*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](#_Toc515488991)

[1 ПОСТАНОВКА ЗАДАЧИ 4](#_Toc515488992)

[1.1 Описание предметной области 4](#_Toc515488993)

[1.2 Основные понятия предметной области 4](#_Toc515488994)

[1.3 Функциональные требования к системе 5](#_Toc515488995)

[1.4 Нефункциональные требования к системе 5](#_Toc515488996)

[1.5 Сравнительный анализ аналогов 5](#_Toc515488997)

[2 АНАЛИЗ ЗАДАЧИ И ФУНКЦИОНАЛЬНАЯ ДЕКОМПОЗИЦИЯ СИСТЕМЫ 6](#_Toc515488998)

[2.1 Анализ бизнес-процесса "Достижение цели" 6](#_Toc515488999)

[2.2 Сценарий взаимодействия пользователя с системой 9](#_Toc515489000)

[2.3 Авторизация пользователя в системе 9](#_Toc515489001)

[2.4 Регистрация пользователя в системе 10](#_Toc515489002)

[2.5 Изменение данных пользователя 10](#_Toc515489003)

[2.6 Добавление дефолтной цели 11](#_Toc515489004)

[2.7 Добавление своей цели 11](#_Toc515489005)

[2.8. Просмотр своих целей 12](#_Toc515489006)

[2.9.Отметка о прохождении цели 12](#_Toc515489007)

[2.10. Просмотр рейтинга пользователей 12](#_Toc515489008)

[2.11. Создание темы на форуме 13](#_Toc515489009)

[2.12. Создание сообщения на форуме 13](#_Toc515489010)

[3 СТРУКТУРНАЯ ОРГАНИЗАЦИЯ СИСТЕМЫ 15](#_Toc515489011)

[3.1 Общие принципы организации системы 15](#_Toc515489012)

[3.2 Взаимодействие с внешними сервисами/удалённым модулем для оплаты 15](#_Toc515489013)

[3.3 Организация доступа к данным 17](#_Toc515489014)

[3.4 Организация бизнес-логики 18](#_Toc515489015)

[3.5 Организация веб-интерфейса 18](#_Toc515489016)

[3.6 Взаимодействие компонентов системы для цели 23](#_Toc515489017)

[3.7 Организация обработки ошибок 24](#_Toc515489018)

[3.8 Организация управления доступом 25](#_Toc515489019)

[4 РЕАЛИЗАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ 26](#_Toc515489020)

[4.1 EJB-компоненты/компоненты бизнес-логики 26](#_Toc515489021)

[4.2 Веб-компоненты 28](#_Toc515489022)

[4.3 Физическая структура баз данных 33](#_Toc515489023)

[4.4 Обеспечение целостности данных 34](#_Toc515489024)

[4.5 Реализация взаимодействия с внешним сервисом 36](#_Toc515489025)

[5 ИНСТРУКЦИЯ ПО УСТАНОВКЕ ПРИЛОЖЕНИЯ 38](#_Toc515489026)

[5.1 Установка приложения 38](#_Toc515489027)

[5.2 Установка Web-сервиса 44](#_Toc515489028)

[6 ОРГАНИЗАЦИЯ УПРАВЛЕНИЯ ПРОЕКТОМ 48](#_Toc515489029)

[6.1 Общие принципы взаимодействия в команде 48](#_Toc515489030)

[6.2 Распределение ролей в команде/зоны ответственности 49](#_Toc515489031)

[6.3 Календарный план работ 49](#_Toc515489032)

[Рисунок . Диаграмма Ганта. 50](#_Toc515489033)

[7 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ 51](#_Toc515489034)

[ЗАКЛЮЧЕНИЕ 52](#_Toc515489035)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 53](#_Toc515489036)

[ПРИЛОЖЕНИЕ А Листинг компонентов системы 54](#_Toc515489037)

[ПРИЛОЖЕНИЕ Б Скрипт создания базы данных 55](#_Toc515489038)

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

В проекте будет присутствовать несколько характерных для нашего приложения сущностей. Для начала опишем понятия из системы, с пониманием которых у пользователя могут возникнуть проблемы, а далее рассмотрим диаграмму классов концептуального уровня (рис. 1).

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 на рисунке 5.

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.

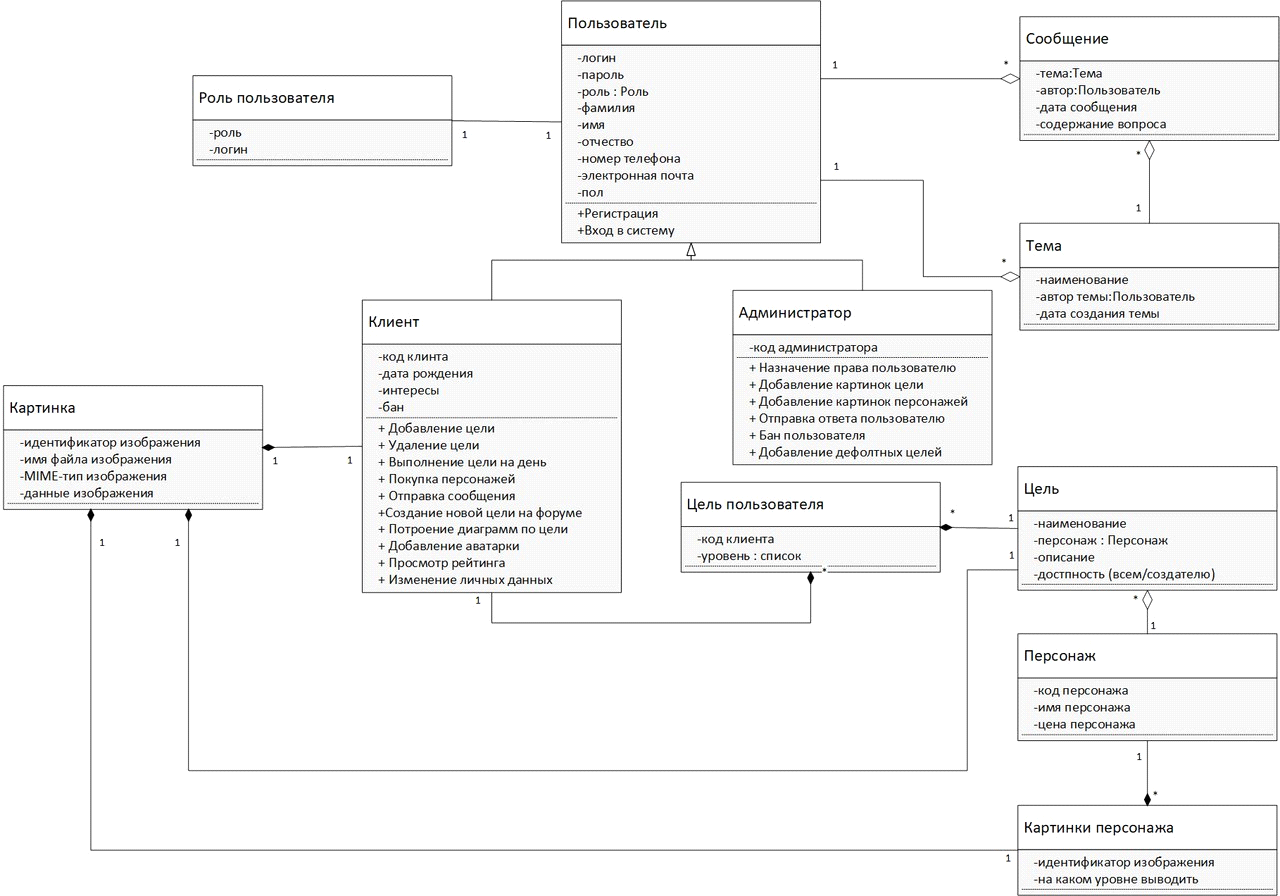


Рисунок 1. Диаграмма классов концептуального уровня.

В диаграмме классов концептуального уровня присутствует несколько сущностей, далее немного подробнее рассмотрим каждую.

«Пользователь» – человек, который использует данную систему для выполнения конкретной функции, у этой сущности в нашей системе будет «роль пользователя», которая определяет, каким из двух типов пользователей он будет: «администратором» или «клиентом». Для клиента присущ такой класс, как «картинка», он нужен для реализации возможности хранить, обрабатывать и выводить изображения. Не стоит путать его с классом «картинка персонажа», который отвечает за изображения, отображающие стадии развития персонажа. «Картинка персонажа» связана с сущностью «персонаж», которая отвечает за персонажей, относящихся к «цели». Так же есть «цель пользователя», ответственная за цель, находящуюся в обращении у конкретного пользователя. Ещё есть два таких класса, как «тема» и «сообщение», они и есть форум.

1.4 Функциональные требования к системе

Необходимо было сформулировать конкретные 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.

Приложение должно предоставлять пользователю возможность заполнить профиль персональными данными, то есть содержать такие поля, как логин, имя, фамилия, контактный телефон, э-мейл, так же она будет отображать рейтинг пользователя и плагин рейтинга. А ещё пользователь должен иметь возможность добавить в профиль свою фотографию.

Так же существует такое требование, как создание или добавление цели. Добавление цели – это процесс выбора цели из заготовленных заранее. Готовые цели бывают дефолтными, то есть созданными администратором системы, и созданными другими пользователями, они могут попасть в готовые, если администратора заинтересует чья-то цель, созданная для личного осваивания, в случае, когда администратор одобрит чью-то цель, она станет доступна для всех пользователей. Личную цель пользователь может создать сам, придумав для неё название, заполнив все подцели в количестве двадцать одной штуки и выбрав любого из доступных персонажей, как платного, так и бесплатного. Так же в процессе достижения цели пользователь должен иметь возможность отмечать каждый пункт опознавательным знаком, тем самым, развивая персонажа, присущего своей цели.

Система должна предоставлять возможность контактировать с другими пользователями и с администрацией ресурса для повышения уровня мотивации в процессе общения и для решения каких-либо технических неполадок, чтобы удовлетворить данное требование в приложении будет реализован форум. Пользователь должен будет иметь возможность отследить свою позицию в общем рейтинге пользователей, что будет подстёгивать его ещё сильнее в выполнении целей. Рейтинг будет зависеть от количества выполненных целей.

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.

После определения конкретных функциональных требований можно переходить к их анализу.

1.4 Нефункциональные требования к системе

Сформулируем нефункциональные требования, в которых определимся с выбором языка программирования, платформы, структуры приложения и других деталей предстоящей разработки.

Ограничения:

* Разработка системы должна быть на языке Java, платформа Java EE.
* Система должна представлять из себя веб-приложение.
* Система должна использовать технологию EJB.
* 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. Она должна выдерживать не менее 120 пользователей в минуту

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).

На данном этапе были определены все основные нюансы разработки системы.

1.5 Сравнительный анализ аналогов

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

Название: Lift

Интерфейс представлен на рисунке 2.

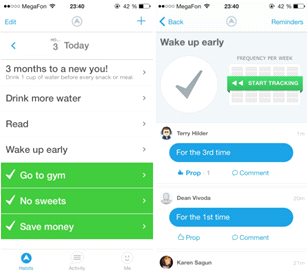


Рисунок 2. Интерфейс Lift.

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

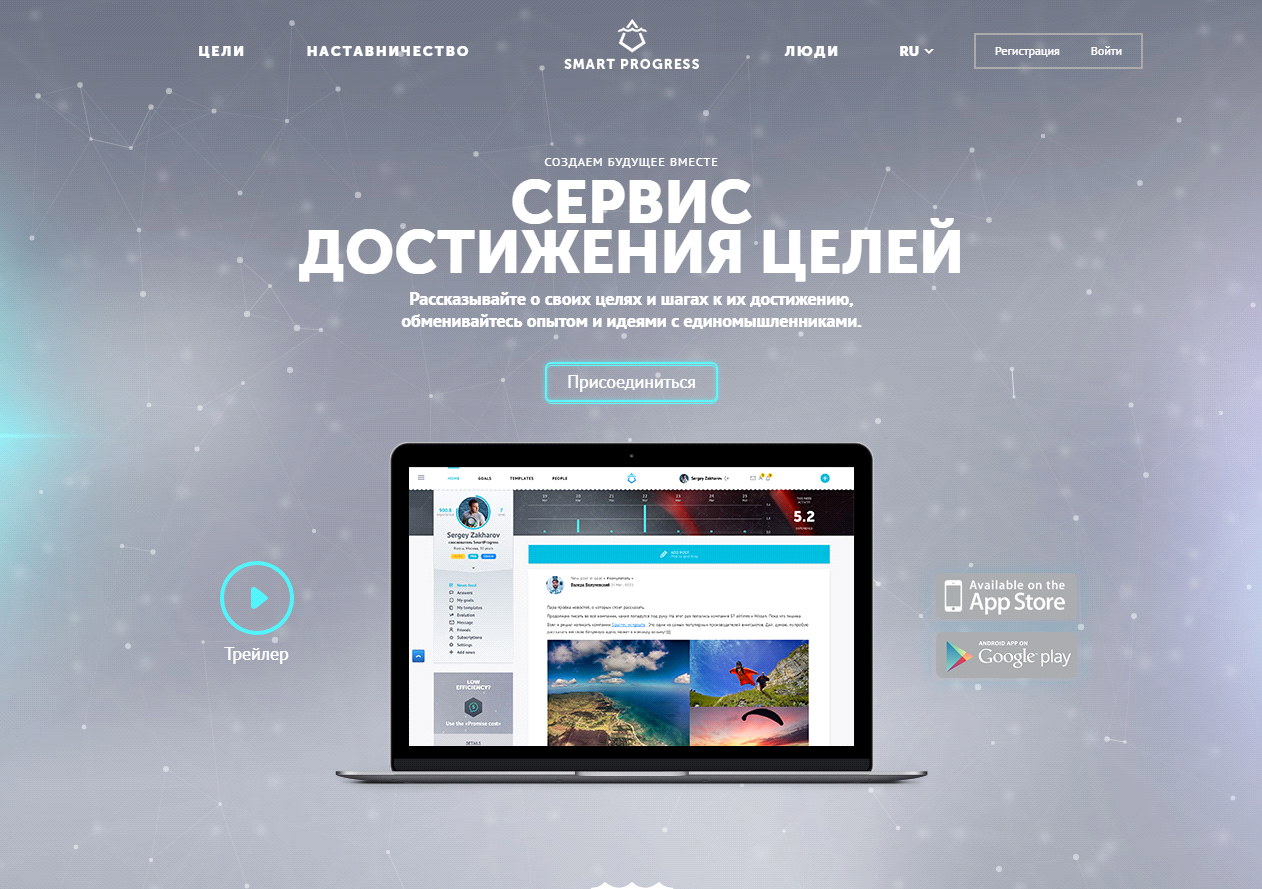


Рисунок 3. Интерфейс SmartProgress.

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

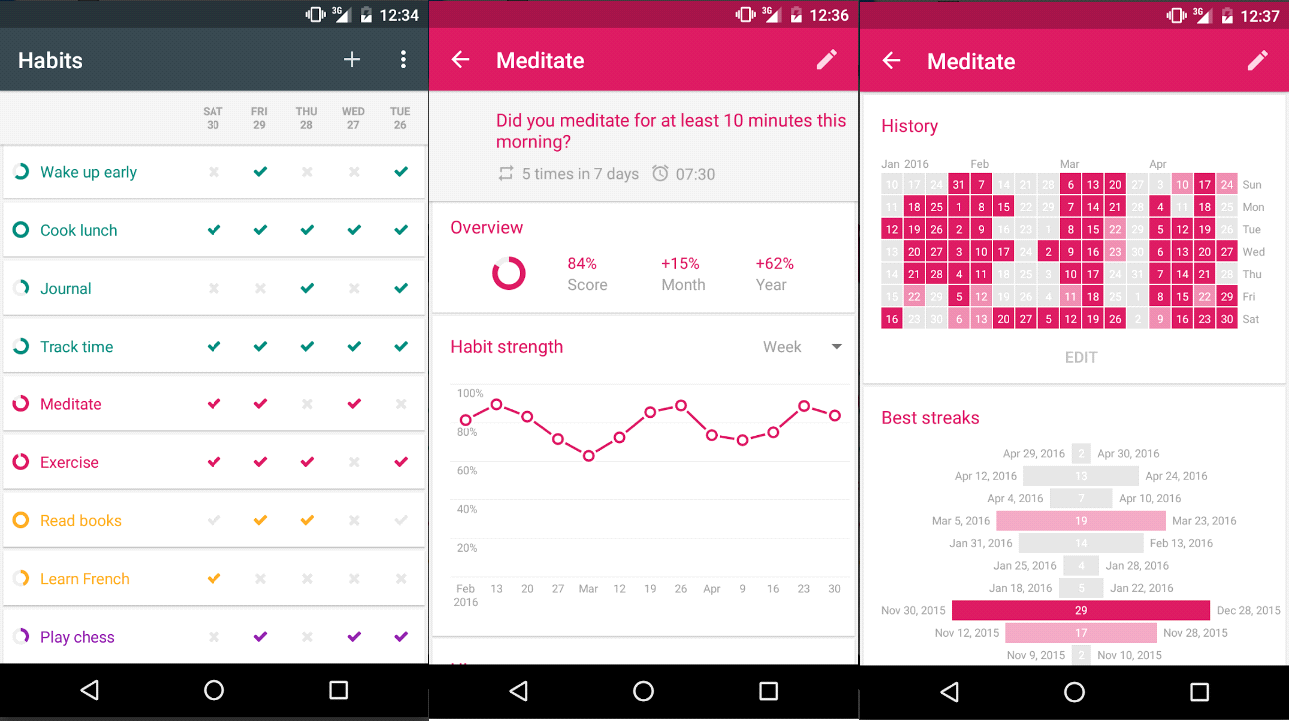


Рисунок 4. Интерфейс Трекера привычек Loop.

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

В результате анализа аналогов было выяснено, что наше приложение ничуть не хуже других приложений из данной области: интерфейс на должном уровне, игровая часть в нашем приложении выражена явно, что заставляет пользователя заходить в систему чаще, добавляет интерес. Единственный наш минус - это то, что наша система пока взаимодействует с пользователем только на русском языке, но в дальнейшем развитии системы этот функцоинал будет не сложно добавить. Так что, наше приложение лучше аналогов.

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].

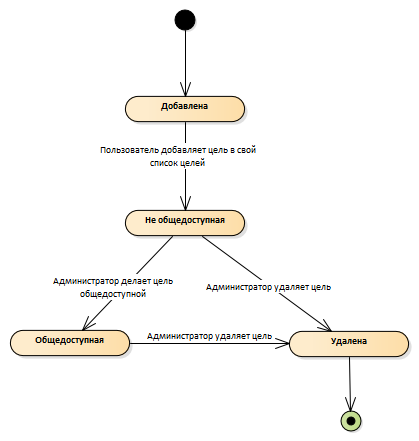


Рисунок 10. Диаграмма состояния объекта процесса «Добавление цели».

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 нажимает кнопку «Мои данные», 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 «Цель» 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 «Цель» с доступными целями для добавления. Если он не нашел подходящую цель, то может создать свою нажав кнопку «Добавить цель».

2. Клиент заполняет все необходимые поля и нажимает кнопку «Добавить цель».

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 профиля 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, на которую может перейти с профиля, нажав на выбранную цель;

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 «Рейтинг», 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 «Создать тему»;

2. Клиент вводи название темы и нажимает кнопку «Создать».

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, перейдя по нужной теме или создав новую, отображающую форум отправки сообщения на форум;

2. Клиент заполняет поле ввод сообщения и нажимает кнопку «Добавить».

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.

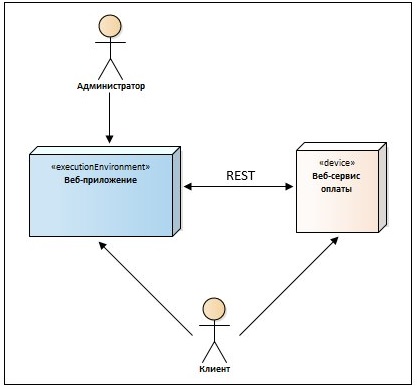
****

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)

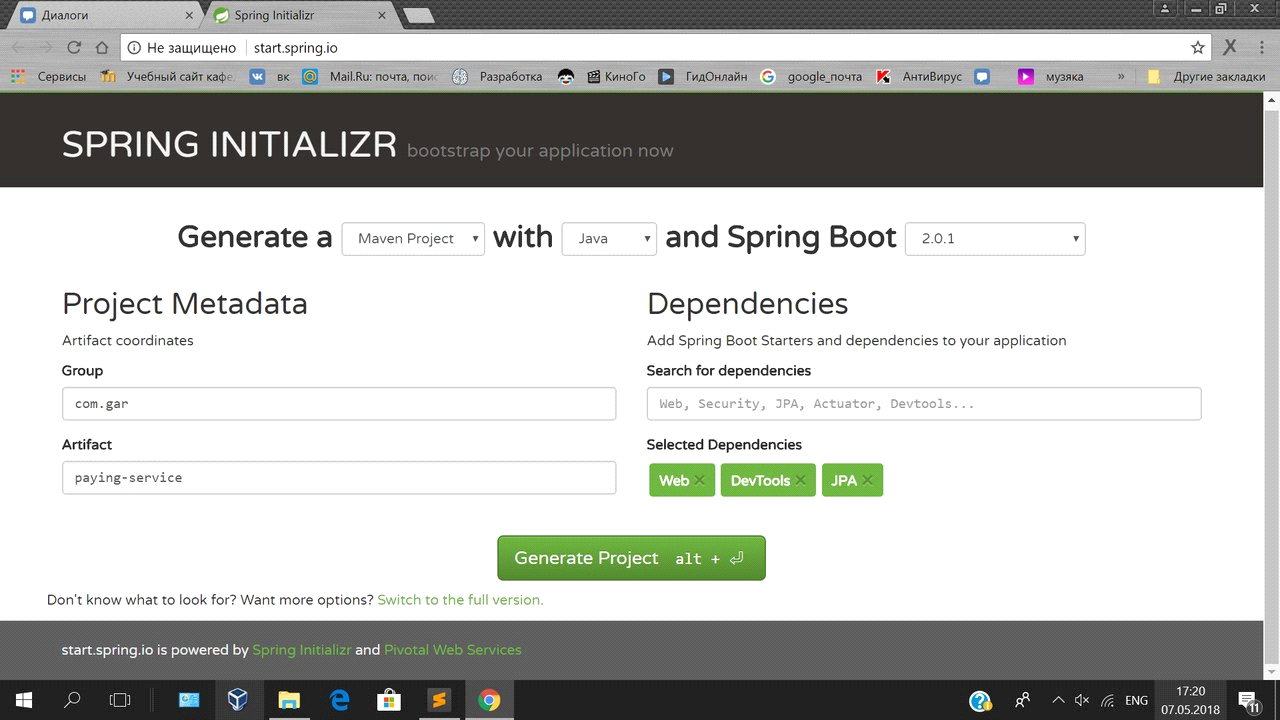


Рисунок 12. Создание Maven Project.

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

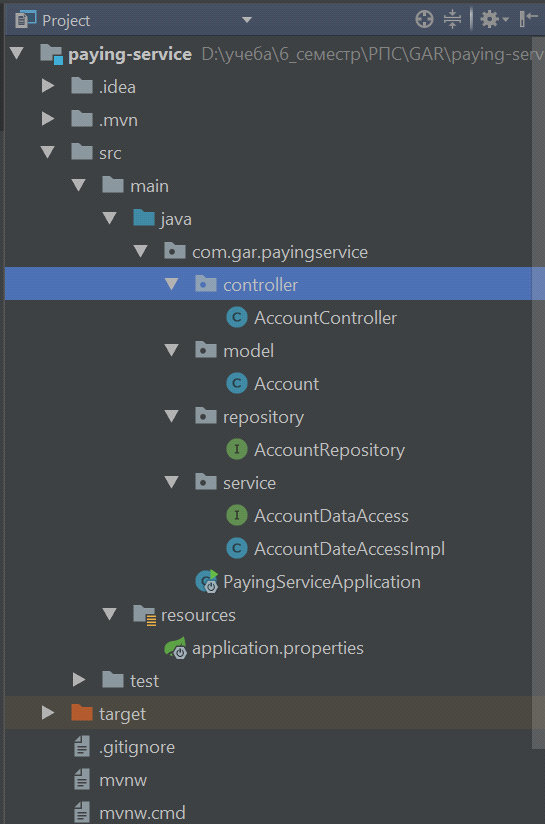


Рисунок 13. Структура проекта.

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

3.3 Организация доступа к данным

Для того, чтобы отразить, каким образом организован доступ к данным, используется ряд диаграмм: диаграмма классов уровня доступа к данным и диаграмма классов-сущностей, которые представлены ниже.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

Для отражения организации бизнес-логики используется диаграмма уровня бизнес-логики. Данная диаграмма для удобства восприятия разделена на несколько частей 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.

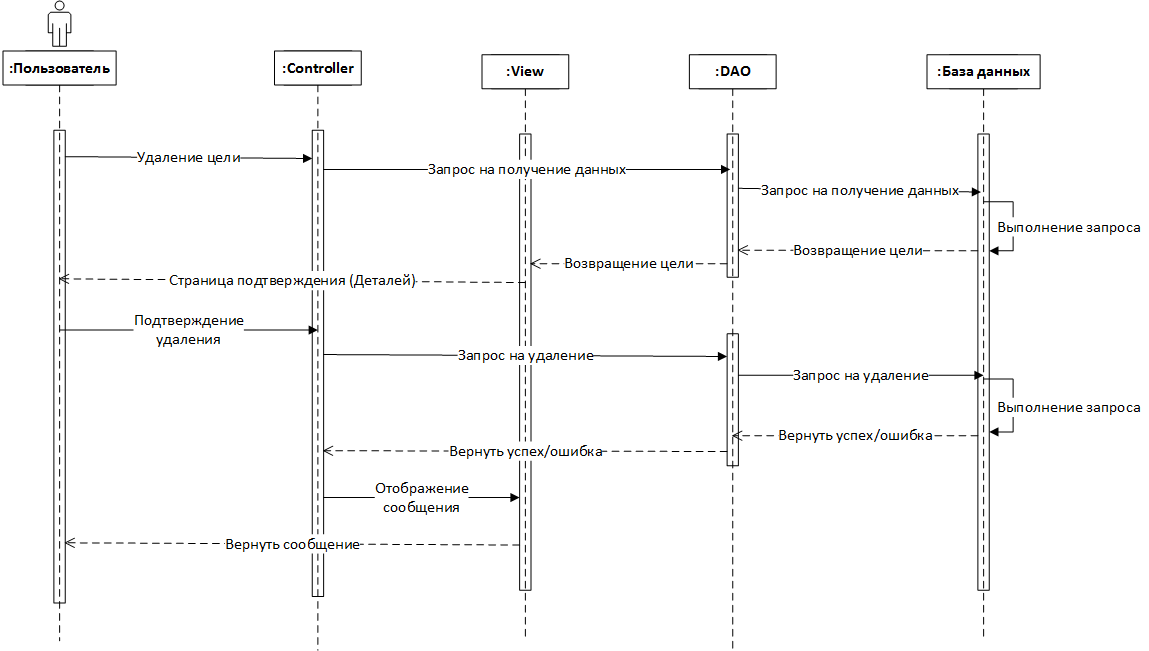


Рисунок 23. Диаграмма последовательностей для процесса «Удаление цели»

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.

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 № 1. 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 № 2

Листинг № 2. 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№ 3).

Листинг № 3. 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, а именно классы управляемых бинов, JSF-страницы, конфигурационные файлы.

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 № 4.

Листинг № 4. 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 № 5), а также листинг страницы authorization.jsf (листинг № 6).

Листинг № 5. 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 № 6. 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 № 7. 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 ….

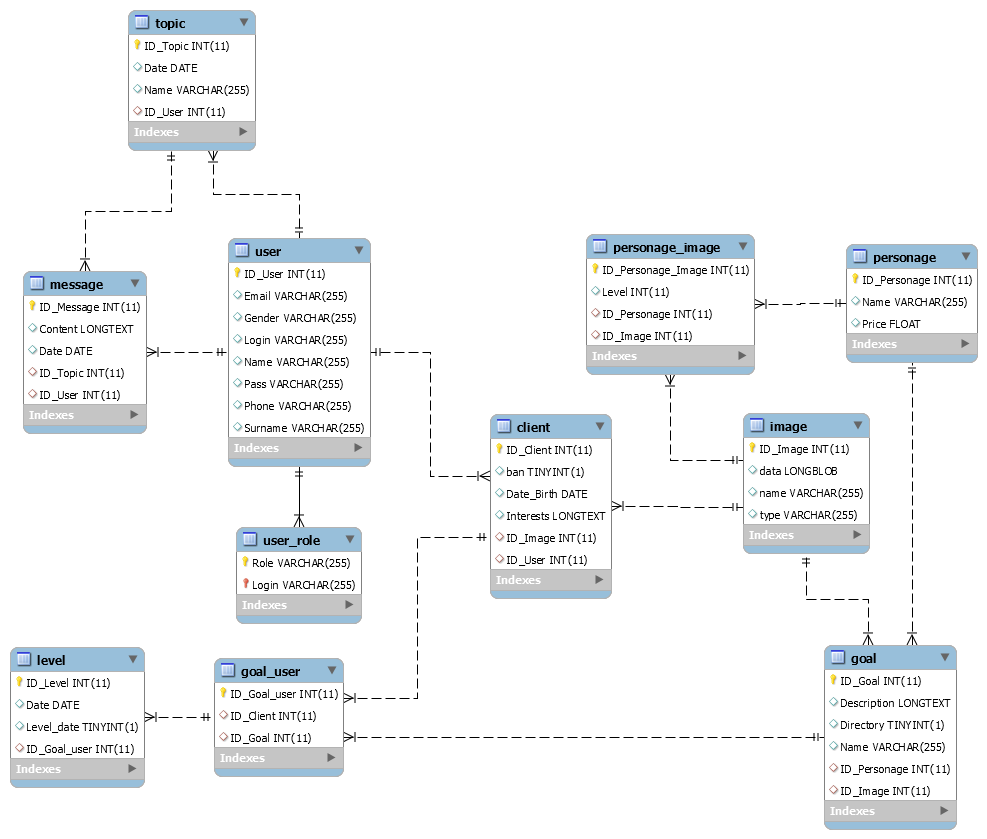


Рисунок 24.Физическая модель БД.

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 [2]. 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://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 (Рисунок 25):

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

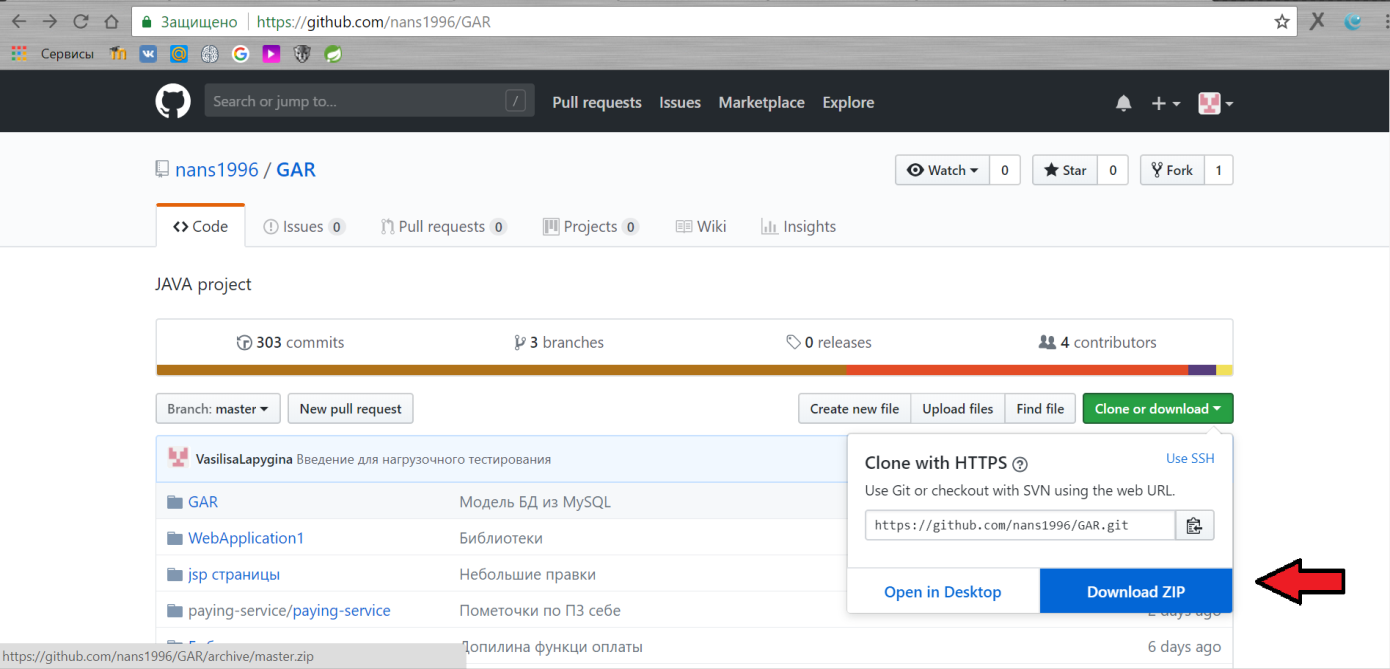


Рисунок 25. Скачивание ZIP архива с проектом.

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 (рисунок 26):

* Откройте терминал (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

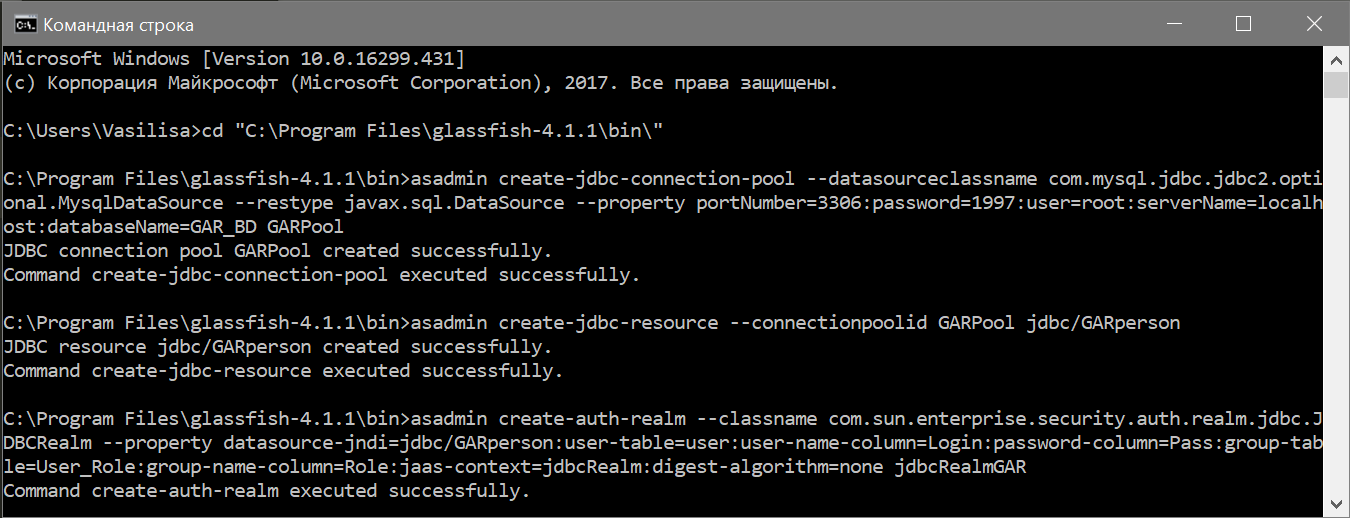


Рисунок 26. Создание подключения и настройка домена безопасности.

* Убедимся в успешности выполненных действий.

Для этого запустите GlassFish сервер

В адресную строку браузера введем: localhost:4848 (рисунок 27)

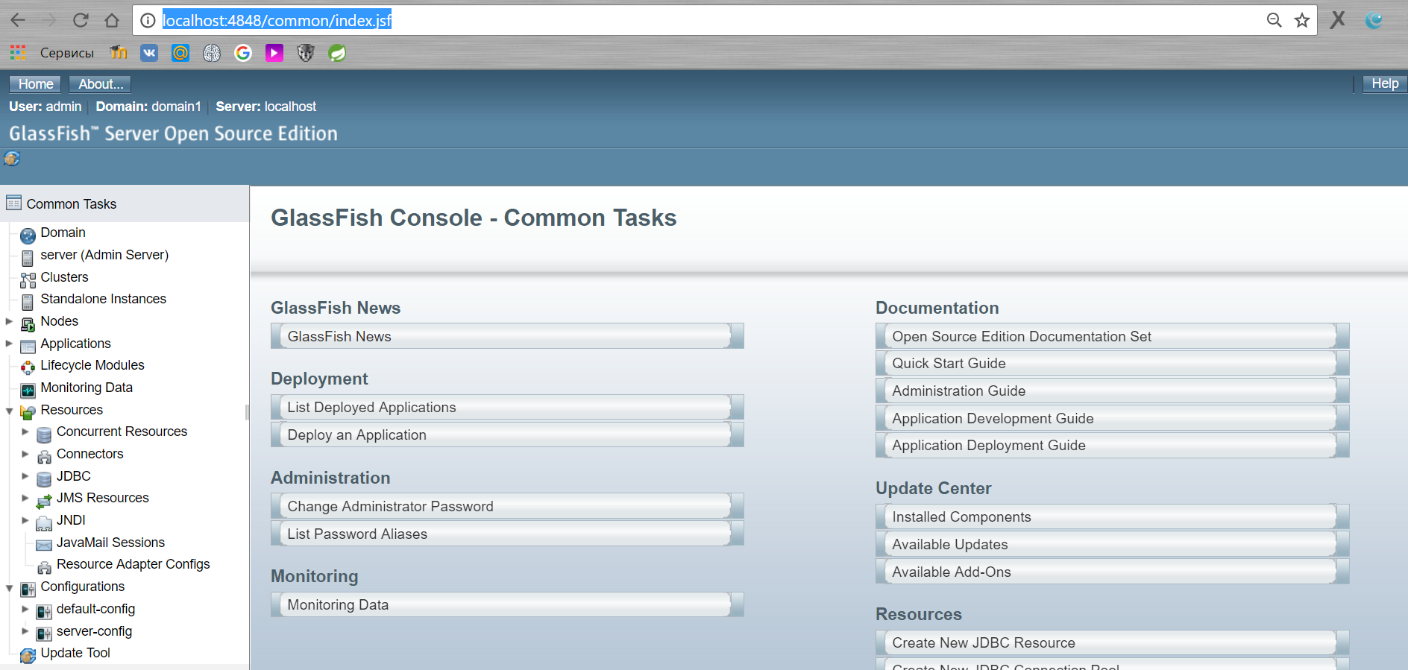


Рисунок 27. Админка сервера.

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

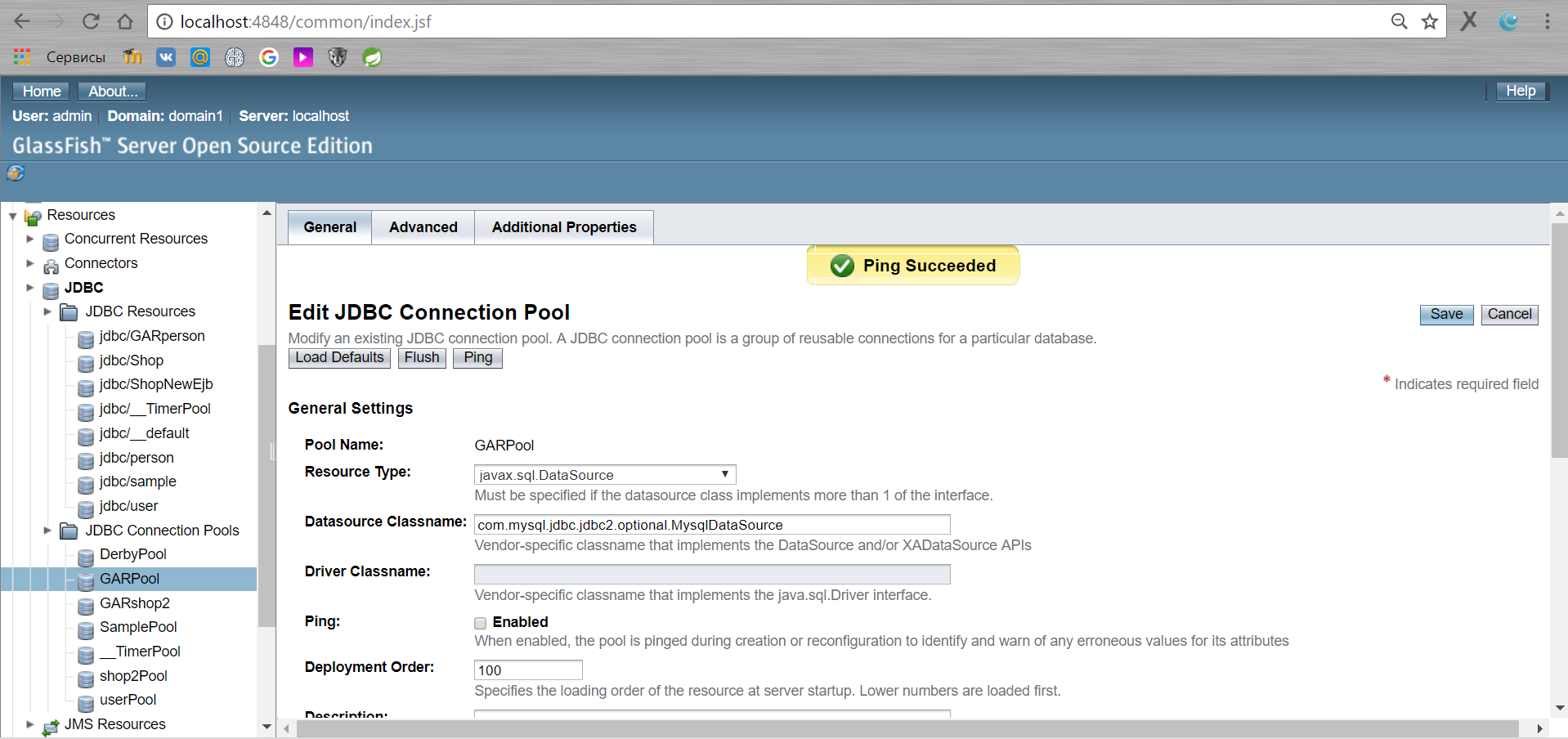


Рисунок 28. Установленное подключение.

Шаг 7: Открытие проекта

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

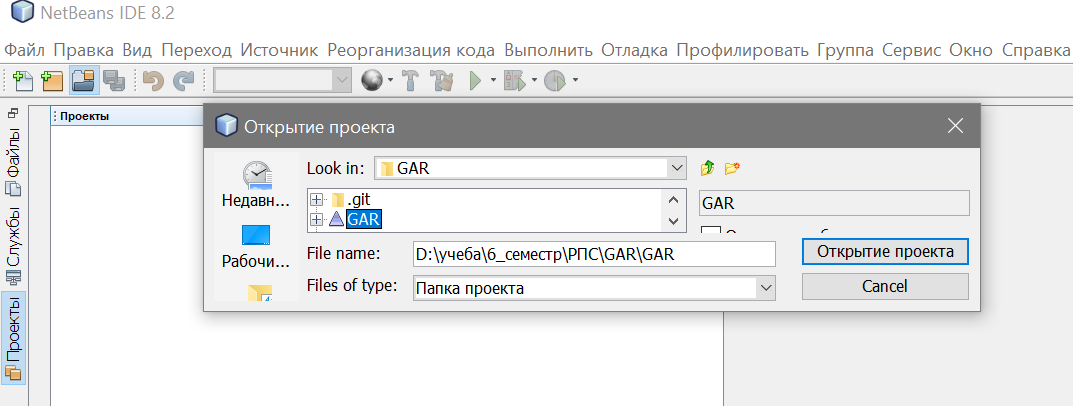


Рисунок 29. Открытие проекта.

* Откроем необходимые модули (рисунок 30)

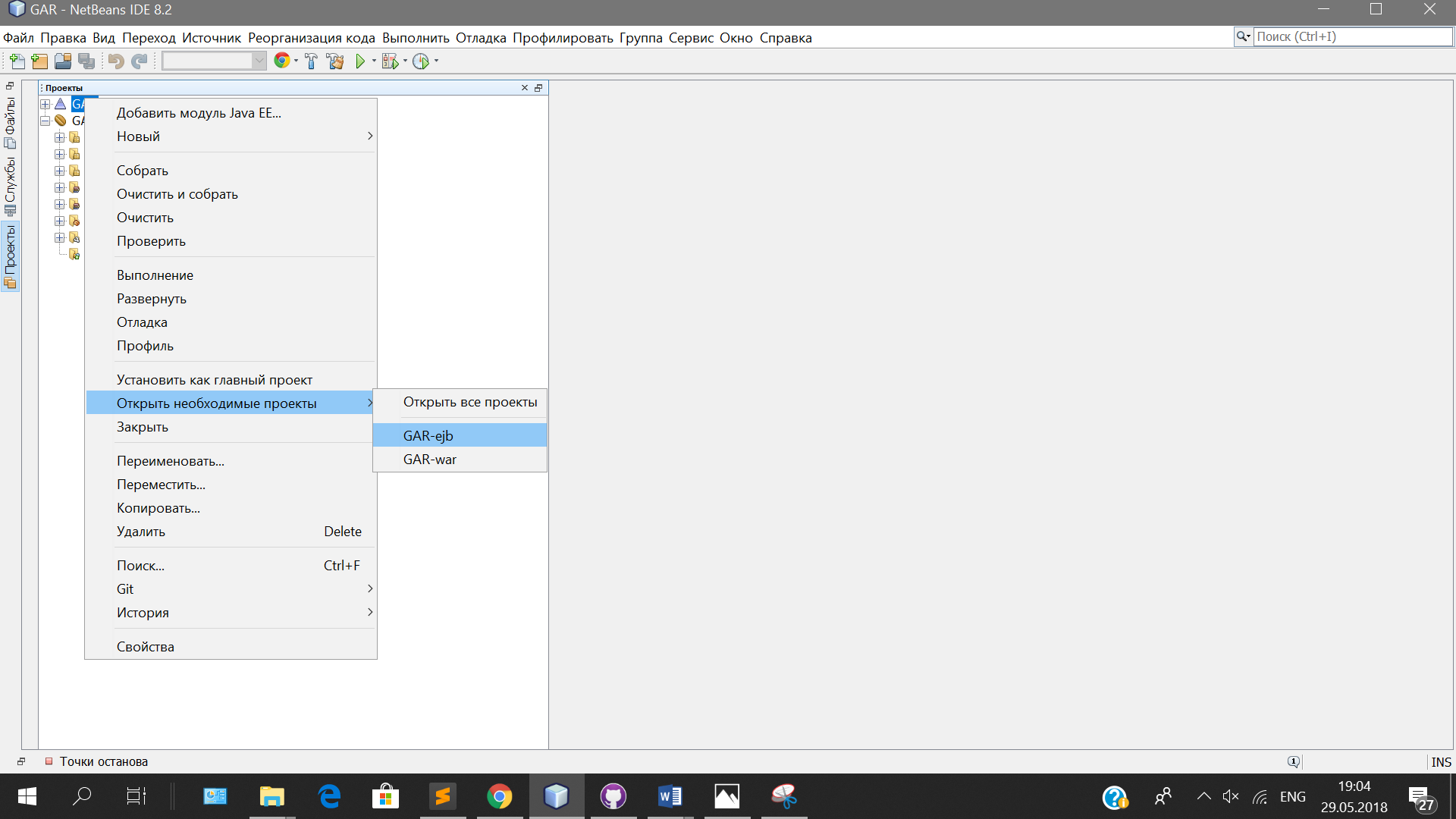


Рисунок 30. Открытие необходимых модулей.

Шаг 8: Деплой проекта

* В модуле persistence.xml укажем стратегию создания таблиц: Create. (рисунок 31)

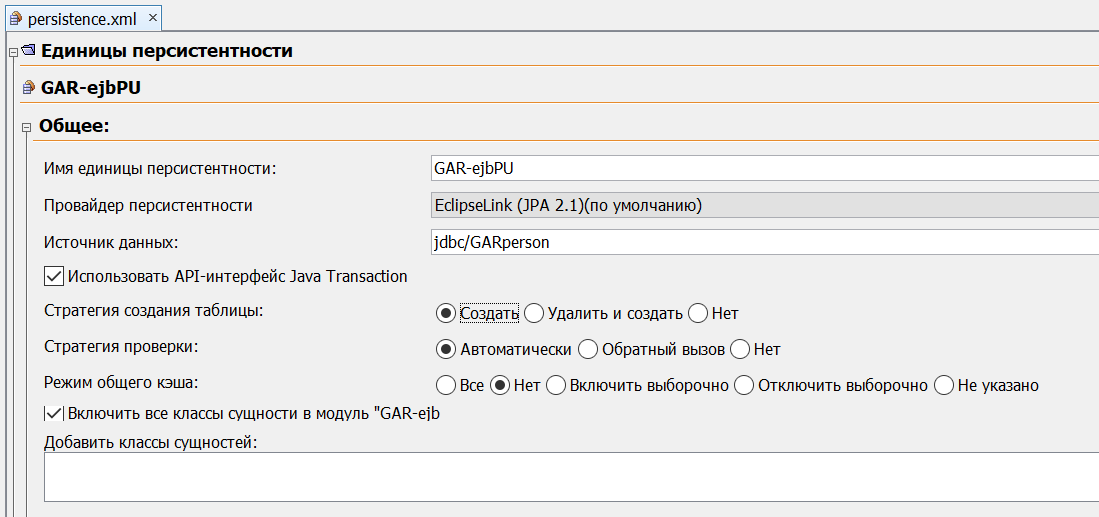


Рисунок 31. Модуль персистенстности.

* Развернём наш проект как показано на рисунке 32

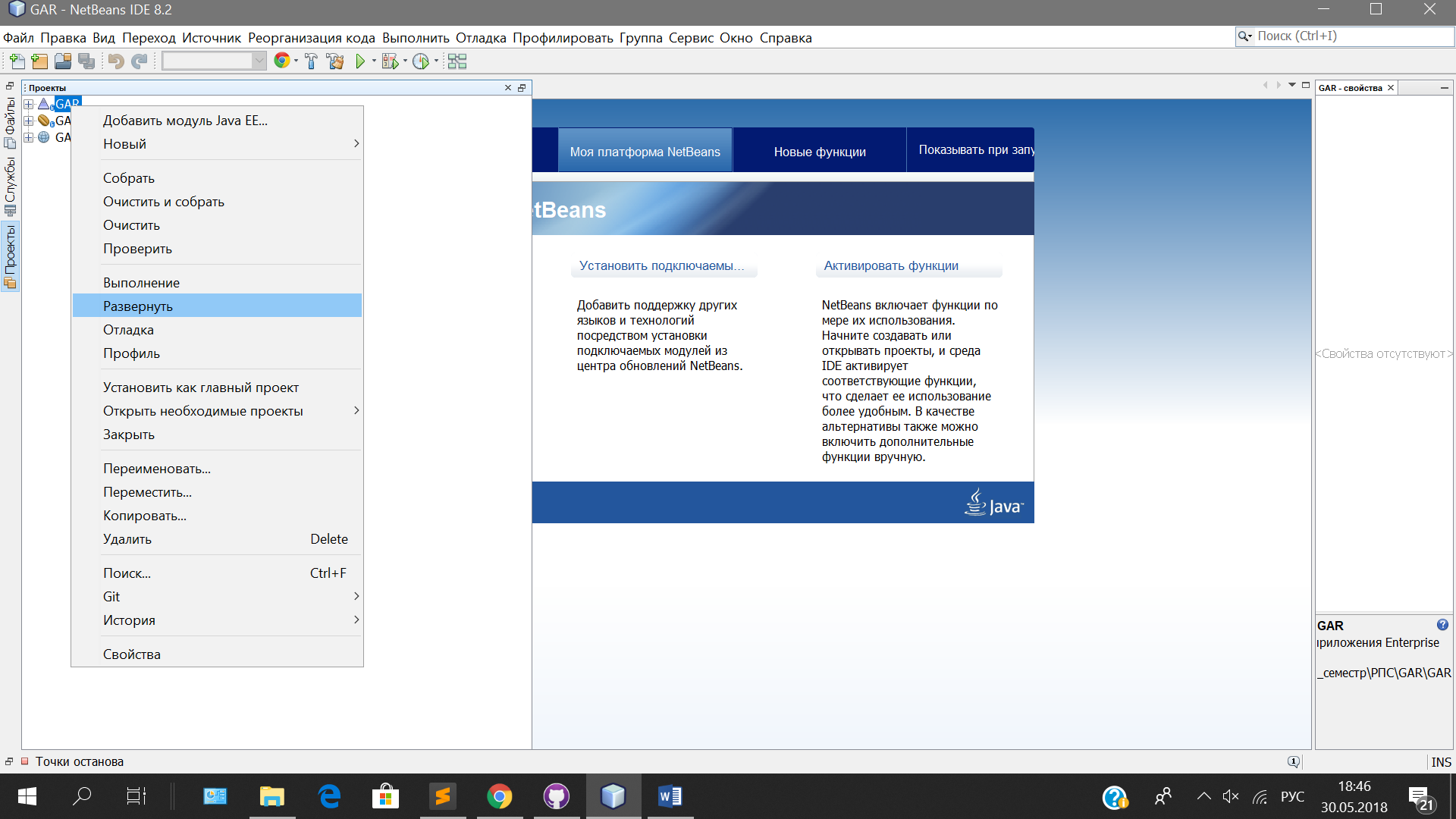


Рисунок 32. Разворачиваем Java EE приложение.

Шаг 9: Открытие проекта (Рисунок 33)

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

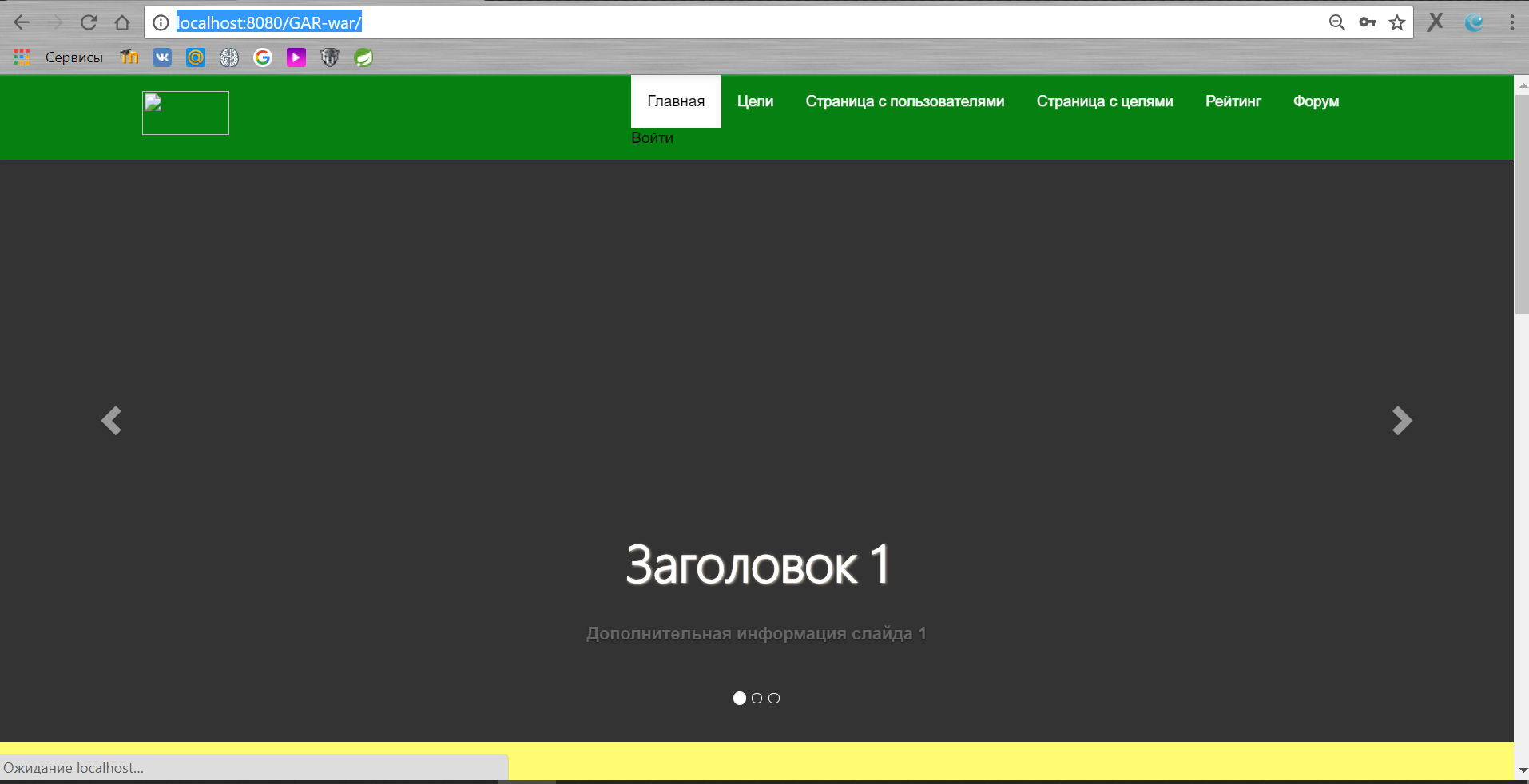


Рисунок 33. Запущенное Web-приложение.

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 (рисунок 34).

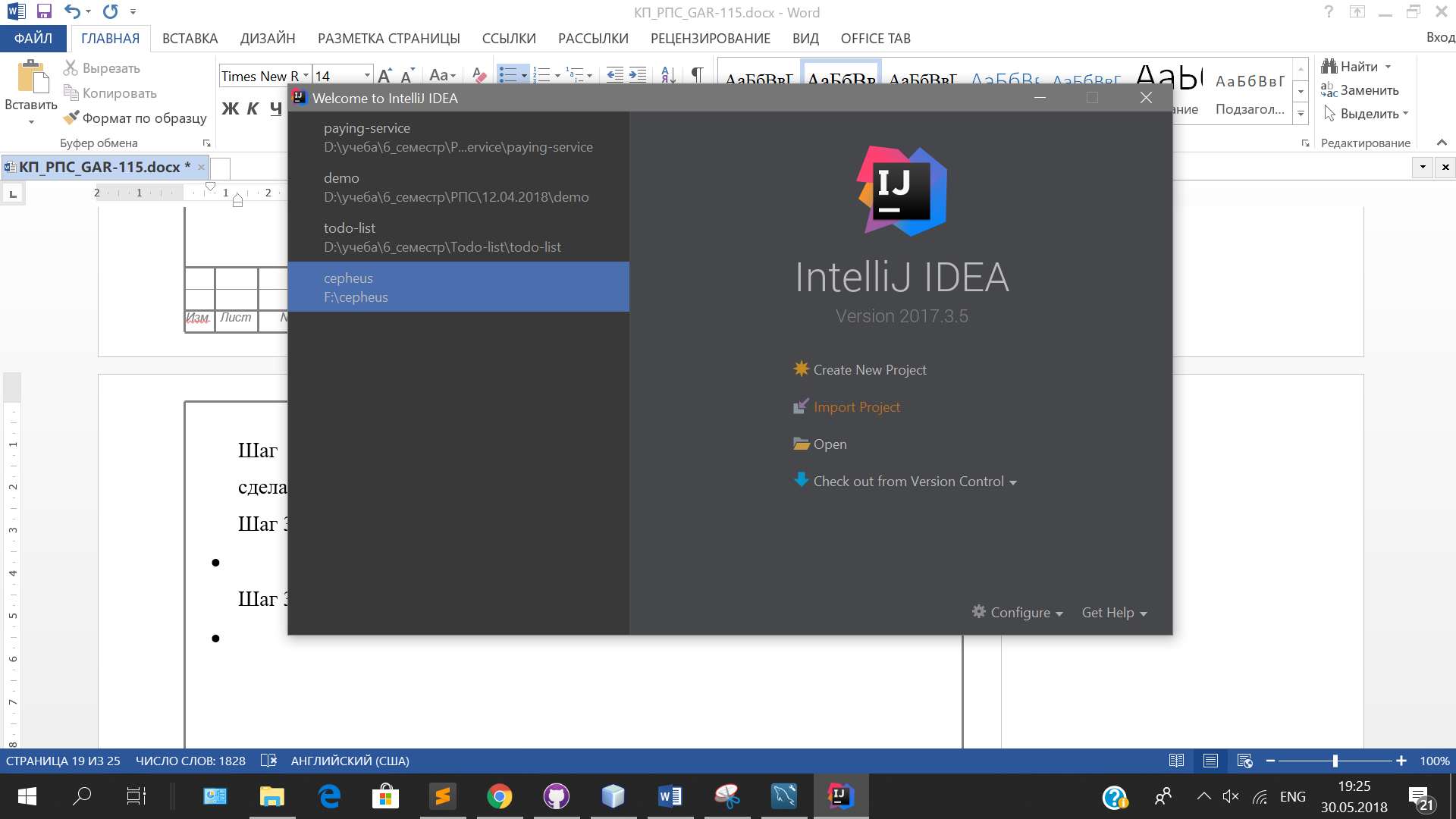


Рисунок 34. Импортируем проект.

* В проекте сервиса выбираем pom.properties (рисунок 35)

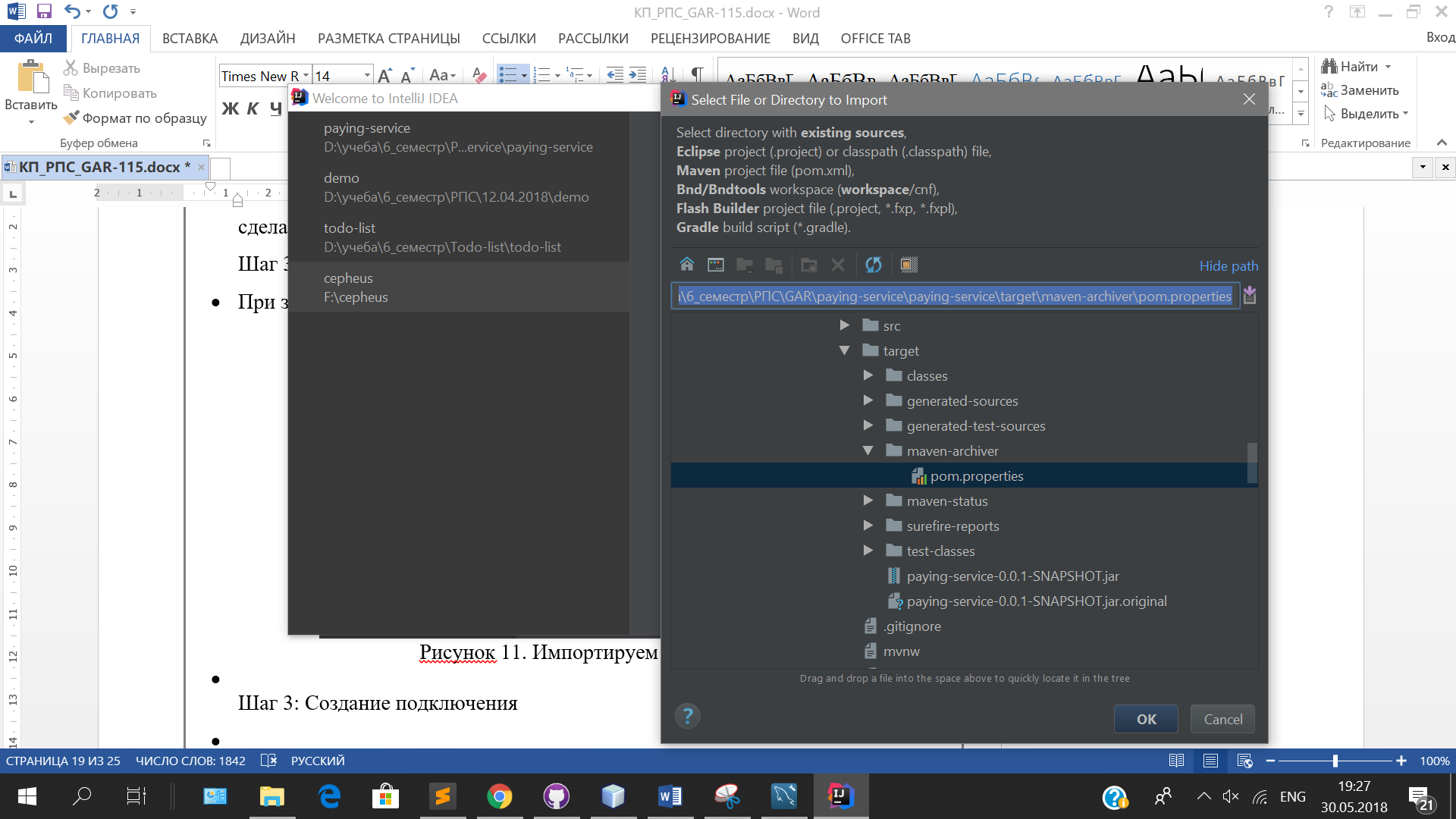


Рисунок 35. Импорт проекта.

* Подгруздка зависимостей. Устанавливаем галочку на против пункта: Import Maven project automaticall (рисунок 36)

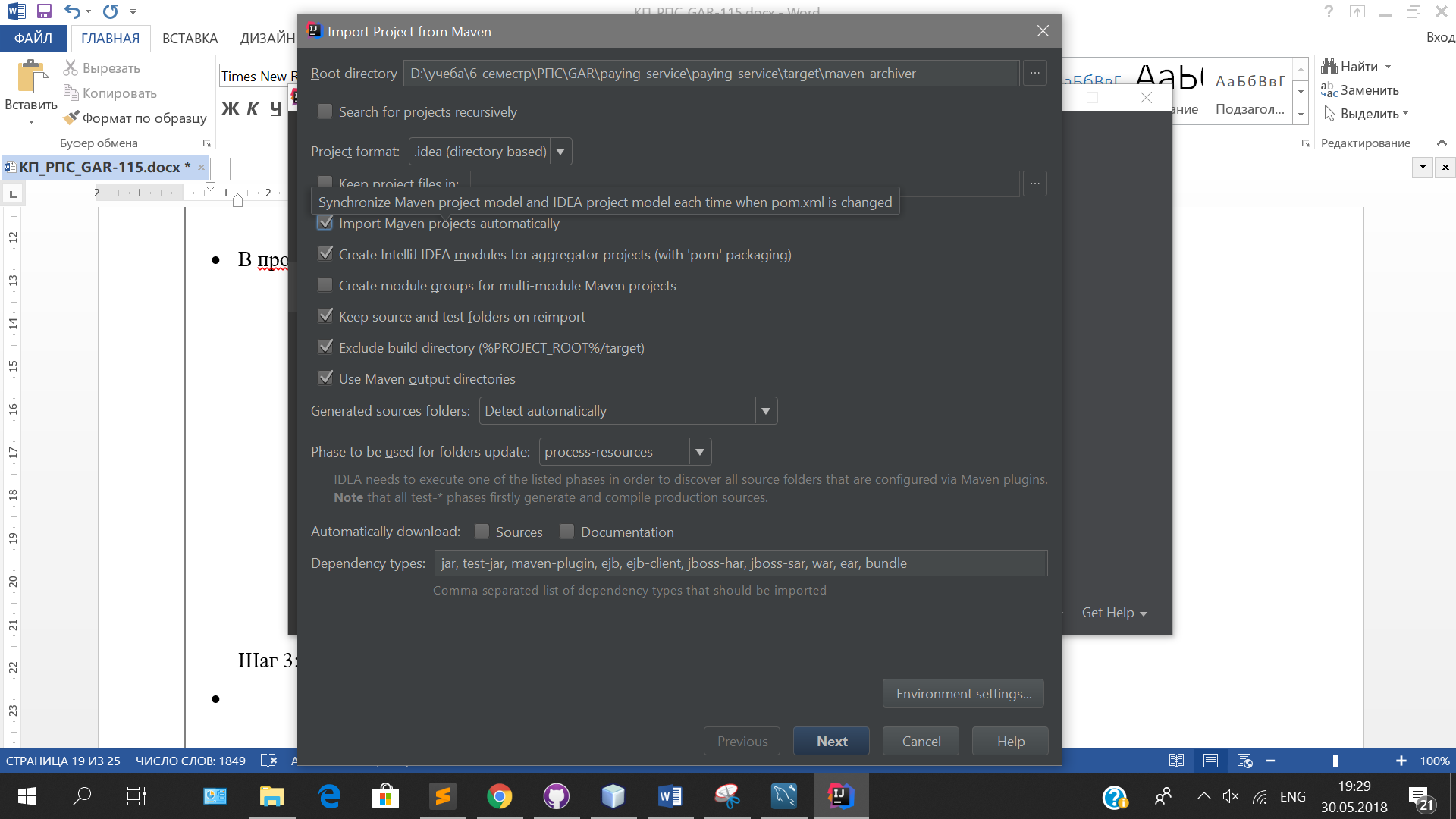


Рисунок 36. Настройка Maven проекта.

* Остальные настройки выполняем по желанию (в последнем пункте желательно добавить понятное вам имя проекта)

Шаг 4: Создание подключения (рисунок 37)

* 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.

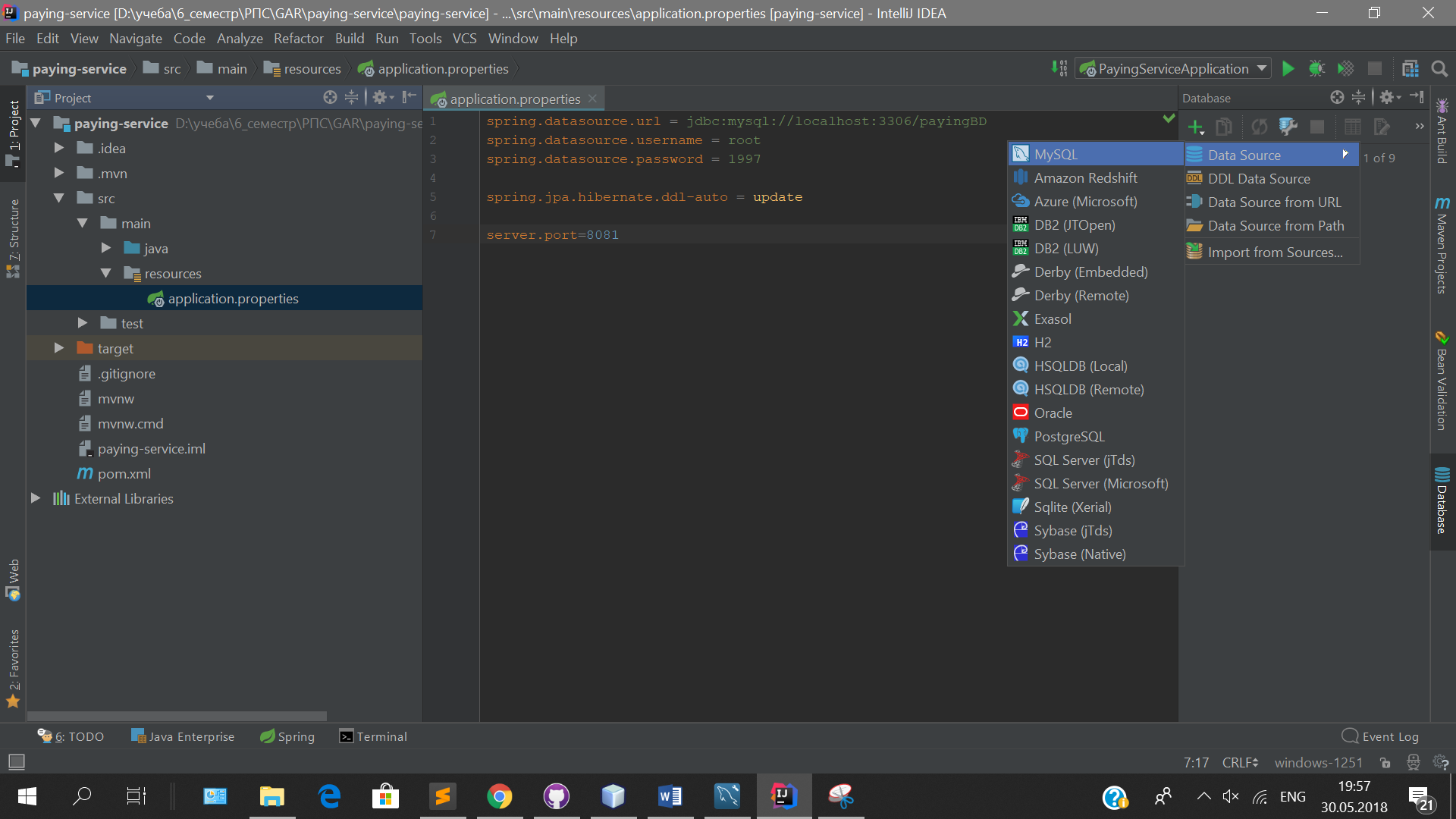


Рисунок 37. Создание подключения.

* Заполним необходимые поля и пропингуем подключение (рисунок 38).

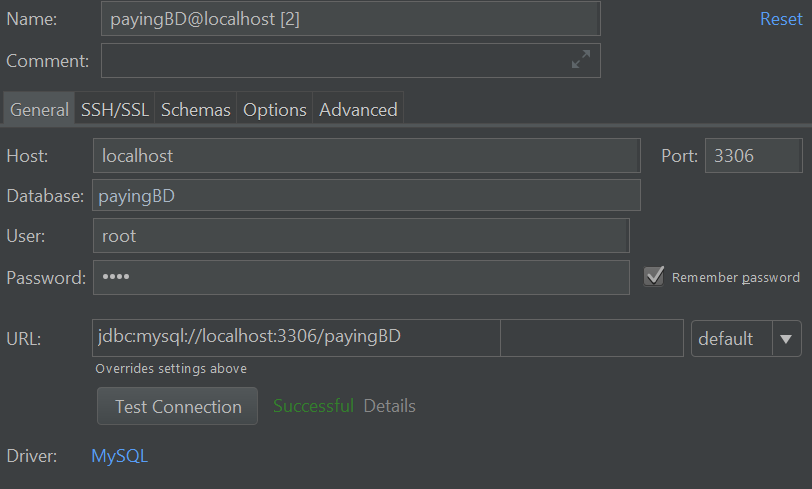


Рисунок 38. Создание подключения.

Шаг 5: Настройка информации о хранилище. Откроем файл application.properties и отредактируем в соответствии с этапами, выполненными ранее (рисунок 39)

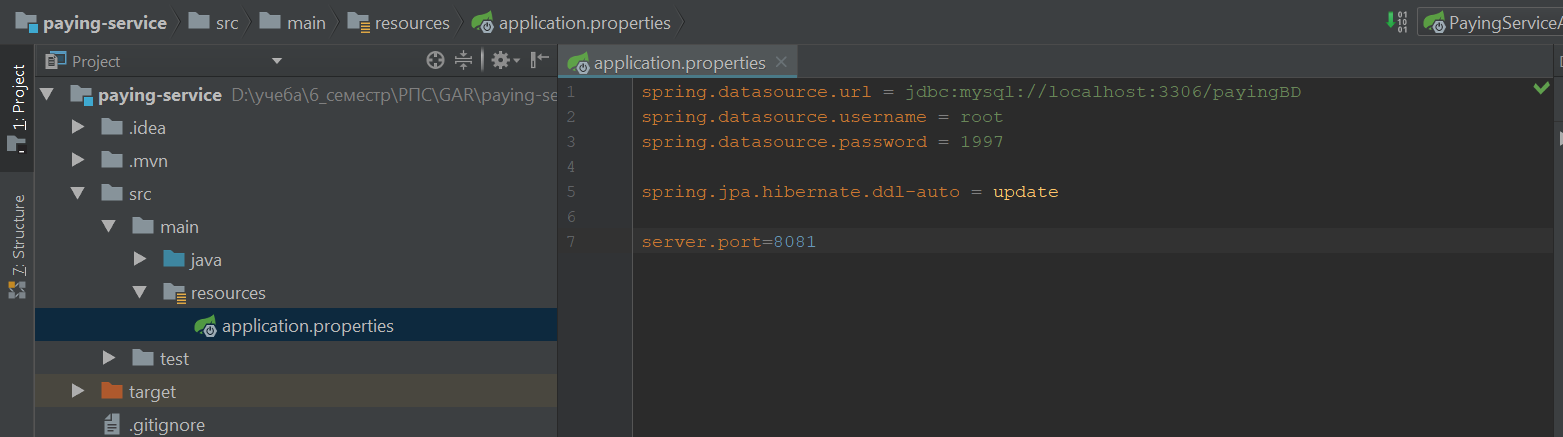


Рисунок 39. Application.properties.

Шаг 6: Сборка и разворачивание прокта (рисунок 40)

* На вкладке Maven Projects выполним команду собрать. После чего запустим наш проект.

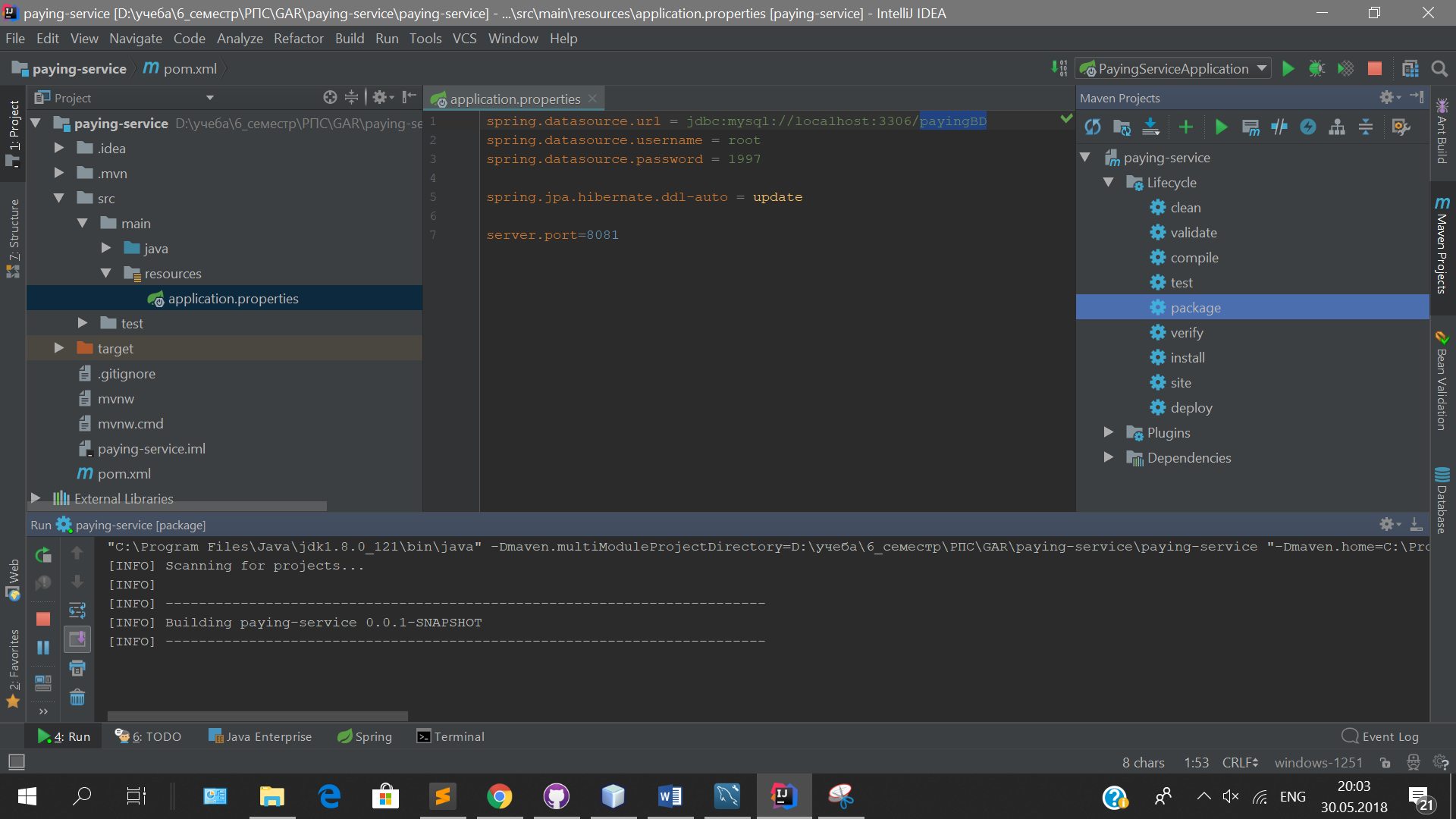


Рисунок 40. Сборка проекта.

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.

Благодаря принципам работы в команде, которых мы придерживались, наша работа была более четкой и эффективной.

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 (рисунок 41). На данной диаграмме представлен план выполнения работ, связанных с организацией нашего проекта.

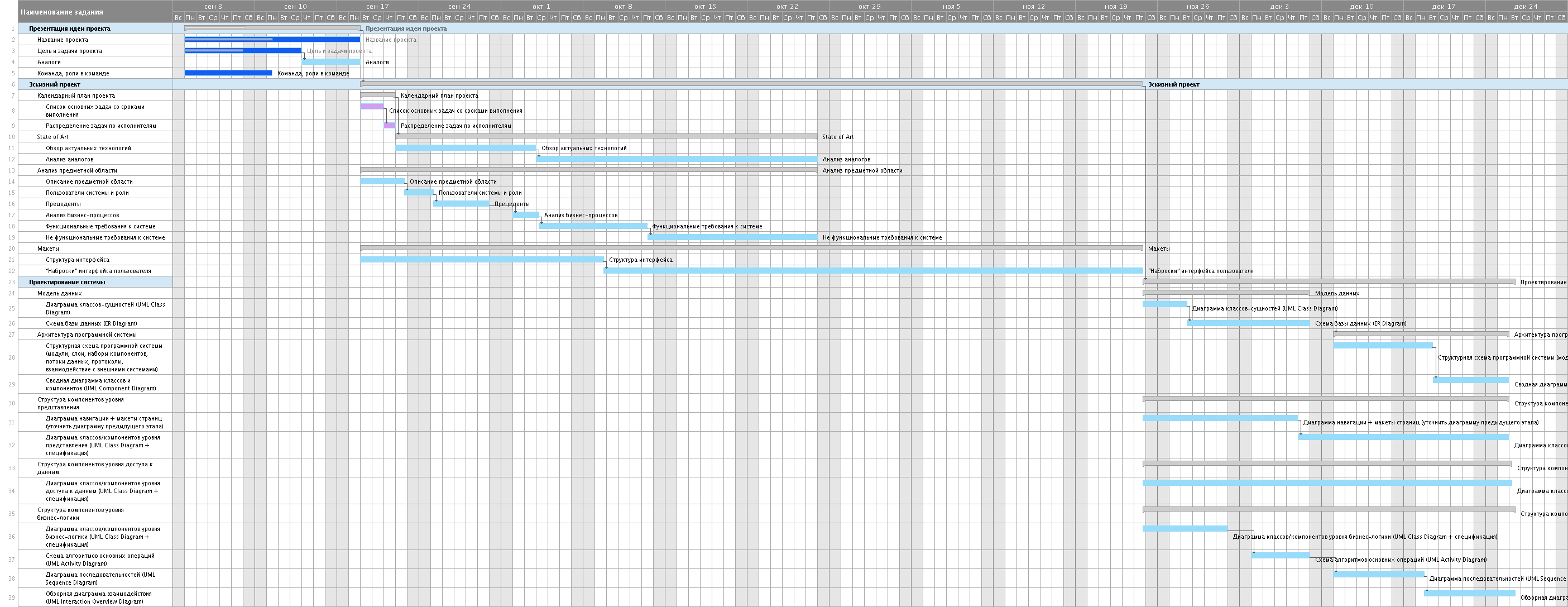


Рисунок 41. Диаграмма Ганта.

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).

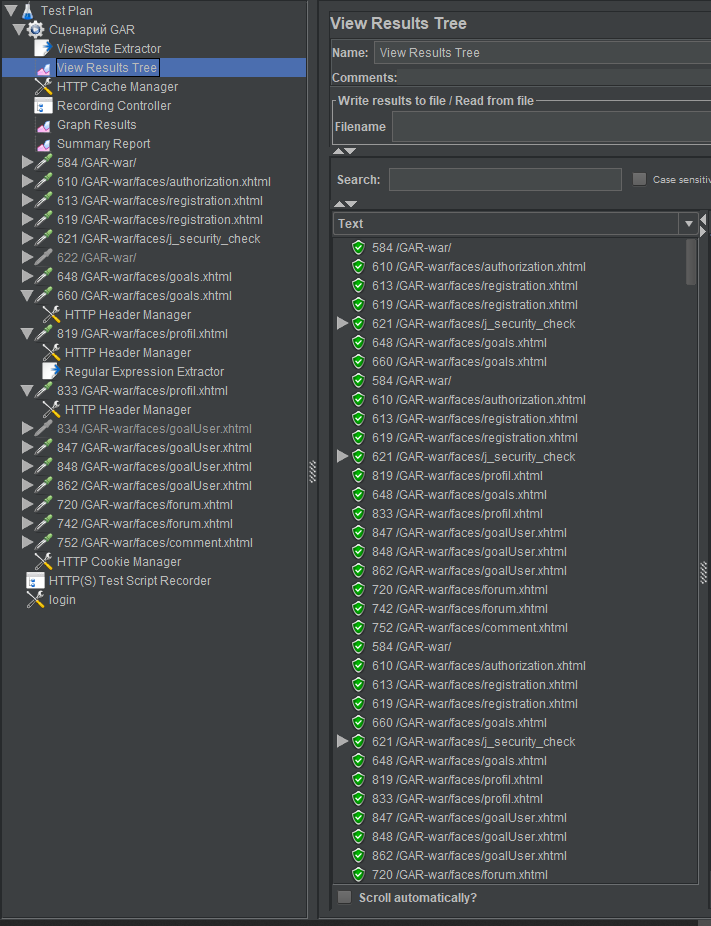


Рисунок 41. Структура.

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

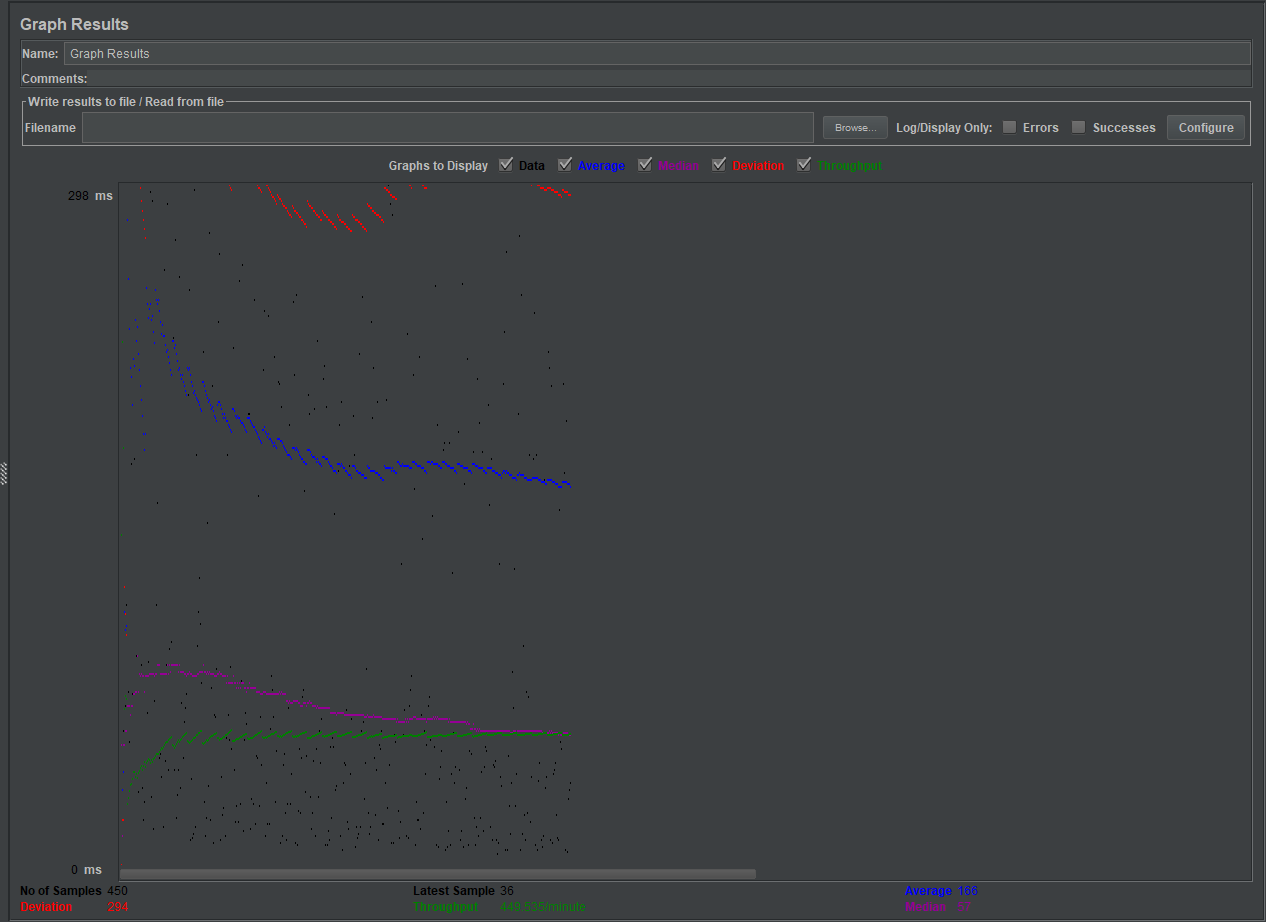


Рисунок 42. 30 пользователей в минуту.

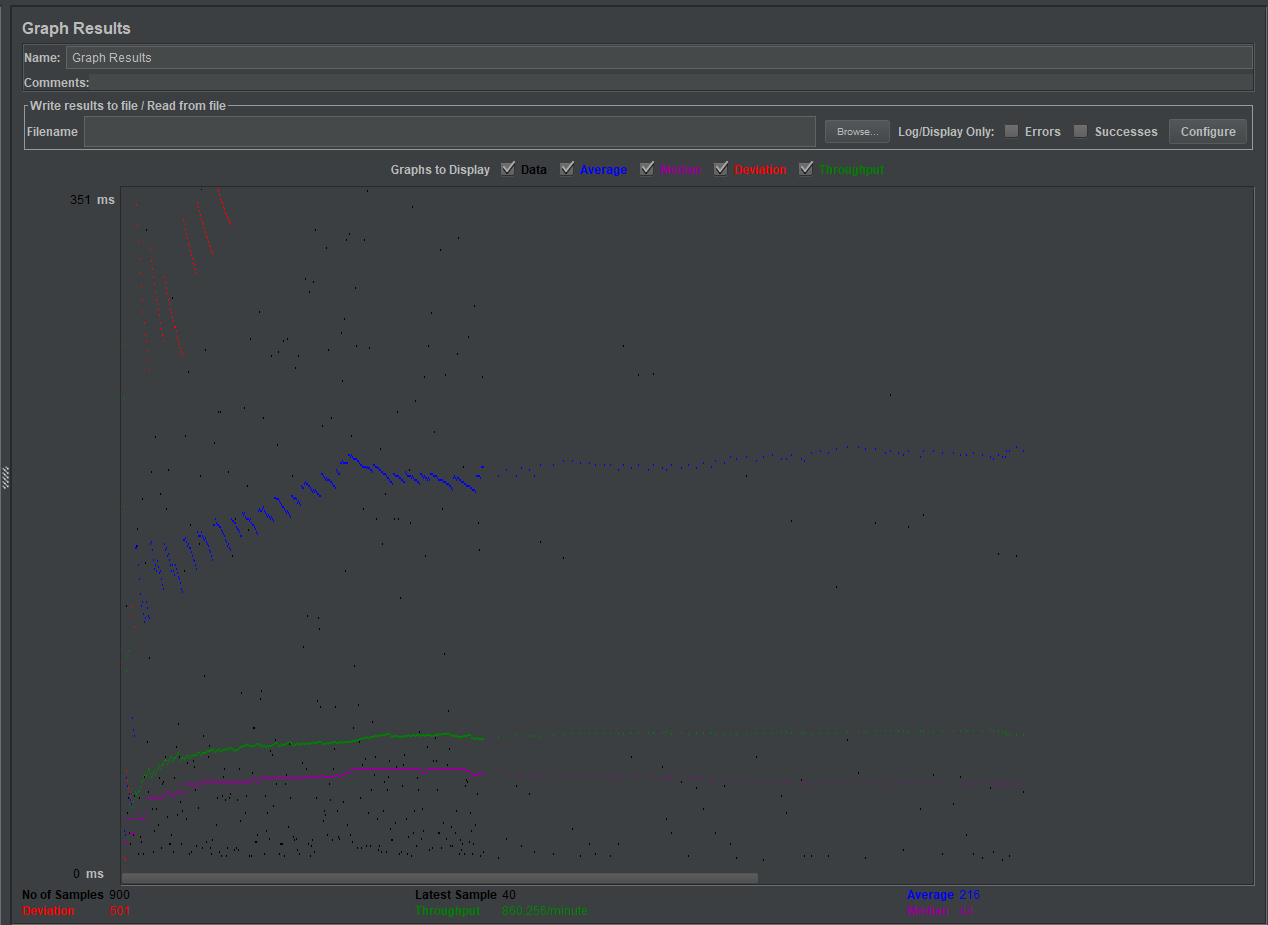


Рисунок 43. 60 пользователей в минуту.

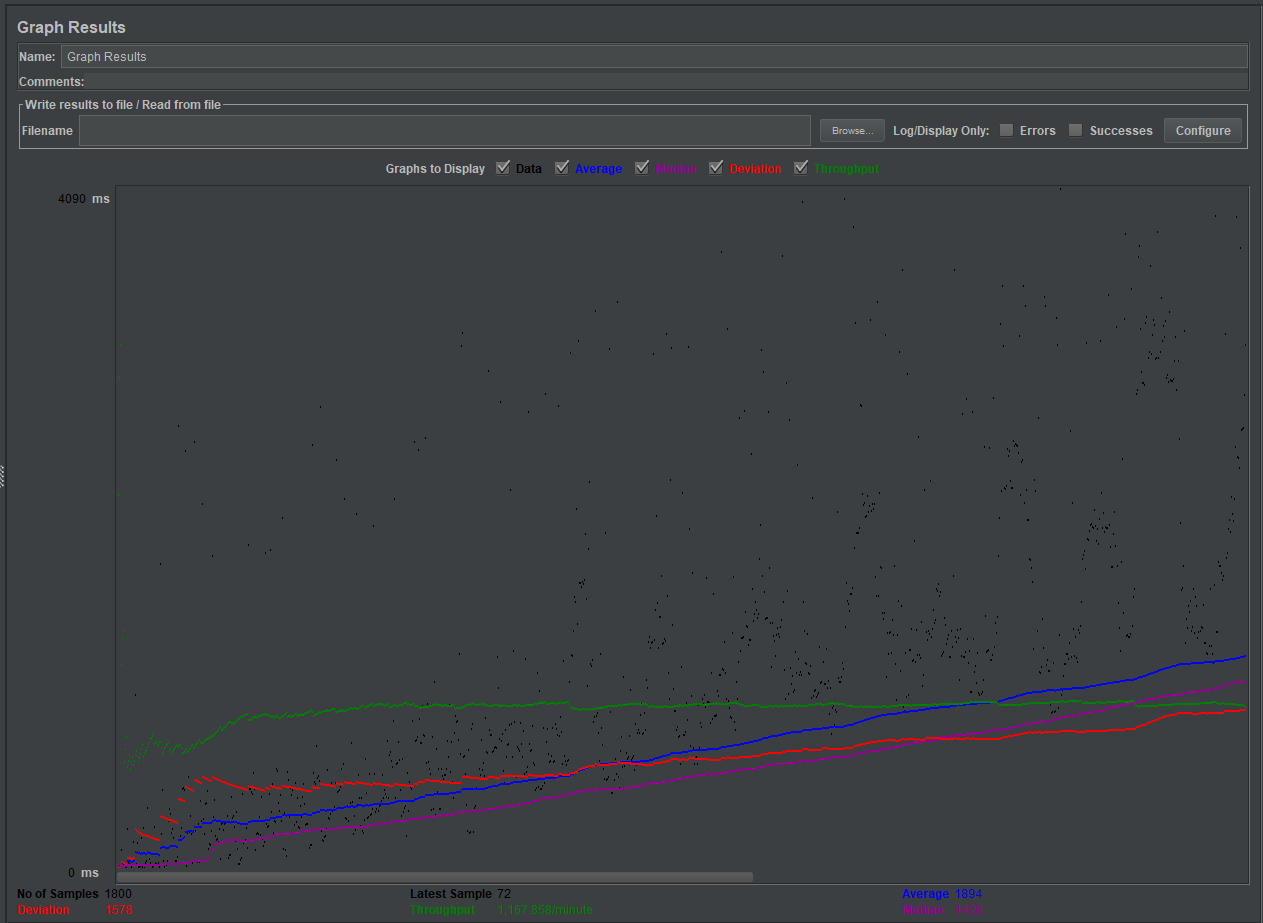


Рисунок 44. 120 пользователей в минуту.

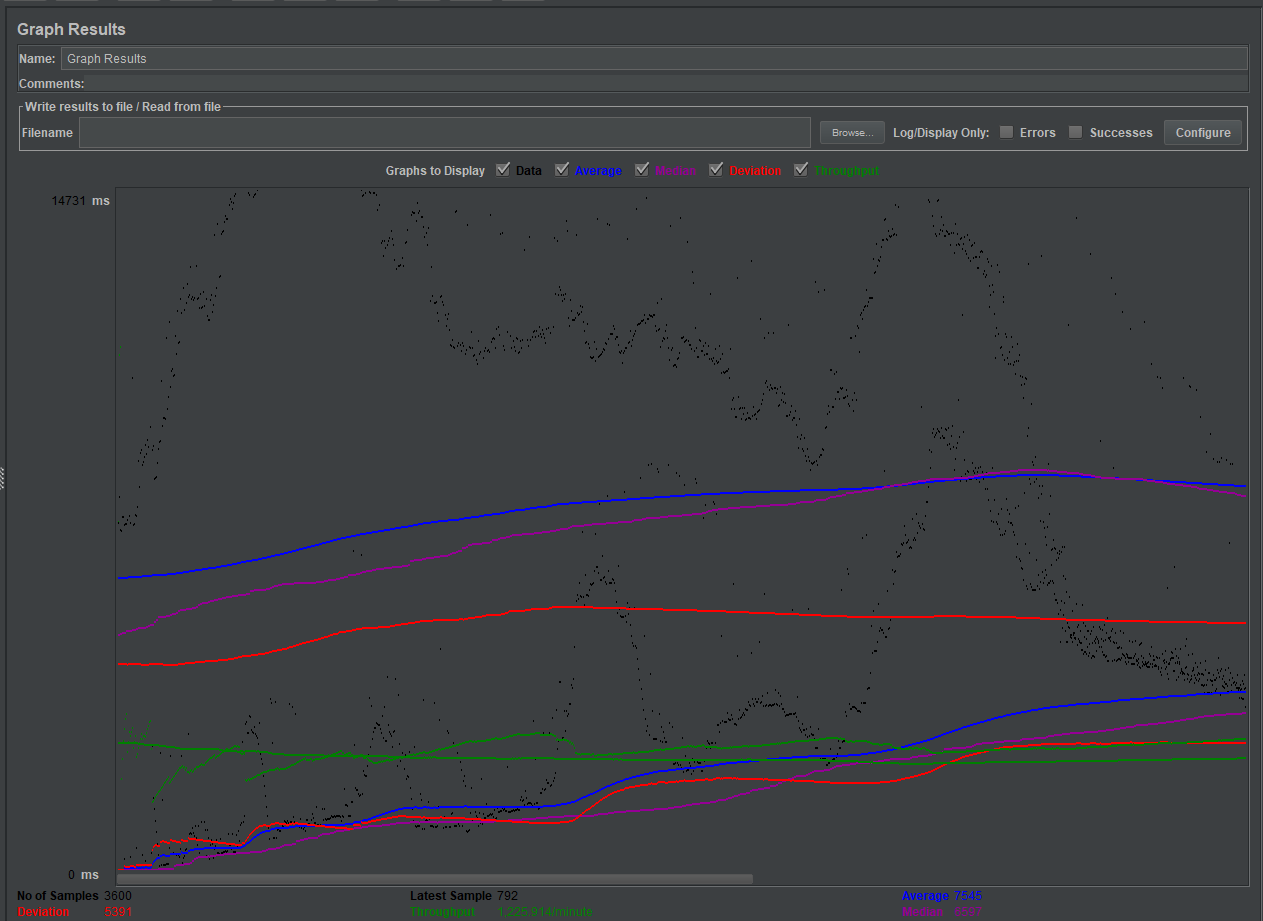


Рисунок 45. 240 пользователей в минуту.

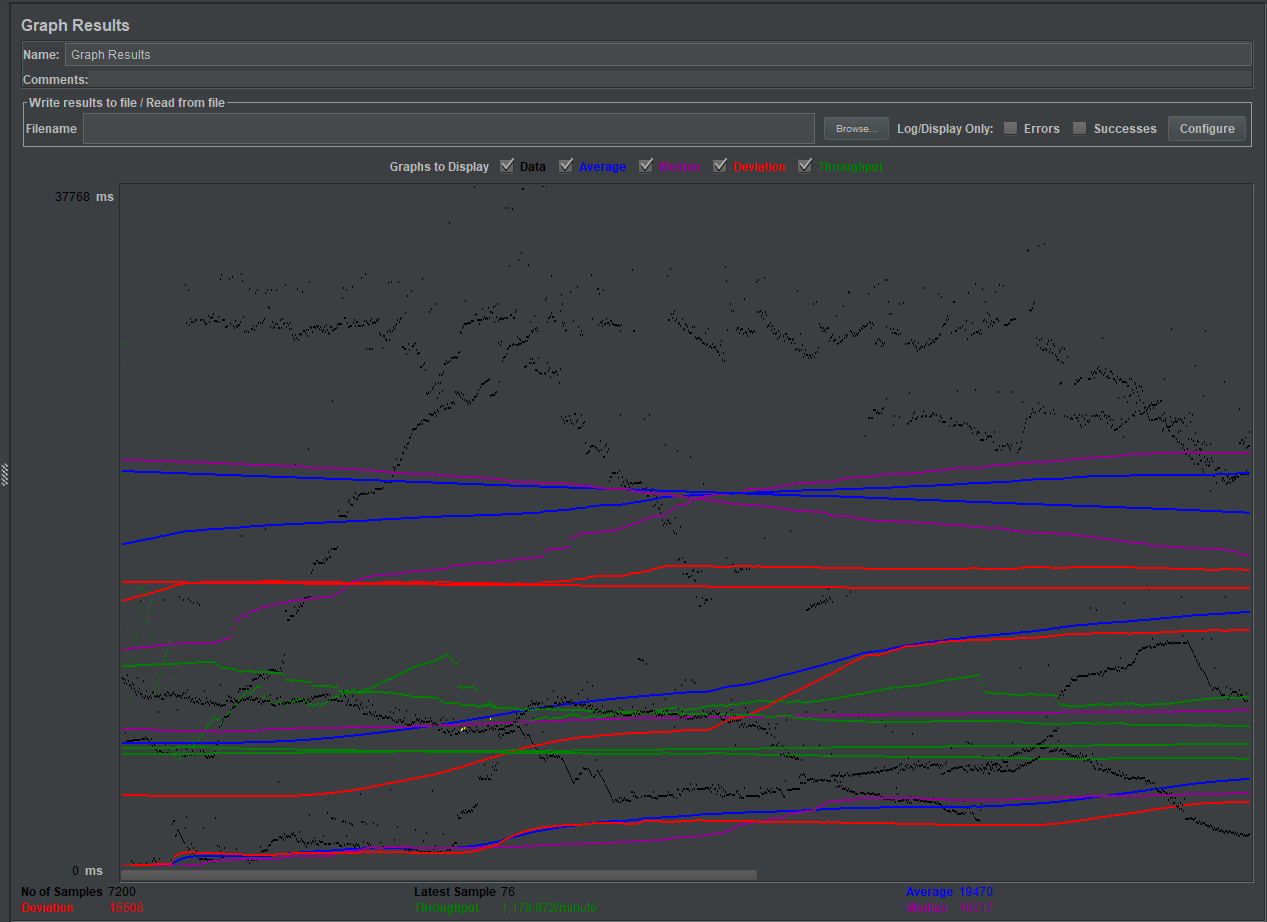


Рисунок 46. 480 пользователей в минуту

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 предел 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

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: 30.10.2017).

5. Bootstrap 3, документация на русском [электронный ресурс]. URL: http://bootstrap-3.ru/index.php (дата обращения: 20.03.2018).

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

7. Spring boot, документация [электронный ресурс]. 2018. URL: https://spring.io/guides/gs/spring-boot/ (дата обращения: 24.04.2018).

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. Apache JMeter: документация [электронный ресурс]. 2018. URL: https://jmeter.apache.org/usermanual/build-web-test-plan.html (дата обращения: 27.05.2018).

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:

Модуль EJB

* Пакет entities (содержит все сущности ):
* Client

Содержит методы get и set для своих полей

* Goal

Содержит методы get и set для своих полей

* GoalUser

Содержит методы get и set для своих полей

* Level

Содержит методы get и set для своих полей

* Message

Содержит методы get и set для своих полей

* Personage

Содержит методы get и set для своих полей

* Topic

Содержит методы get и set для своих полей

* User

Содержит методы get и set для своих полей

* UserRole

Содержит методы get и set для своих полей

* UserRolePK

Содержит методы get и set для своих полей

* Пакет model
* AbstractFacade – абстрактный класс
* public void create(T entity) – на вход приходит объект сущности, создается новая запись в базе данных
* public void edit(T entity) – на вход приходит объект сущности, обновляется запись в базе данных
* public void remove(T entity) – на вход приходит объект сущности, удаляется запись из базы данных
* public T find(Object id) – на вход приходит id, возвращается объект с соответствующим id
* public List<T> findAll() – вывод всех записей из таблицы БД
* ClientFacade – класс, соответствующий сущности client из базы данных, наследует AbstractFacade, реализует ClientFacadeLocal
* ClientFacadeLocal – локальный интерфейс, содержит методы:
* public void create(Client client) – на вход приходит объект сущности, создается новая запись в базе данных
* public void edit(Client client) – на вход приходит объект сущности, обновляется запись в базе данных
* public void remove(Client client) – на вход приходит объект сущности, удаляется запись из базы данных
* public T find(Object id) – на вход приходит id, возвращается объект с соответствующим id
* public List<Client> findAll() – вывод всех записей из таблицы БД
* GoalFacade - класс, соответствующий сущности goal из базы данных, наследует AbstractFacade, расширяет GoalFacadeLocal
* GoalFacadeLocal – локальный интерфейс, содержит методы:
* public void create(Goal goal) – на вход приходит объект сущности, создается новая запись в базе данных
* public void edit(Goal goal) – на вход приходит объект сущности, обновляется запись в базе данных
* public void remove(Goal goal) – на вход приходит объект сущности, удаляется запись из базы данных
* public T find(Object id) – на вход приходит id, возвращается объект с соответствующим id
* public List<Goal> findAll() – вывод всех записей из таблицы БД
* GoalUserFacade - класс, соответствующий сущности goal\_user из базы данных, наследует AbstractFacade, расширяет GoalUserFacadeLocal
* GoalUserFacadeLocal - локальный интерфейс, содержит методы:
* public void create(GoalUser goaluser) – на вход приходит объект сущности, создается новая запись в базе данных
* public void edit(GoalUser goaluser) – на вход приходит объект сущности, обновляется запись в базе данных
* public void remove(GoalUser goaluser) – на вход приходит объект сущности, удаляется запись из базы данных
* public T find(Object id) – на вход приходит id, возвращается объект с соответствующим id
* public List<GoalUser> findAll() – вывод всех записей из таблицы БД
* LevelFacade - класс, соответствующий сущности level из базы данных, наследует AbstractFacade, реализует LevelFacadeLocal
* LevelFacadeLocal - локальный интерфейс, содержит методы:
* public void create(Level level) – на вход приходит объект сущности, создается новая запись в базе данных
* public void edit(Level level) – на вход приходит объект сущности, обновляется запись в базе данных
* public void remove(Leevl level) – на вход приходит объект сущности, удаляется запись из базы данных
* public T find(Object id) – на вход приходит id, возвращается объект с соответствующим id
* public List<Level> findAll() – вывод всех записей из таблицы БД
* MessageFacade - класс, соответствующий сущности message из базы данных, наследует AbstractFacade, реализует MessageFacadeLocal
* MessageFacadeLocal - локальный интерфейс, содержит методы:
* public void create(Message message) – на вход приходит объект сущности, создается новая запись в базе данных
* public void edit(Message message )– на вход приходит объект сущности, обновляется запись в базе данных
* public void remove(Message message) – на вход приходит объект сущности, удаляется запись из базы данных
* public T find(Object id) – на вход приходит id, возвращается объект с соответствующим id
* public List<Message> findAll() – вывод всех записей из таблицы БД
* PersonageFacade - класс, соответствующий сущности personage из базы данных, наследует AbstractFacade, реализует PersonageFacadeLocal
* PersonageFacadeLocal - локальный интерфейс, содержит методы:
* public void create(Personage personage) – на вход приходит объект сущности, создается новая запись в базе данных
* public void edit(Personage personage)– на вход приходит объект сущности, обновляется запись в базе данных
* public void remove(Personage personage) – на вход приходит объект сущности, удаляется запись из базы данных
* public T find(Object id) – на вход приходит id, возвращается объект с соответствующим id
* public List< Personage> findAll() – вывод всех записей из таблицы БД
* TopicFacade - класс, соответствующий сущности topic из базы данных, наследует AbstractFacade, реализует TopicFacadeLocal
* TopicFacadeLocal - локальный интерфейс, содержит методы:
* public void create(Topic topic) – на вход приходит объект сущности, создается новая запись в базе данных
* public void edit(Topic topic)– на вход приходит объект сущности, обновляется запись в базе данных
* public void remove(Topic topic) – на вход приходит объект сущности, удаляется запись из базы данных
* public T find(Object id) – на вход приходит id, возвращается объект с соответствующим id
* public List<Topic> findAll() – вывод всех записей из таблицы БД
* UserFacade - класс, соответствующий сущности topic из базы данных, наследует AbstractFacade, реализует UserFacadeLocal
* UserFacadeLocal - локальный интерфейс, содержит методы:
* public void create(User user) – на вход приходит объект сущности, создается новая запись в базе данных
* public void edit(User user)– на вход приходит объект сущности, обновляется запись в базе данных
* public void remove(User user) – на вход приходит объект сущности, удаляется запись из базы данных
* public T find(Object id) – на вход приходит id, возвращается объект с соответствующим id
* public List<User> findAll() – вывод всех записей из таблицы БД
* UserRoleFacade - класс, соответствующий сущности topic из базы данных, наследует AbstractFacade, реализует UserRoleFacadeLocal
* UserRoleFacadeLocal - локальный интерфейс, содержит методы:
* public void create(UserRole userrole) – на вход приходит объект сущности, создается новая запись в базе данных
* public void edit(UserRole userrole)– на вход приходит объект сущности, обновляется запись в базе данных
* public void remove(UserRole userrole) – на вход приходит объект сущности, удаляется запись из базы данных
* public T find(Object id) – на вход приходит id, возвращается объект с соответствующим id
* public List<UserRole> findAll() – вывод всех записей из таблицы БД

Модуль WAR

* ForumBean – бин форума
* public void SendMessage() – метод отправки сообщений
* public String messageIdTopic()– получение списка сообщений по теме

обрабатываемое исключение: EJBException

* public List<Topic> getAllTopics() - получение списка тем

обрабатываемое исключение: EJBException

* public List<Message> getAllMessage() – получение списка сообщений

обрабатываемое исключение: EJBException

* public String deleteMessage(Message message) – удаление сообщения

обрабатываемое исключение: EJBException

* public String createTopic() – создание темы

обрабатываемое исключение: EJBException

* public String createMessage() – создание сообщения

обрабатываемое исключение: EJBException

* public String ban() – назначить или снять бан на форуме

обрабатываемое исключение: EJBException

* set и get методы для полей
* AdministratorBean:
* public List<User> getAllUser() – вывод списка пользователей
* public StreamedContent getImageCreate() – вывод только что добавленной картинки

обрабатываемое исключение: IOException

* public byte[] InputStreamToArryByte(InputStream in) – вывод картинки в байтовый массив

обрабатываемое исключение: IOException

* public void uploadImagePersonage() – добавление картинки персонажу на уровень

обрабатываемое исключение: EJBException

* public List<String> levelFild() – генерация списка уровней для добавления
* public String ban() – назначить или снять бан пользователя

обрабатываемое исключение: EJBException

* public String addAdmin() – назначение прав администратора

обрабатываемое исключение: EJBException

* public List<Goal> getAllGoal() – вывод общего списка целей
* public String directory() – добавление цели в список дефолтных

обрабатываемое исключение: EJBException, IOException

* public void uploadImageGoalDef() – добавление картинки дефолтной цели

обрабатываемое исключение: EJBException

* public String deleteGoal() – удаление цели

обрабатываемые исключения: EJBException

* set и get методы для полей
* ClientBean:
* public String editClient(Client c) – просмотр личных данных

обрабатываемые исключения: EJBException

* public List<GoalUser> findAllGoalCurrentClient() – вывод целей клиента

обрабатываемые исключения: EJBException

* public String createGoalDefoltUser() – добавление дефолтной цели

обрабатываемые исключения: EJBException, NumberFormatException

* public String editGoalUser() – метод перехода на конкретную цель

обрабатываемые исключения: EJBException, NumberFormatException

* public String deleteGoalUser() – удаление цели пользователя

обрабатываемые исключения: EJBException, NumberFormatException

* public String createGoalUser() – добавление новой цели пользователя

обрабатываемое исключение: EJBException, IOException

* public void findAllGoalDefolt() – поиск по дефолтным целям

обрабатываемое исключение: EJBException

* public List<Goal> AllGoalDefolt() – вывод дефолтных целей

обрабатываемое исключение: EJBException

* public String addLevel() – выполнение цели на текущий день

обрабатываемое исключение: EJBException

* public int percentСomplianceGoalUser() – просчитывается выхоленность цели

обрабатываемое исключение: EJBException

* public ScheduleModel getEventModel() – модель для календаря
* private void createAreaModel() – метод для прорисовки графика

обрабатываемое исключение: EJBException, EnumConstantNotPresentException

* public boolean checkDate() – метод скрытия кнопки

обрабатываемое исключение: EJBException

* public boolean isDateEqual(Date dateLevel) – метод выполняющий сравнение дат
* public entity.Client calculateClientReiting(entity.Client client) – рейтинг одного пользователя
* public List<entity.Client> calculateClientsRating() – метод который считает рейтинг для всех пользователей

обрабатываемое исключение: EJBException

* private int compaeInts(int a, int b) – компоратор для сортировки по рейтингу
* public boolean payment(String holder, String codeCard, String codeSecurity, String expirationDate, String purchaseValue) throws IOException – оплата персонажа
* public StreamedContent getStreamedImageById() – вывод картинки персонажу

обрабатываемое исключение: EJBException, NumberFormatException

* public StreamedContent getStreamedImageByIdGoalDefolt() – вывод картинок дефолтным целям

обрабатываемое исключение: EJBException, NumberFormatException

* public StreamedContent getStreamedImageByIdAndLevel() – вывод картинок по уровню

обрабатываемое исключение: EJBException, NumberFormatException

* public List<Personage> AllPersonages() – получение списка персонажей

обрабатываемое исключение: EJBException

* public byte[] InputStreamToArryByte(InputStream in) throws IOException – чтение картинки из потока в байтовый массив
* public void uploadProfile() – добавление фотографии на профиль пользователя

обрабатываемое исключение: EJBException, IOException

* public StreamedContent getAvatar() – вывод фотографии пользователя

обрабатываемое исключение: EJBException

* public List<Personage> personagesFild() – получения списка персонажей
* set и get методы для полей
* UserBean:
* public String createUser() – создание нового пользователя

обрабатываемое исключение: EJBException

* public User getCurrentUserObject() – вернуть текущего пользователя

обрабатываемое исключение: EJBException

* public String getCurrentUser() – получение логина текущего пользователя

обрабатываемое исключение: EJBException, NumberFormatException

* public String logout() – выйти из системы
* public boolean isUserLoggedIn() – проверка, аутентифицирован ли пользователь

AdministratorBean:

@ManagedBean(name = "administratorBean")

@SessionScoped

public class AdministratorBean {

public AdministratorBean(){

}

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

private User user = new User();

@EJB

private UserFacadeLocal userFacade;

@EJB

ImageFacadeLocal imageFacadeLocal;

Image image = new Image();

@EJB

PersonageImageFacadeLocal personageImageFacadeLocal;

@EJB

PersonageFacadeLocal personageFacadeLocal;

Personage personage = new Personage();

@EJB

GoalFacadeLocal goalFacadeLocal;

private Goal goal = new Goal();

private PersonageImage personageImage = new PersonageImage();

private String quantityLevel;

@EJB

private ClientFacadeLocal clientFacadeLocal;

//для назначения админов

private UserRole userRole = new UserRole();

private UserRolePK rolePK = new UserRolePK();

private String USER\_ROLE = "admin";

@EJB

private UserRoleFacadeLocal userRoleFacadeLocal;

@EJB

private ClientFacadeLocal clientFacade;

//Вывод списка пользователей для администратора

public List<User> getAllUser() {

List<User> users = userFacade.findAll();

if (users == null) {

LOGGER.error("Пустой список пользователей.");

}

return users;

}

public String getQuantityLevel() {

return quantityLevel;

}

public void setQuantityLevel(String quantityLevel) {

this.quantityLevel = quantityLevel;

}

public Image getImage() {

return image;

}

public void setImage(Image image) {

this.image = image;

}

public PersonageImage getPersonageImage() {

return personageImage;

}

public void setPersonageImage(PersonageImage personageImage) {

this.personageImage = personageImage;

}

public Personage getPersonage() {

return personage;

}

public void setPersonage(Personage personage) {

this.personage = personage;

}

public Goal getGoal() {

return goal;

}

public void setGoal(Goal goal) {

this.goal = goal;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

//Добавление и вывод картинок персонажей

private UploadedFile file;

public UploadedFile getFile() {

return file;

}

public void setFile(UploadedFile file) {

this.file = file;

}

public StreamedContent getImageCreate() throws IOException {

if (file != null) {

try {

return new DefaultStreamedContent(file.getInputstream(), file.getContentType());

} catch (IOException ioe) {

LOGGER.error("Ошибка при добавлении картинок",ioe);

}

} else {

LOGGER.error("Картинка при добавлении пустая и не может быть выведена.");

}

return null;

}

//читаем картиночку из потока в байтовый массив

public byte[] InputStreamToArryByte(InputStream in) {

byte[] imageInByte = null;

try {

ByteArrayOutputStream baos = new ByteArrayOutputStream();

BufferedImage bImageFromConvert = ImageIO.read(in);

ImageIO.write(bImageFromConvert, "png", baos);

imageInByte = baos.toByteArray();

} catch (IOException ex) {

LOGGER.error("Ошибка при чтении картинки из потока в байтовый массив.", ex);

}

return imageInByte;

}

//добавление картинки персонажу на уровень

public void uploadImagePersonage(){

FacesMessage message;

if (file != null) {

try {

image.setName(file.getFileName());

image.setType(file.getContentType());

image.setPersonageImage(null);

image.setData(InputStreamToArryByte(file.getInputstream()));

imageFacadeLocal.create(image);

personage = personageFacadeLocal.findPersonageByName(personage.getName());

personageImage.setIDImage(image);

personageImage.setIDPersonage(personage);

personageImage.setLevel(Integer.parseInt(quantityLevel));

personageImageFacadeLocal.create(personageImage);

message = new FacesMessage("Добавлен файл:", file.getFileName());

} catch (EJBException | IOException ex) {

LOGGER.error("Ошибка при добавлении картинки.", ex);

message = new FacesMessage("Не добавлен файл:", file.getFileName());

}

} else {

LOGGER.error("Файл пуст и не может быть добавлен.");

message = new FacesMessage("Файл пуст и не может быть добавлен.");

}

FacesContext.getCurrentInstance().addMessage(null, message);

}

public List<Personage> personagesFild() {

return personageFacadeLocal.findAll();

}

//список дефолтных персонажей

public List<Goal> goalDef() {

return goalFacadeLocal.findGoalDefolt();

}

public List<String> levelFild() {

List<String> results = new ArrayList<String>();

for (int i = 1; i < 22; i++) {

results.add("" + i);

}

return results;

}

//Назначить бан/снять бан

public String ban() {

FacesMessage message;

try {

FacesContext fc = FacesContext.getCurrentInstance();

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

int id = Integer.parseInt(params.get("id"));

Client client = clientFacadeLocal.find(id);

client.setBan(!client.getBan());

clientFacadeLocal.edit(client);

message = new FacesMessage("Успешно", "Успешно услаовлен/снят бан");

} catch (EJBException ex) {

LOGGER.error("Ошибка при установки/снятии бана на пользователя.", ex);

message = new FacesMessage("Ошибка при установки/снятии бана на пользователя.");

}

FacesContext.getCurrentInstance().addMessage(null, message);

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

}

public String addAdmin() {

FacesMessage message;

try {

FacesContext fc = FacesContext.getCurrentInstance();

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

int id = Integer.parseInt(params.get("id"));

Client client = clientFacadeLocal.find(id);

User user = userFacade.find(client.getIDUser().getIDUser());

rolePK.setLogin(user.getLogin());

rolePK.setRole(USER\_ROLE);

userRole.setUserRolePK(rolePK);

userRole.setUser(user);

userRoleFacadeLocal.create(userRole);

clientFacadeLocal.edit(client);

message = new FacesMessage("Успешно", "Успешно добавлен администратор");

FacesContext.getCurrentInstance().addMessage(null, message);

} catch (EJBException ex) {

LOGGER.error("Ошибка при добавлении администратора.", ex);

message = new FacesMessage("Ошибка добавление администратора");

}

FacesContext.getCurrentInstance().addMessage(null, message);

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

}

public List<Goal> getAllGoal() {

return goalFacadeLocal.findAll();

}

//Установка статуса цели

public String directory() {

FacesMessage message;

try {

FacesContext fc = FacesContext.getCurrentInstance();

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

int id = Integer.parseInt(params.get("id"));

Goal goal = goalFacadeLocal.find(id);

goal.setDirectory(!goal.getDirectory());

goalFacadeLocal.edit(goal);

message = new FacesMessage("Успех ", "Статус цели изменен.");

FacesContext.getCurrentInstance().addMessage(null, message);

} catch (EJBException ex) {

LOGGER.error("Ошибка при изменении статуса цели.", ex);

message = new FacesMessage("Ошибка ", "Статус цели не изменен.");

}

FacesContext.getCurrentInstance().addMessage(null, message);

return "goalAdministration";

}

//добавление картинки дефолтной цели

public void uploadImageGoalDef() {

FacesMessage message;

if (file != null) {

try {

//добавляем картинку в БД

image.setName(file.getFileName());

image.setType(file.getContentType());

image.setPersonageImage(null);

image.setData(InputStreamToArryByte(file.getInputstream()));

imageFacadeLocal.create(image);

//изменяем цель

goal = goalFacadeLocal.find(goal.getIDGoal());

goal.setiDImage(image);

goalFacadeLocal.edit(goal);

message = new FacesMessage("Добавлен файл:", file.getFileName());

} catch (EJBException | IOException ex) {

LOGGER.error("Ошибка при добавлении картинки.", ex);

message = new FacesMessage("Не добавлен файл:", file.getFileName());

}

} else {

LOGGER.error("Файл пуст и не может быть добавлен.");

message = new FacesMessage("Файл пуст и не может быть добавлен.");

}

FacesContext.getCurrentInstance().addMessage(null, message);

}

//Удаление цели из системы

public String deleteGoal (){

FacesMessage message;

try {

FacesContext fc = FacesContext.getCurrentInstance();

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

int id = Integer.parseInt(params.get("id"));

Goal goal = goalFacadeLocal.find(id);

//логика удаления

goalFacadeLocal.remove(goal);

message = new FacesMessage("Успех ", "Цель удалена.");

FacesContext.getCurrentInstance().addMessage(null, message);

} catch (EJBException ex) {

LOGGER.error("Ошибка при удалении цели.", ex);

message = new FacesMessage("Ошибка ", "Цель не удалена.");

}

FacesContext.getCurrentInstance().addMessage(null, message);

return "goalAdministration";

}

}

ClientBean:

@Named(value = "clientBean")

@SessionScoped

public class ClientBean implements Serializable {

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

Topic topic = new Topic();

Message message = new Message();

private int idGoal;

@EJB

private GoalUserFacadeLocal goalUserFacadeLocal;

GoalUser goalUser = new GoalUser();

@EJB

private GoalFacadeLocal goalFacadeLocal;

private Goal goal = new Goal();

@EJB

private LevelFacadeLocal levelFacadeLocal;

Level level = new Level();

@EJB

private ClientFacadeLocal clientFacade;

entity.Client client = new entity.Client();

@EJB

private UserFacadeLocal userFacadeLocal;

private User user = new User();

private UserBean userBean = new UserBean();

@EJB

private PersonageFacadeLocal personageFacadeLocal;

private Personage personage = new Personage();

@EJB

ImageFacadeLocal imageFacadeLocal;

Image image = new Image();

private String search;

private List<Goal> listGoals;

// private List<Level> listLevel;

@EJB

private PersonageImageFacadeLocal personageImageFacadeLocal;

public ClientBean() {

}

public Personage getPersonage() {

return personage;

}

public void setPersonage(Personage personage) {

this.personage = personage;

}

public UserBean getUserBean() {

return userBean;

}

public Goal getGoal() {

return goal;

}

public User getUser() {

return user;

}

public void setUser(User user) {

this.user = user;

}

public void setGoal(Goal goal) {

this.goal = goal;

}

public void setUserBean(UserBean userBean) {

this.userBean = userBean;

}

public Topic getTopic() {

return topic;

}

public void setTopic(Topic topic) {

this.topic = topic;

}

public Message getMessage() {

return message;

}

public void setMessage(Message message) {

this.message = message;

}

public GoalUser getGoalUser() {

return goalUser;

}

public void setGoalUser(GoalUser goalUser) {

this.goalUser = goalUser;

}

public entity.Client getClient() {

return client;

}

public void setClient(entity.Client client) {

this.client = client;

}

public String editClient() {

try {

userFacadeLocal.edit(user);

clientFacade.edit(client);

} catch (EJBException ejbe) {

LOGGER.error("Ошибка просмотра данных клиента:", ejbe);

FacesMessage message = new FacesMessage("Ошибка", "Ошибка просмотра данных клиента.");

FacesContext.getCurrentInstance().addMessage(null, message);

}

return "profile";

}

//вывести цели клиента

public List<GoalUser> findAllGoalCurrentClient() {

List<GoalUser> goalUsers = null;

try {

user = userFacadeLocal.findLogin(userBean.getCurrentUser());

client = clientFacade.findIdUser(user.getIDUser());

goalUsers = goalUserFacadeLocal.findGoalCurrentClient(client.getIDClient());

} catch (EJBException ejbe) {

LOGGER.error("Ошибка отображения целей клиента:", ejbe);

}

return goalUsers;

}

//Добавление пользователю дефолтной цели

public String createGoalDefoltUser() {

FacesMessage message;

try {

FacesContext fc = FacesContext.getCurrentInstance();

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

int id = Integer.parseInt(params.get("id"));

goal = goalFacadeLocal.find(id);

goalUser.setIDGoal(goal);

user = userFacadeLocal.findLogin(userBean.getCurrentUser());

client = clientFacade.findIdUser(user.getIDUser());

goalUser.setIDClient(client);

goalUser.setLevelCollection(null);

this.goalUserFacadeLocal.create(this.goalUser);

//заполняем 21 день как не выполненые что-бы было подряд

Date date = new Date();

for (int i = 0; i < 21; i++) {

level.setIDGoaluser(goalUser);

level.setDate(date);

Calendar instance = Calendar.getInstance();

instance.setTime(date); //устанавливаем дату, с которой будет производить операции

instance.add(Calendar.DAY\_OF\_MONTH, 1);// прибавляем день

date = instance.getTime(); // получаем измененную дату

level.setLeveldate(false);

levelFacadeLocal.create(level);

}

message = new FacesMessage("Успех", "Дефолтная цель успешно добавлена в список ваших целей!");

} catch (NumberFormatException nfe) {

message = new FacesMessage("Ошибка", "Ошибка добавления дефолтной цели. Обратитесь к администратору.");

LOGGER.error("Ошибка добавления дефолтной цели. Пришли не верные параметры:", nfe);

} catch (EJBException ejbe) {

message = new FacesMessage("Ошибка", "Ошибка добавления дефолтной цели. Обратитесь к администратору.");

LOGGER.error("Ошибка добавления дефолтной цели:", ejbe);

}

FacesContext.getCurrentInstance().addMessage(null, message);

return "goals";

}

public String editGoalUser() {

FacesMessage message = null;

String resultStr;

try {

FacesContext fc = FacesContext.getCurrentInstance();

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

setIdGoal(Integer.parseInt(params.get("id")));

goalUser = goalUserFacadeLocal.find(idGoal);

eventModel = new DefaultScheduleModel();

//вычисление даты

List<Level> list = levelFacadeLocal.findGoalLevel(goalUserFacadeLocal.find(getIdGoal()));

for(Level t : list){

if (t.getDate().getTime() == new Date().getTime()){

break;

}

else {

if (t.getLeveldate()){

eventModel.addEvent(new DefaultScheduleEvent("Выполнено", t.getDate(), t.getDate()));

}

else

eventModel.addEvent(new DefaultScheduleEvent("НЕ выполнено", t.getDate(), t.getDate()));

}

}

lazyEventModel = new LazyScheduleModel() {

};

resultStr = "/goalUser?faces-redirect=true";

} catch (NumberFormatException nfe) {

message = new FacesMessage("Ошибка", "Ошибка перехода на страницу целей. Обратитесь к администратору.");

LOGGER.error("Ошибка перехода на страницу целей. Пришли не верные параметры:", nfe);

resultStr = "profil";

} catch (EJBException ejbe) {

message = new FacesMessage("Ошибка", "Ошибка перехода на страницу целей. Обратитесь к администратору.");

LOGGER.error("Ошибка перехода на страницу целей:", ejbe);

resultStr = "profil";

}

if (message != null) {

FacesContext.getCurrentInstance().addMessage(null, message);

}

return resultStr;

}

public String deleteGoalUser() {

FacesMessage message;

try {

FacesContext fc = FacesContext.getCurrentInstance();

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

int id = Integer.parseInt(params.get("id"));

goalUser = goalUserFacadeLocal.find(id);

goalUserFacadeLocal.remove(goalUser);

message = new FacesMessage("Успех", "Цель успешно удалена");

} catch (NumberFormatException nfe) {

message = new FacesMessage("Ошибка", "Ошибка удаления цели. Обратитесь к администратору.");

LOGGER.error("Ошибка удаления цели. Пришли не верные параметры:", nfe);

} catch (EJBException ejbe) {

message = new FacesMessage("Ошибка", "Ошибка удаления цели. Обратитесь к администратору.");

LOGGER.error("Ошибка удаления цели:", ejbe);

}

FacesContext.getCurrentInstance().addMessage(null, message);

return "profil";

}

public String createGoalUser() {

FacesMessage message = null;

try {

personage = personageFacadeLocal.find(personage.getIDPersonage());

boolean paymentFlag = false;

//производим оплату персонажа

if (personage.getPrice() > 0) {

String purchaseValue = String.valueOf(personage.getPrice());

try {

if (!holder.isEmpty() && !codeCard.isEmpty() && !codeSecurity.isEmpty() && !expirationDate.isEmpty()) {//ну хоть что-то проверим

paymentFlag = payment(holder, codeCard, codeSecurity, expirationDate, purchaseValue);

if (paymentFlag) {

message = new FacesMessage("Успех", "Прошла оплата персонажа.");

} else {

message = new FacesMessage("Ошибка", "Оплата не прошла. Проверьте корректность и аклуальность введенных данных.");

}

} else {

message = new FacesMessage("Ошибка", "Данные оплаты не заполнены.");

}

} catch (IOException ex) {

LOGGER.error("Оплата персонажа не прошла. Ошибка подключения к сервису.", ex);

message = new FacesMessage("Ошибка", "Оплата персонажа не прошла. Ошибка с серсвисом.");

}

} else {

paymentFlag = true;

}

if (paymentFlag) {

user = userFacadeLocal.findLogin(userBean.getCurrentUser());

client = clientFacade.findIdUser(user.getIDUser());

goal.setDirectory(true);

goal.setGoalUserCollection(null);

goal.setIDPersonage(personage);

goalFacadeLocal.create(goal);

goalUser.setIDGoal(goal);

goalUser.setIDClient(client);

goalUser.setLevelCollection(null);

this.goalUserFacadeLocal.create(this.goalUser);

//заполняем 21 день как не выполненые что-бы было подряд

Date date = new Date();

for (int i = 0; i < 21; i++) {

level.setIDGoaluser(goalUser);

level.setDate(date);

Calendar instance = Calendar.getInstance();

instance.setTime(date); //устанавливаем дату, с которой будет производить операции

instance.add(Calendar.DAY\_OF\_MONTH, 1);// прибавляем день

date = instance.getTime(); // получаем измененную дату

level.setLeveldate(false);

levelFacadeLocal.create(level);

}

message = new FacesMessage("Успех","Цель успешно добавлена.");

}

} catch (EJBException ejbe) {

LOGGER.error("Персонаж не добавлен. Ошибка: ", ejbe);

message = new FacesMessage("Ошибка","Персонаж не добавлен. Обратитесь к администратору.");

}

if (message != null) FacesContext.getCurrentInstance().addMessage(null, message);

return "goals";

}

//поиск по дефолтным целям

public void findAllGoalDefolt() {

try {

if (search.trim().length() == 0){//возможно просто длину достаточно проверить

setListGoals(goalFacadeLocal.findGoalDefolt());

} else {

setListGoals(goalFacadeLocal.findGoalSearch(search.trim()));

}

}catch (EJBException ejbe){

LOGGER.error("Ошибка при поиске дефолтных целей: ", ejbe);

}

}

//вывод дефолтных целей

public List<Goal> AllGoalDefolt() {

try {

if (listGoals == null)

setListGoals(goalFacadeLocal.findGoalDefolt());

}catch (EJBException ejbe){

LOGGER.error("Ошибка при выводе дефолтных целей: ", ejbe);

}

return listGoals;

}

public void setListGoals(List<Goal> listGoals) {

this.listGoals = listGoals;

}

//Выполнение цели на день

public String addLevel() {

FacesMessage message = null;

try {

Collection<Level> levels = goalUserFacadeLocal.find(goalUser.getIDGoaluser()).getLevelCollection();

for (Level item : levels) {

if (isDateEqual(item.getDate()) && !item.getLeveldate()) {

item.setLeveldate(true);

levelFacadeLocal.edit(item);

message = new FacesMessage("Успех","Цель успешно выполнена на сегодня.");

}

}

} catch (EJBException ejbe){

LOGGER.error("Ошибка при выполнении цели на день: ", ejbe);

message = new FacesMessage("Ошибка","Ошибка при выполнении цел на день. Обратитесь к администратору.");

}

if (message != null) FacesContext.getCurrentInstance().addMessage(null, message);

return "goalUser";

}

//выполненость

public int percentСomplianceGoalUser() {

int caunt = 0;

try {

Collection<Level> levels = goalUserFacadeLocal.find(goalUser.getIDGoaluser()).getLevelCollection();

for (Level item : levels) {

if (item.getLeveldate()) {

caunt++;

}

}

} catch (EJBException ejbe){

LOGGER.error("Не удалось подсчитать выполненность цели: ", ejbe);

}

int percent = (100 \* caunt) / 21;

return percent;

}

//тут часть кода отвечвющая за календарик

//тут надо разобраться пока не причесывала

private ScheduleModel eventModel;

private ScheduleModel lazyEventModel;

private ScheduleEvent event = new DefaultScheduleEvent();

public Date getInitialDate() {

Calendar calendar = Calendar.getInstance();

calendar.set(calendar.get(Calendar.YEAR), Calendar.FEBRUARY, calendar.get(Calendar.DATE), 0, 0, 0);

return calendar.getTime();

}

public ScheduleModel getEventModel() {

return eventModel;

}

public ScheduleEvent getEvent() {

return event;

}

public void setEvent(ScheduleEvent event) {

this.event = event;

}

public void onEventSelect(SelectEvent selectEvent) {

event = (ScheduleEvent) selectEvent.getObject();

}

public void onDateSelect(SelectEvent selectEvent) {

event = new DefaultScheduleEvent("", (Date) selectEvent.getObject(), (Date) selectEvent.getObject());

}

public String getSearch() {

return search;

}

public void setSearch(String search) {

this.search = search;

}

public List<Goal> getListGoals() {

return listGoals;

}

private LineChartModel areaModel;

public LineChartModel getAreaModel() {

createAreaModel();

return areaModel;

}

//график

private void createAreaModel() {

try {

areaModel = new LineChartModel();

LineChartSeries target = new LineChartSeries();

target.setFill(true);

goalUser = goalUserFacadeLocal.find(goalUser.getIDGoaluser());

target.setLabel(goalUser.getIDGoal().getName());

int namberLevel = 0;

for (Level item : goalUser.getLevelCollection()) {

if (item.getLeveldate()) {

namberLevel++;

}

target.set(item.getDate().toString(), namberLevel);

if (isDateEqual(item.getDate())) {

break;

}

}

areaModel.addSeries(target);

areaModel.setTitle("Достижение цели");

areaModel.setLegendPosition("ne");

areaModel.setStacked(true);

areaModel.setShowPointLabels(true);

Axis xAxis = new CategoryAxis("Дни");

areaModel.getAxes().put(AxisType.X, xAxis);

Axis yAxis = areaModel.getAxis(AxisType.Y);

yAxis.setLabel("Шкала выполнения");

yAxis.setMin(1);

yAxis.setMax(goalUser.getLevelCollection().size());

} catch (EnumConstantNotPresentException ecnpe) {

LOGGER.error("Попытка использования неопределённого значения перечисления: ", ecnpe);

} catch (EJBException ejbe){

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

}

}

//метод скрытия кнопки выполнения цели

public boolean checkDate() {

try {

Collection<Level> levels = goalUserFacadeLocal.find(goalUser.getIDGoaluser()).getLevelCollection();

if (!levels.isEmpty()) {

for (Level level : levels) {

if (isDateEqual(level.getDate()) && level.getLeveldate())

return true;

}

} else {

return false;

}

}catch(EJBException ejbe){

LOGGER.error("Ошибка при получении списка выполнения целей:", ejbe);

}

return false;

}

//сравнение дат

public boolean isDateEqual(Date dateLevel) {

LocalDate current = LocalDate.now();

LocalDate levelLocalDate = Instant.ofEpochMilli(dateLevel.getTime()).atZone(ZoneId.systemDefault()).toLocalDate();

return levelLocalDate.isEqual(current);

}

//рейтинг одного пользователя

public entity.Client calculateClientReiting(entity.Client client) {

int rating = 0;

if (client.getGoalUserCollection() != null) {

for (GoalUser gu : client.getGoalUserCollection()) {

if (gu.getLevelCollection().size() - 1 == 21)

rating++;

}

}

client.setRating(rating);

return client;

}

//метод который считает рейтинг для всех

public List<entity.Client> calculateClientsRating() {

List<entity.Client> ratingList= null;

try {

List<entity.Client> clients = clientFacade.findAll();

ratingList = new ArrayList<>();

for (entity.Client c : clients)

ratingList.add(calculateClientReiting(c));

Collections.sort(ratingList, (c1, c2) -> compaeInts(c1.getRating(), c2.getRating()));

} catch (EJBException ejbe) {

LOGGER.error("Ошибка при получении списка пользователей:", ejbe);

}

return ratingList;

}

private int compaeInts(int a, int b) {

if (a < b) {

return -1;

} else if (a == b) {

return 0;

} else {

return 1;

}

}

//не уверена что это правильно

private String expirationDate;

private String holder;

private String codeSecurity;

private String codeCard;

public String getExpirationDate() {

return expirationDate;

}

public void setExpirationDate(String expirationDate) {

this.expirationDate = expirationDate;

}

public String getHolder() {

return holder;

}

public void setHolder(String holder) {

this.holder = holder;

}

public String getCodeSecurity() {

return codeSecurity;

}

public void setCodeSecurity(String codeSecurity) {

this.codeSecurity = codeSecurity;

}

public String getCodeCard() {

return codeCard;

}

public void setCodeCard(String codeCard) {

this.codeCard = codeCard;

}

//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;

}

//вывод картинки персонажу

public StreamedContent getStreamedImageById() {

try {

FacesContext context = FacesContext.getCurrentInstance();

if (context.getCurrentPhaseId() == PhaseId.RENDER\_RESPONSE) {

return new DefaultStreamedContent();

} else {

context = FacesContext.getCurrentInstance();

Map<String, String> params = context.getExternalContext().getRequestParameterMap();

int id = Integer.parseInt(params.get("id"));

Personage pers = personageFacadeLocal.find(id);

List<PersonageImage> personageImages = personageImageFacadeLocal.findAll();

Image img = null;

for (PersonageImage personageImage : personageImages) {

if ((personageImage.getIDPersonage().getIDPersonage() == pers.getIDPersonage()) && personageImage.getLevel() == 8) {

img = imageFacadeLocal.find(personageImage.getIDImage().getIDImage());

break;

}

}

if (img != null) {

return new DefaultStreamedContent(new ByteArrayInputStream(img.getData()), img.getType());

}

}

} catch (EJBException ejbe) {

LOGGER.error("Ошибка вывода картинок:", ejbe);

} catch (NumberFormatException nfe){

LOGGER.error("Ошибка вывода картинок. Пришли не верные параметры:", nfe);

}

return null;

}

//Вывод картинок дефолтным целям

public StreamedContent getStreamedImageByIdGoalDefolt() {

try {

FacesContext context = FacesContext.getCurrentInstance();

if (context.getCurrentPhaseId() == PhaseId.RENDER\_RESPONSE) {

return new DefaultStreamedContent();

} else {

context = FacesContext.getCurrentInstance();

Map<String, String> params = context.getExternalContext().getRequestParameterMap();

int id = Integer.parseInt(params.get("id"));

Goal goal = goalFacadeLocal.find(id);

Image img = imageFacadeLocal.find(goal.getiDImage().getIDImage());

if (img != null) {

return new DefaultStreamedContent(new ByteArrayInputStream(img.getData()), img.getType());

}

}

} catch (EJBException ejbe) {

LOGGER.error("Ошибка вывода картинки к дефолтной цели:", ejbe);

}catch (NumberFormatException nfe){

LOGGER.error("Ошибка вывода картинки к дефолтной цели. Пришли не верные параметры:", nfe);

}

return null;

}

//Вывод картинки по уровню

public StreamedContent getStreamedImageByIdAndLevel() {

try {

Personage pers = personageFacadeLocal.find(goalUser.getIDGoal().getIDPersonage().getIDPersonage());

List<PersonageImage> personageImages = personageImageFacadeLocal.findAll();

client = clientFacade.find(goalUser.getIDClient().getIDClient());

Image img = null;

int level = 1;

Collection<Level> levels = goalUserFacadeLocal.find(goalUser.getIDGoaluser()).getLevelCollection();

if (!levels.isEmpty()) {

for (Level item : levels) {

if (item.getLeveldate()) {

level++;

}

}

}

for (PersonageImage personageImage : personageImages) {

if (personageImage.getIDPersonage().getIDPersonage() == pers.getIDPersonage() && personageImage.getLevel() == level) {

img = imageFacadeLocal.find(personageImage.getIDImage().getIDImage());

break;

}

}

if (img != null)

return new DefaultStreamedContent(new ByteArrayInputStream(img.getData()), img.getType());

} catch (EJBException ejbe) {

LOGGER.error("Ошибка вывода картиноки по уровню:", ejbe);

}catch (NumberFormatException nfe){

LOGGER.error("Ошибка вывода картинки по уровню. Пришли не верные параметры:", nfe);

}

return null;

}

public List<Personage> AllPersonages() {

List<Personage> personages = null;

try {

personages = personageFacadeLocal.findAll();

} catch (EJBException ejbe) {

LOGGER.error("Ошибка при получении списка персонажей из БД:", ejbe);

}

return personages;

}

//добавление авотарки пользователю

private UploadedFile file;

public UploadedFile getFile() {

return file;

}

public void setFile(UploadedFile file) throws IOException {

this.file = file;

}

//читаем картиночку из потока в байтовый массив

public byte[] InputStreamToArryByte(InputStream in) throws IOException {

byte[] imageInByte;

try (ByteArrayOutputStream baos = new ByteArrayOutputStream()) {

BufferedImage bImageFromConvert = ImageIO.read(in);

ImageIO.write(bImageFromConvert, "jpg", baos);

imageInByte = baos.toByteArray();

}

return imageInByte;

}

//добавление авторарки

public void uploadProfile() {

try {

if (file != null) {

image.setName(file.getFileName());

image.setType(file.getContentType());

image.setPersonageImage(null);

image.setData(InputStreamToArryByte(file.getInputstream()));

imageFacadeLocal.create(image);

user = userFacadeLocal.findLogin(userBean.getCurrentUser());

client = clientFacade.findIdUser(user.getIDUser());

client.setiDImage(image);

clientFacade.edit(client);

FacesMessage message = new FacesMessage("Добавлен файл:", file.getFileName());

FacesContext.getCurrentInstance().addMessage(null, message);

}

} catch (IOException ioe) {

LOGGER.error("Добавленни аватаки пользователю не произошло. Ошибка в чтении из потока в байтовый массив:", ioe);

}catch (EJBException ejbe){

LOGGER.error("Ошибка при добавленни авотаки пользователю:", ejbe);

}

}

public StreamedContent getAvatar() {

Image img = null;

try {

user = userFacadeLocal.findLogin(userBean.getCurrentUser());

client = clientFacade.findIdUser(user.getIDUser());

img = client.getiDImage();

} catch (EJBException ejbe) {

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

}

if (img == null) return null;

return new DefaultStreamedContent(new ByteArrayInputStream(img.getData()), img.getType());

}

public List<Personage> personagesFild() {

List<Personage> personages = null;

try {

personages = personageFacadeLocal.findAll();

} catch (EJBException ejbe) {

LOGGER.error("Ошибка при получении списка персонажей:", ejbe);

}

return personages;

}

/\*\*

\* @return the idGoal

\*/

public int getIdGoal() {

return idGoal;

}

/\*\*

\* @param idGoal the idGoal to set

\*/

public void setIdGoal(int idGoal) {

this.idGoal = idGoal;

}

}

ForumBean:

@Named(value = "forumBean")

@SessionScoped//может стоит на запрос?

public class ForumBean implements Serializable {

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

public ForumBean() {

}

@EJB

private ClientFacadeLocal clientFacade;

private Client client = new Client();

private List<Message> messageTopic;

@EJB

private UserFacadeLocal userFacade;

@EJB

private MessageFacadeLocal messageFacade;

private Message message = new Message();

@EJB

private TopicFacadeLocal topicFacade;

private Topic topic = new Topic();

private User user = new User();

private UserBean userBean = new UserBean();

private int id;

@EJB

private ClientFacadeLocal clientFacadeLocal;

public Topic getTopic() {

return topic;

}

public void setTopic(Topic topic) {

this.topic = topic;

}

public Message getMessage() {

return message;

}

public void setMessage(Message message) {

this.message = message;

}

public void setMessageTopic(List<Message> messageTopic) {

this.messageTopic = messageTopic;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public List<Message> getMessageTopic() {

List<Message> messages = null;

try {

messages = messageTopic = messageFacade.findByIdTopic(getId());

} catch (EJBException ejbe) {

LOGGER.error("Ошибка при получении списка сообщений по теме:", ejbe);

}

return messages;

}

public List<Topic> getAllTopics() {

List<Topic> topics = null;

try {

topics = topicFacade.findAll();

} catch (EJBException ejbe) {

LOGGER.error("Ошибка при получении списка тем:", ejbe);

}

return topics;

}

public List<Message> getAllMessage() {

List<Message> messages = null;

try {

messages = messageFacade.findAll();

} catch (EJBException ejbe) {

LOGGER.error("Ошибка при получении списка сообщений:", ejbe);

}

return messages;

}

// Вывести сообщения по id темы

public String messageIdTopic() {

FacesMessage message = null;

try {

FacesContext fc = FacesContext.getCurrentInstance();

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

id = Integer.parseInt(params.get("id"));

messageTopic = messageFacade.findByIdTopic(getId());

} catch (EJBException ejbe) {

LOGGER.error("Ошибка при получении списка сообщений:", ejbe);

message = new FacesMessage("Ошибка", "Ошибка при выводе сообщений.");

} catch (NumberFormatException nfe){

LOGGER.error("Ошибка при выводе сообщений. Пришли не верные параметры:", nfe);

message = new FacesMessage("Ошибка", "Ошибка при выводе сообщений.");

}

if (message != null) FacesContext.getCurrentInstance().addMessage(null, message);

return "comment";

}

public String deleteMessage(Message message) {

FacesMessage facesMessage;

try {

messageFacade.remove(message);

facesMessage = new FacesMessage("Успех", "Сообщение успешно удалено.");

} catch (EJBException ejbe) {

LOGGER.error("Ошибка при удалении сообщения:", ejbe);

facesMessage = new FacesMessage("Ошибка", "Ошибка при удалении сообщения.");

}

FacesContext.getCurrentInstance().addMessage(null, facesMessage);

return "comment";

}

public String createTopic() {

FacesMessage facesMessage;

try {

topic.setDate(new Date());

user = userFacade.findLogin(userBean.getCurrentUser());

topic.setIDUser(user);

this.topicFacade.create(this.getTopic());

facesMessage = new FacesMessage("Ошибка", "Тема успешно создана.");

}catch (EJBException ejbe){

LOGGER.error("Ошибка создании темы:", ejbe);

facesMessage = new FacesMessage("Ошибка", "Ошибка создании темы.");

}

FacesContext.getCurrentInstance().addMessage(null, facesMessage);

topic.setName("");

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

}

public String createMessage() {

try {

user = userFacade.findLogin(userBean.getCurrentUser());

client = clientFacade.findIdUser(user.getIDUser());

if (!client.getBan()) {

message.setDate(new Date());

message.setIDTopic(topicFacade.find(getId()));

message.setIDUser(user);

this.messageFacade.create(this.message);

message.setContent("");

}

} catch (EJBException ejbe) {

LOGGER.error("Ошибка при отправке сообщения:", ejbe);

FacesMessage facesMessage = new FacesMessage("Ошибка", "Ошибка при отправке сообщения.");

FacesContext.getCurrentInstance().addMessage(null, facesMessage);

}

return "comment";

}

//Назначить бан/снять бан

public String ban() {

try {

FacesContext fc = FacesContext.getCurrentInstance();

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

int id = Integer.parseInt(params.get("id"));

Client client = clientFacadeLocal.find(id);

client.setBan(!client.getBan());

clientFacadeLocal.edit(client);

} catch (EJBException ex) {

LOGGER.error("Ошибка при установки/снятии бана на пользователя.", ex);

FacesMessage message = new FacesMessage("Ошибка при установки/снятии бана на пользователя.");

FacesContext.getCurrentInstance().addMessage(null, message);

}

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

}

}

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!!!"));// не уверена...ооочень не уверена

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

}

return "authorization";

}

//вырнуть текущего

public User getCurrentUserObject(){

try {

user = userFacade.findLogin(getCurrentUser());

} catch (EJBException ejbe) {

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

}

return user;

}

//наименование такное-себе но менять не помню где

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;

}

}

ПРИЛОЖЕНИЕ Б  
Скрипт создания базы данных

CREATE TABLE User (

ID\_User int AUTO\_INCREMENT NOT NULL,

Login varchar(10) NOT NULL UNIQUE,

Pass varchar(10) NOT NULL,

Surname varchar(20) NOT NULL,

Name varchar(20) NOT NULL,

Phone varchar(10) NOT NULL,

Email varchar(30) NULL,

Gender varchar(10) NULL,

#UNIQUE KEY `Login\_UNIQUE` (`Login`),

PRIMARY KEY (ID\_User)

);

CREATE TABLE User\_Role (

Role varchar(10) NOT NULL,

Login varchar(10) NOT NULL,

PRIMARY KEY (Role,Login),

CONSTRAINT `FK\_myuser\_role\_1` FOREIGN KEY (Login) REFERENCES User (Login)

);

CREATE TABLE Message (

ID\_Message Int AUTO\_INCREMENT NOT NULL,

Date Date NOT NULL,

Content Text NULL,

ID\_User Int NOT NULL,

ID\_Topic Int NOT NULL,

PRIMARY KEY (ID\_Message)

);

CREATE TABLE Topic (

ID\_Topic Int AUTO\_INCREMENT NOT NULL,

Name varchar(30) NOT NULL,

Date Date NOT NULL,

ID\_User Int NOT NULL,

PRIMARY KEY (ID\_Topic)

);

CREATE TABLE Client (

ID\_Client Int AUTO\_INCREMENT NOT NULL,

Date\_Birth Date NULL,

Interests Text NULL,

ID\_User Int NOT NULL,

ID\_Image Int UNSIGNED NULL,

PRIMARY KEY (ID\_Client),

ban Bit NOT NULL,

UNIQUE (ID\_Image),

UNIQUE (ID\_User)

);

CREATE TABLE Goal\_user (

ID\_Goal\_user Int AUTO\_INCREMENT NOT NULL,

ID\_Goal Int NOT NULL,

ID\_Client Int NOT NULL,

PRIMARY KEY (ID\_Goal\_user)

);

CREATE TABLE Goal (

ID\_Goal Int AUTO\_INCREMENT NOT NULL,

Name varchar(30) NOT NULL,

Directory Bit NOT NULL,

Description text not null,

ID\_Personage Int NOT NULL,

ID\_Image Int UNSIGNED NULL,

PRIMARY KEY (ID\_Goal),

UNIQUE (ID\_Image)

);

CREATE TABLE Personage (

ID\_Personage Int AUTO\_INCREMENT NOT NULL,

Name varchar(30) NOT NULL,

Price Float NOT NULL,

PRIMARY KEY (ID\_Personage)

);

CREATE TABLE Level (

ID\_Level Int AUTO\_INCREMENT NOT NULL,

ID\_Goal\_user Int NOT NULL,

Date Date NOT NULL,

Level\_date Bit NULL,#тут правка

PRIMARY KEY (ID\_Level)

);

CREATE TABLE Image(

ID\_Image Int UNSIGNED NOT NULL AUTO\_INCREMENT,

name VARCHAR(30) NOT NULL,

type VARCHAR(20) NOT NULL,

data MEDIUMBLOB NOT NULL,

PRIMARY KEY (ID\_Image),

UNIQUE (name)

);

CREATE TABLE Personage\_Image (

ID\_Personage\_Image Int AUTO\_INCREMENT NOT NULL,

ID\_Image Int UNSIGNED NULL NULL,#Картиночка на уровне

Level Int NULL NULL,#на каком уровне выводить данную картинку

ID\_Personage Int NULL NULL,#Какой-то песонаж

PRIMARY KEY (ID\_Personage\_Image),

UNIQUE (ID\_Image)

);

#//связи между таблицами

ALTER TABLE Goal ADD FOREIGN KEY (ID\_Personage) REFERENCES Personage (ID\_Personage);

ALTER TABLE Level ADD FOREIGN KEY (ID\_Goal\_user) REFERENCES Goal\_user (ID\_Goal\_user);

ALTER TABLE Goal\_user ADD FOREIGN KEY (ID\_Goal) REFERENCES Goal (ID\_Goal);

ALTER TABLE Goal\_user ADD FOREIGN KEY (ID\_Client) REFERENCES Client (ID\_Client);

ALTER TABLE Client ADD FOREIGN KEY (ID\_User) REFERENCES User (ID\_User);

ALTER TABLE Message ADD FOREIGN KEY (ID\_User) REFERENCES User (ID\_User);

ALTER TABLE Message ADD FOREIGN KEY (ID\_Topic) REFERENCES Topic (ID\_Topic);

ALTER TABLE Topic ADD FOREIGN KEY (ID\_User) REFERENCES User (ID\_User);

#Ключи

ALTER TABLE Personage\_Image ADD FOREIGN KEY (ID\_Image) REFERENCES Image (ID\_Image);

ALTER TABLE Personage\_Image ADD FOREIGN KEY (ID\_Personage) REFERENCES Personage (ID\_Personage);

ПРИЛОЖЕНИЕ B  
Скрипт paying-service

Пакет model:

@Entity

public class Account {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long codeCard;

@Temporal(javax.persistence.TemporalType.DATE)

private Date expirationDate;

private int codeSecurity;

private float balance;

private String holder;

public Account() {

}

public Account(Date expirationDate, int codeSecurity, String holder,Long codeCard) {

this.expirationDate = expirationDate;

this.codeSecurity = codeSecurity;

this.holder = holder;

this.codeCard = codeCard;

}

public Long getCodeCard() {

return codeCard;

}

public void setCodeCard(Long codeCard) {

this.codeCard = codeCard;

}

public Date getExpirationDate() {

return expirationDate;

}

public void setExpirationDate(Date expirationDate) {

this.expirationDate = expirationDate;

}

public int getCodeSecurity() {

return codeSecurity;

}

public void setCodeSecurity(int codeSecurity) {

this.codeSecurity = codeSecurity;

}

public float getBalance() {

return balance;

}

public void setBalance(float balance) {

this.balance = balance;

}

public String getHolder() {

return holder;

}

public void setHolder(String holder) {

this.holder = holder;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Account)) return false;

Account account = (Account) o;

return codeSecurity == account.codeSecurity &&

Objects.equals(codeCard, account.codeCard) &&

Objects.equals(expirationDate, account.expirationDate) &&

Objects.equals(holder, account.holder);

}

}

Пакет controller:

@RestController

@RequestMapping("/account")

public class AccountController {

public AccountController() {

}

@Autowired

private AccountDataAccess accountDataAccess;

@RequestMapping(value = "/purchase", method = RequestMethod.POST)

public boolean purchaseTest(@RequestParam("expirationDate") String expirationDate,@RequestParam("holder") String holder,@RequestParam("codeSecurity") int codeSecurity,@RequestParam("codeCard") Long codeCard, @RequestParam("purchaseValue") float purchaseValue) throws ParseException {

SimpleDateFormat format = new SimpleDateFormat();

format.applyPattern("yyyy-MM-dd");

Date docDate= format.parse(expirationDate);

Account account = new Account(docDate,codeSecurity,holder,codeCard);

boolean result = false;

Account systemAccount = accountDataAccess.getById(account.getCodeCard());

if (systemAccount.equals(account) && systemAccount.getBalance() > purchaseValue) {

systemAccount.setBalance(systemAccount.getBalance() - purchaseValue);

accountDataAccess.update(systemAccount);

result = true;

}

return result;

}

}