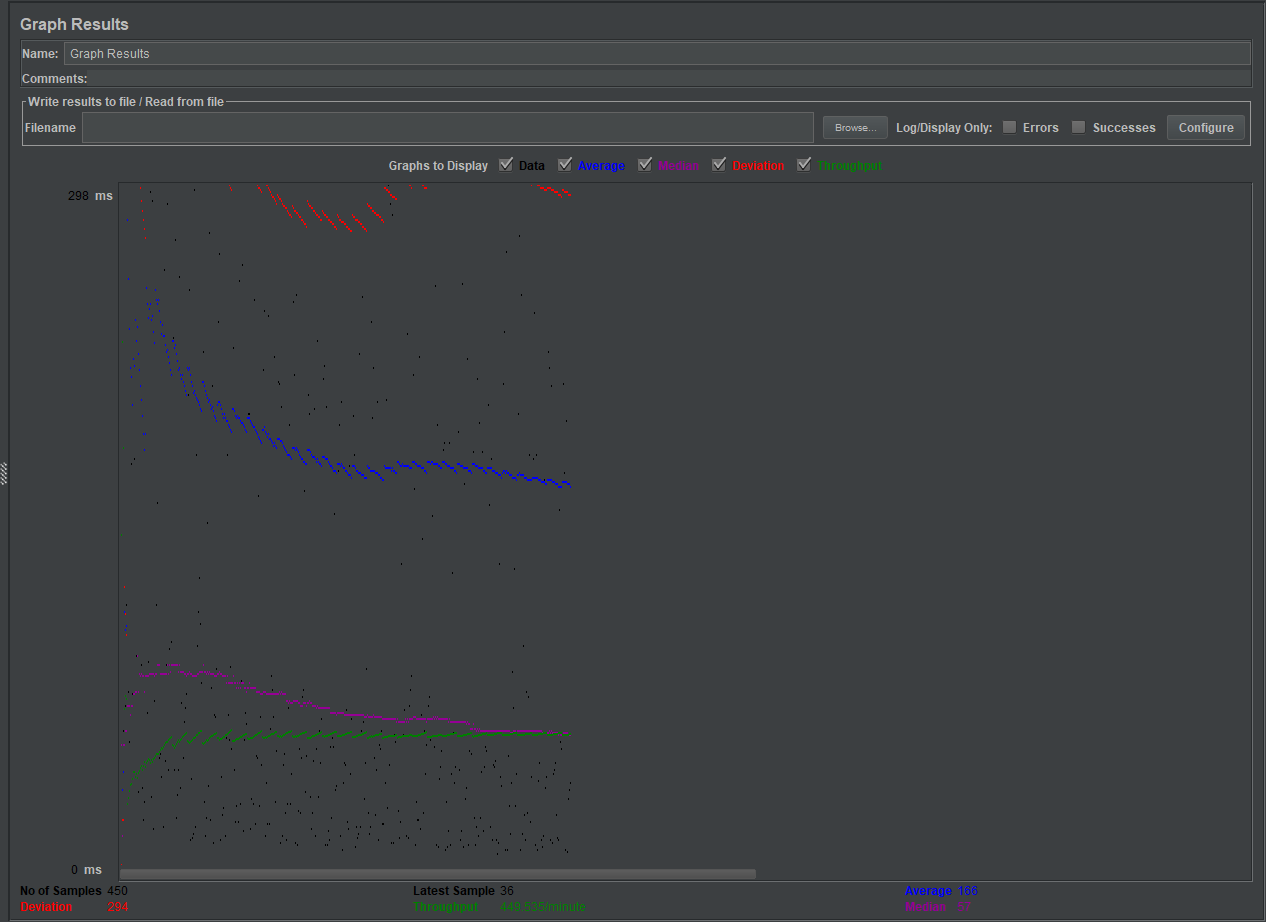


Рисунок … 30 пользователей в минуту.

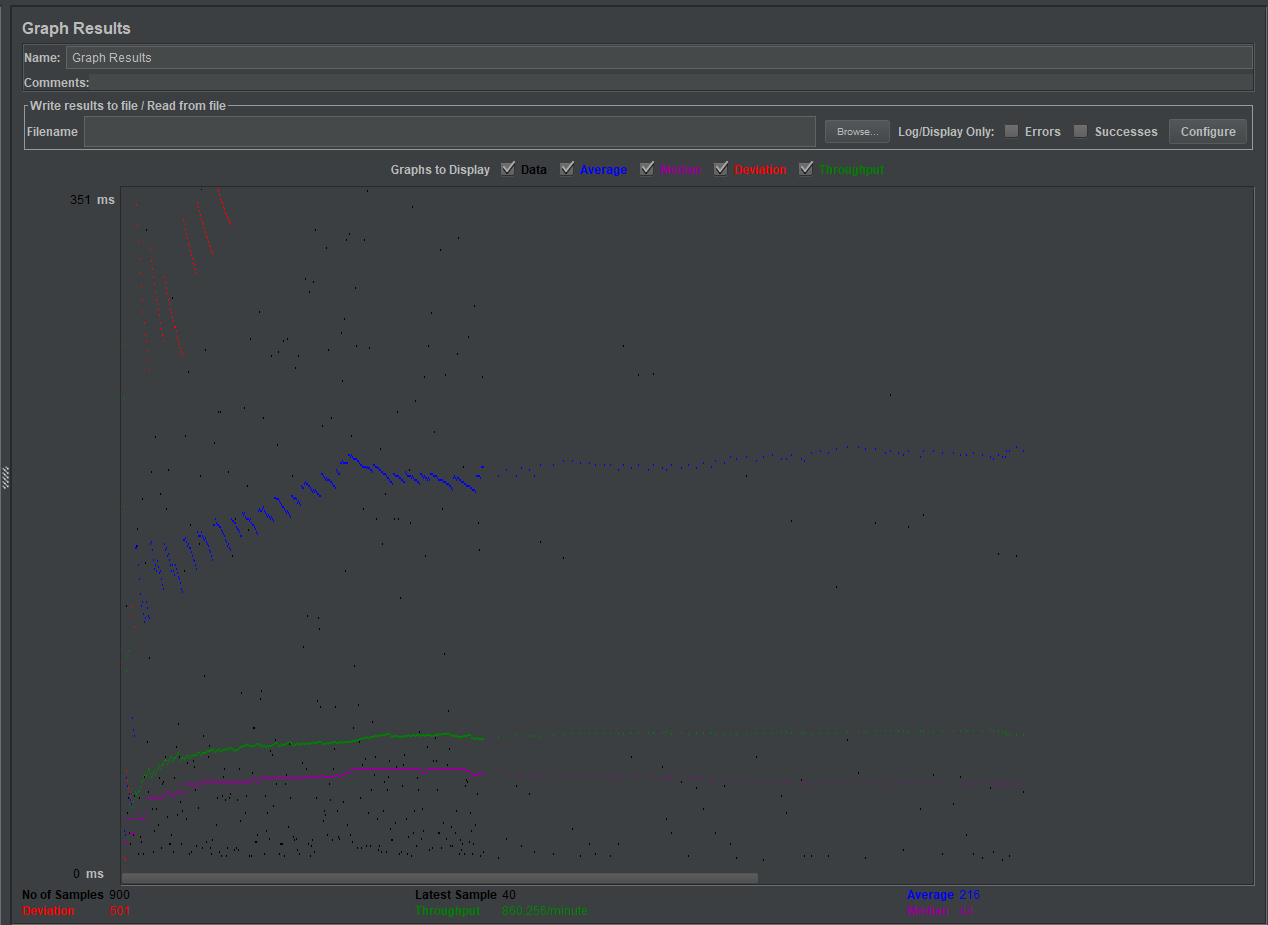


Рисунок … 60 пользователей в минуту.

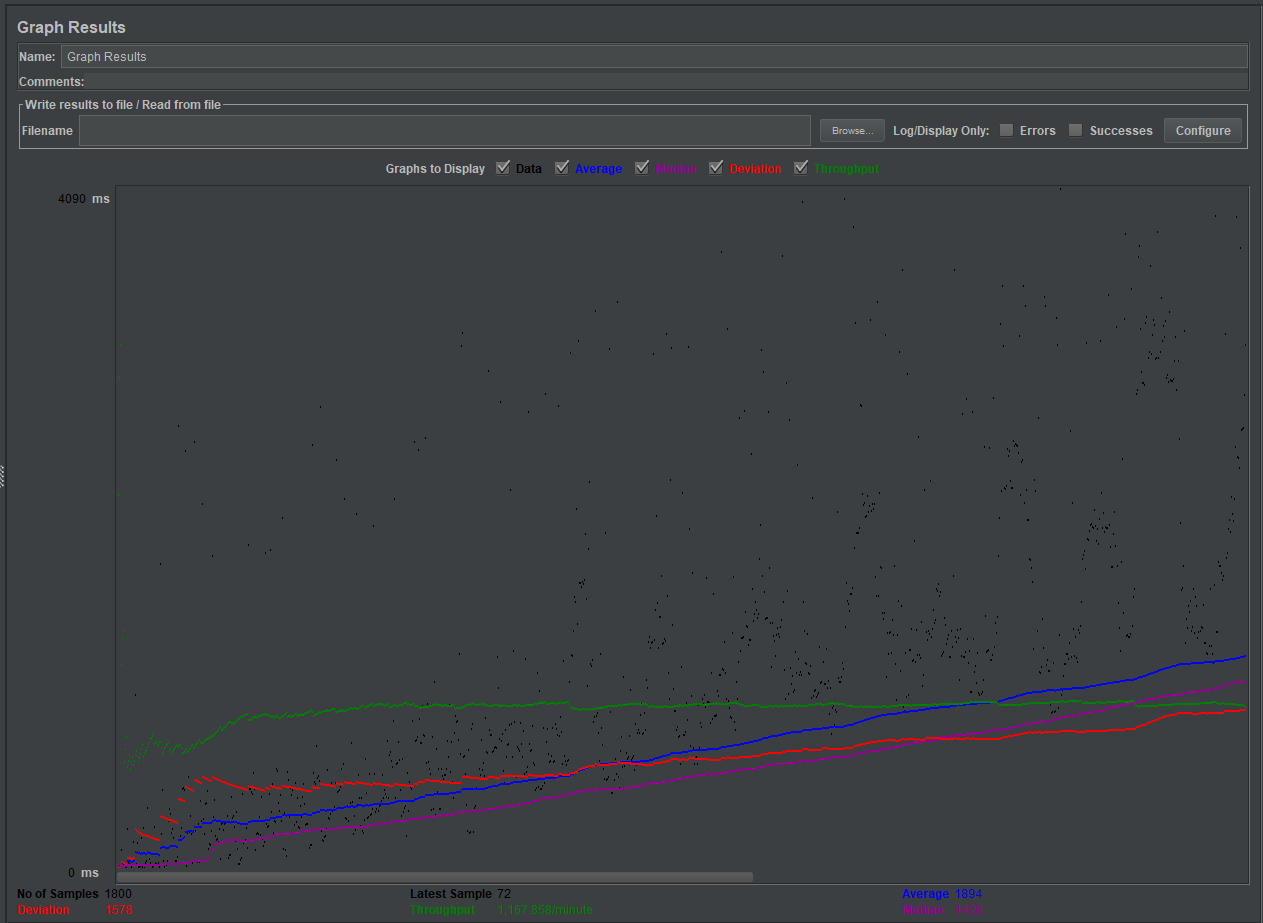


Рисунок … 120 пользователей в минуту.

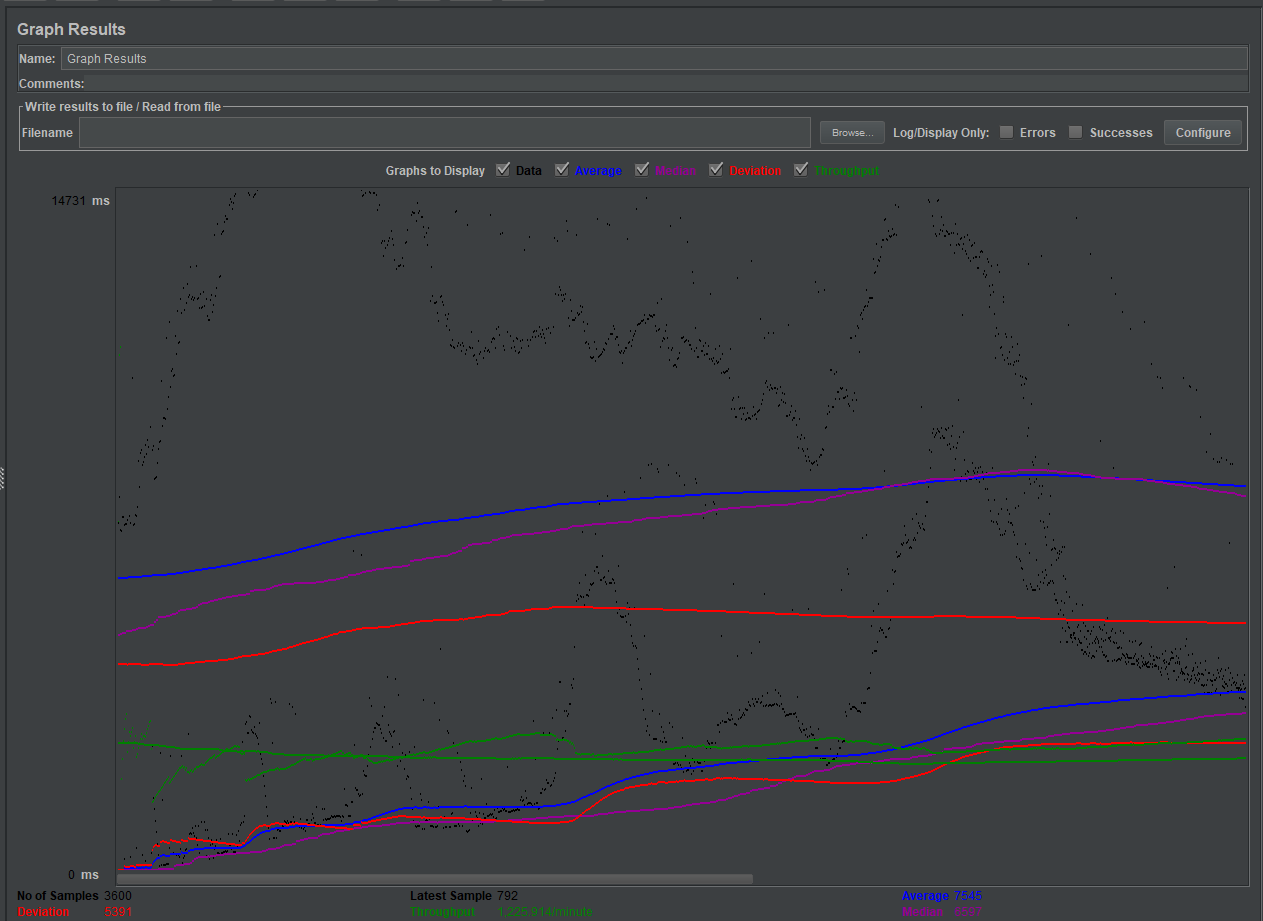


Рисунок … 240 пользователей в минуту.

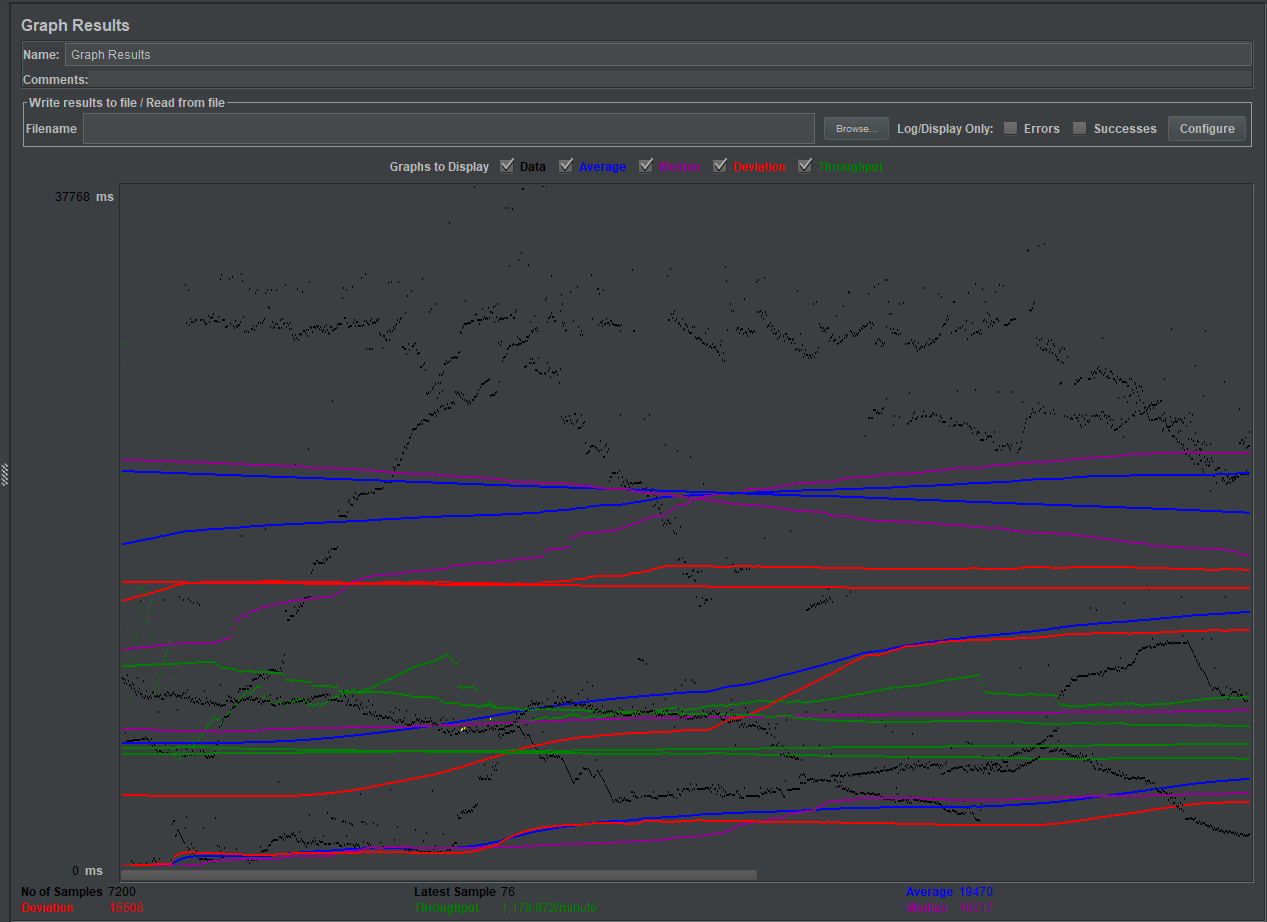


Рисунок … 480 пользователей в минуту

Когда вы ознакомились с результатами на графиках перейдем к их расшифровке:

* Data — время отклика на каждый выполненный запрос.
* Average — среднее время отклика сервера, объективный график нагрузки.
* Deviation — погрешность, стандартное отклонение.
* Throughput — скорость выполнения самого запроса.

Например, обратим внимание на среднее время отклика от сервера мы видим, что в первых двух вариантах время возрастает прямо пропорционально возрастанию количества запросов. Что касается дальнейших результатов этот показатель резко начинает увеличиваться, что означает чрезмерную загруженность нашего приложения. После 480 пользователей было решено что придел установлен, так как ожидание отклика от системы слишком возросло.

Нагрузочное тестирование проводилось с помощью Apache JMeter. Это удобный инструмент для проведения нагрузочного тестирования, с его помощью можно имитироваться нагрузку на Web-приложение в результате чего получать отчетность в виде графиков.

В результате нагрузочного тестирования было выявлено, что наше приложение удовлетворят нашим ожиданиям. Но конечно неоспоримым фактом является необходимость развиться и самосовершенствоваться.

**ЗАКЛЮЧЕНИЕ**

В данном проекте было создано веб-приложение мотивирующего характера с элементами игры. Были изучены новые технологии (такие как EJB, API), разработан собственный веб-сервис, затронута разработка Back-End и Front-End, изучены методы проектирования UX интерфейса. Но, в процессе выполнения данной курсовой работы были изучены не только технические способы разработки системы. Эта работа позволила получить существенный опыт в различных сферах, которые составляют разработку приложения: аналитика, разработка архитектуры, дизайн, который включает изучение цветовых сочетаний, создание макетов, тестирование, работа в команде. И результатом нашей совместной работы является готовое приложение, основная задача которого – помощь людям в достижении целей.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Блинов и. н., Романчик в. с. Java. Методы программирования : уч.-мет. пособие / И. Н. Блинов, В. С. Романчик. — Минск : издательство «Четыре четверти», 2013. — 896 с.

2. Гонсалвес Э. Г65 Изучаем Java EE 7. — СПб.: Питер, 2014. — 640 с.: ил.

3. Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Devika Gollapudi, Kim Haase, William Markito, Chinmayee Srivathsa: документация JAVA EE [электронный ресурс]. 2013. URL: https://docs.oracle.com/javaee/6/tutorial/doc/ (дата обращения: 06.06.2018).

4. Интуит, бесплатное дистанционное обучение [электронный ресурс].//введение в UML, 2008. URL: https://www.intuit.ru/studies/courses/1007/229/lecture/5950 (дата обращения: 06.06.2018).

5. Bootstrap 3, документация на русском [электронный ресурс]. URL: http://bootstrap-3.ru/index.php (дата обращения: 06.06.2018).

6. htmlbook.ru, самоучитель css. – pdf файл

7. Spring boot, документация [электронный ресурс]. 2018. URL: https://spring.io/guides/gs/spring-boot/ (дата обращения: 06.06.2018).

8. Александров Д.В., Шакеров Р.А. Администрирование в информационных ситемах. Методические указания к лабораторным работам. Владим. гос. ун- т. – Владимир : Ред.-издат. комплекс, 2005. – 46 с.

9. источник по тестированию

10. ГОСТ 2.105-95 Общие требования к текстовым документам

11. ГОСТ 19.701-90 Единая система программной документации СХЕМЫ АЛГОРИТМОВ, ПРОГРАММ, ДАННЫХ И СИСТЕМ

**ПРИЛОЖЕНИЕ А  
Листинг компонентов системы**

Спецификация:

**Спецификация:**

**ПРИЛОЖЕНИЕ Б  
Скрипт создания базы данных**