МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направление подготовки / специальности)

Разработка программного обеспечения
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка веб-приложения для автоматизации процессов взаимодействия с клиентами в компании ООО «Квартплата»»

Обучающийся Н.С. Зайцев
(Инициалы Фамилия) (Личная подпись)

Руководитель кандидат пед. наук, доцент, Е.А. Ерофеева
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант кандидат фил. наук, доцент, М.В. Дайнеко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Бакалаврская работа посвящена разработке веб-приложения для автоматизации сбора и обработки пользовательских отзывов в компании ООО «Квартплата 24». Актуальность исследования обусловлена необходимостью совершенствования каналов обратной связи с клиентами в сфере жилищно-коммунальных услуг.

Целью работы является создание веб-интерфейса, позволяющего систематизировать процесс получения и анализа пользовательских отзывов. В ходе исследования решен комплекс задач, включающий анализ предметной области, проектирование архитектуры системы, выбор технологического стека и реализацию функционального прототипа.

Практическая значимость работы заключается в создании масштабируемого решения, интегрируемого в существующую ИТ-инфраструктуру компании. Разработанное приложение позволяет оптимизировать процессы обработки пользовательской обратной связи и повысить качество клиентского сервиса.

Структурно работа состоит из введения, двух глав, заключения, списка использованных источников и двух приложений.

В первой главе проведен анализ предметной области, сформированы функциональные требования и разработано техническое задание.

Вторая глава посвящена логическому проектированию системы, реализации и тестированию программного решения.

Объем работы составляет 52 страницы, включая 57 иллюстраций и 2 таблицы.

Abstract

The bachelor's thesis is devoted to the development of a web application for automating the collection and processing of user reviews at «Kvartplata24» LLC. The relevance of the study is due to the need to improve customer feedback channels in the field of housing and communal services.

The purpose of the work is to create a web interface that allows you to systematize the process of receiving and analyzing user reviews. In the course of the research were solved some tasks, including domain analysis, system architecture design, technology stack selection, and implementation of a functional prototype.

The practical significance of the work lies in creating a scalable solution that company can integrate into its existing IT infrastructure. The developed application allows you to optimize the processing of user feedback and improve the quality of customer service.

Structurally, the work consists of an introduction, two chapters, a conclusion, a list of sources used, and two appendices.

The first chapter describes an analysis of the subject’s area, functional requirements and a technical task.

The second chapter is devoted to the logical design of the system, the implementation and testing of the software solution.

The volume of the work is 52 pages, including 57 illustrations and 2 tables.

Оглавление

Введение

В условиях активной цифровизации экономики и социальной сферы особое значение приобретают технологии, обеспечивающие эффективное взаимодействие между поставщиками услуг и их потребителями. Особенно остро этот вопрос стоит в сфере жилищно-коммунального хозяйства, где качество сервиса напрямую влияет на уровень жизни населения. При этом современные ИТ-решения позволяют не только автоматизировать процессы расчета и начисления платежей, но и создавать удобные каналы коммуникации с конечными пользователями.

Актуальность исследования обусловлена возрастающими требованиями потребителей к качеству и прозрачности коммунальных услуг, необходимостью оперативного получения и обработки обратной связи от пользователей и потребностью в специализированных решениях, интегрируемых с существующими системами учета ЖКХ.

Разработка эффективного инструмента сбора и анализа пользовательских отзывов становится важной задачей для компаний, работающих в этом секторе. Особую значимость такое решение приобретает для организаций, предоставляющих программное обеспечение для автоматизации ЖКХ, где обратная связь от клиентов позволяет совершенствовать функционал и повышать качество сервиса.

Объект исследования - ИТ-компания «Квартплата 24», специализирующаяся на разработке программных решений для автоматизации расчетов в сфере жилищно-коммунального хозяйства.

Предмет исследования - методы и технологии создания веб-интерфейсов для сбора и обработки пользовательских отзывов в системах автоматизации ЖКХ.

Цель работы - разработка и внедрение веб-интерфейса для сбора и систематизации отзывов пользователей программного обеспечения компании «Квартплата 24».

Для достижения поставленной цели необходимо решить следующие задачи:

– проанализировать существующие решения в области сбора пользовательской обратной связи;
– определить функциональные требования к системе;
– разработать архитектуру веб-интерфейса;
– выбрать оптимальный технологический стек для реализации;
– провести тестирование разработанного решения.

Практическая значимость работы заключается в том, что внедрение разработанного интерфейса позволит:

– улучшить качество обратной связи с пользователями;
– сократить время обработки отзывов;
– повысить удовлетворенность клиентов сервисом компании.

Работа имеет четкую практическую направленность и соответствует современным тенденциям цифровизации сферы ЖКХ. Разработанное решение может быть интегрировано в существующую ИТ-инфраструктуру компании без существенных изменений в текущих бизнес-процессах.

Глава 1 Постановка задачи на разработку программного обеспечения для информационной системы предприятия

1.1 Анализ предметной области

Предметная область компании — Жилищно-коммунальное хозяйство. Миссия состоит в том, чтобы самая непрозрачная и непонятная сфера деятельности стала прозрачной и понятной, как для организаций — участников процесса, так и для жителей. Задачей организации является разработана и поддержка экосистемы облачных сервисов, которая помогает автоматизировать расчет и учет оплаты за жилищно-коммунальные услуги, что обеспечивает удобство всем клиентам, обеспечивает соответствие расчета законодательству, а прием платежей делает практически мгновенным и прозрачным.

Клиенты «Квартплаты24», которых более 700 — ТСЖ, ЖСК, управляющие и ресурсоснабжающие организации, крупные ЕИРЦ, расположенные по всей территории РФ. Система сама производит сложные расчеты, формирует необходимую отчетность, коллектив оказывает поддержку системы и её совершенствование. Регламентированная передача информации (например, в ГИС ЖКХ или ОСЗН) перестает быть проблемой, а наличие сервиса личного кабинета жителя делает взаимодействие с системой проще.

Организационная схема представлена на рисунке 1.

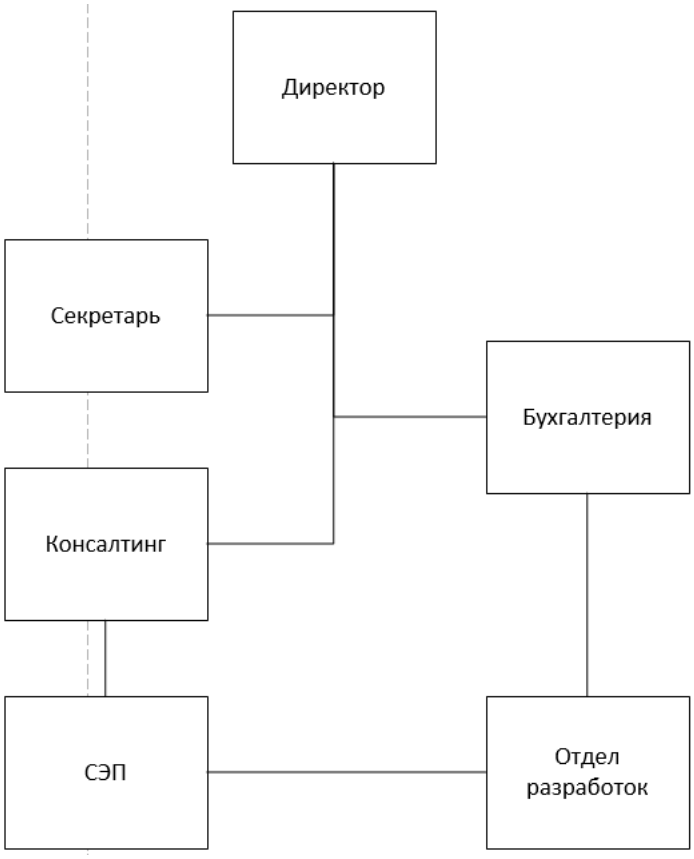


Рисунок 1 - Организационная схема компании

Схема отражает основные компоненты организации:

– директор участвует в разработке сервисов, определяя основные приоритеты компании, хотя не является членом отдела разработок;

– консалтинг переговаривается с клиентами;

– бухгалтерия поддерживает финансовую деятельность организации;

– секретарь принимает клиентов и партнёров в офисе;

– служба экспертной поддержки оказывает помощь клиентам по горячей линии;

– отдел разработок занимается развитием и поддержкой облачных сервисов;

Схема на рисунке 2 показывает бизнес-процесс «Квартплаты24»:

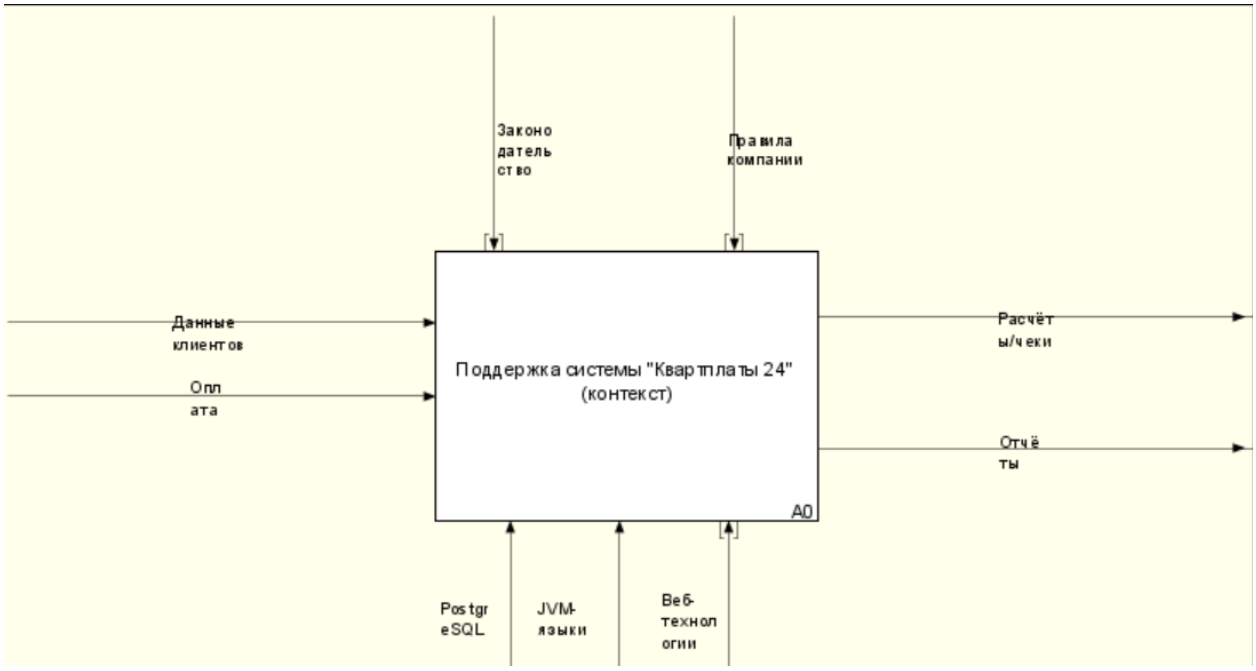


Рисунок 2 - Схема процесса системы «Квартплаты24»

Процесс направлен на обновление и доработку системы облачных вычислений, которая должна сохранять удовлетворение клиентов и партнёров компании. Это обеспечивается различными информационными технологиями, такими как базы данных, облачные вычисления и корпоративные системы [1].

Общий процесс был декомпозирован для дальнейшей детализации (рисунок 3):

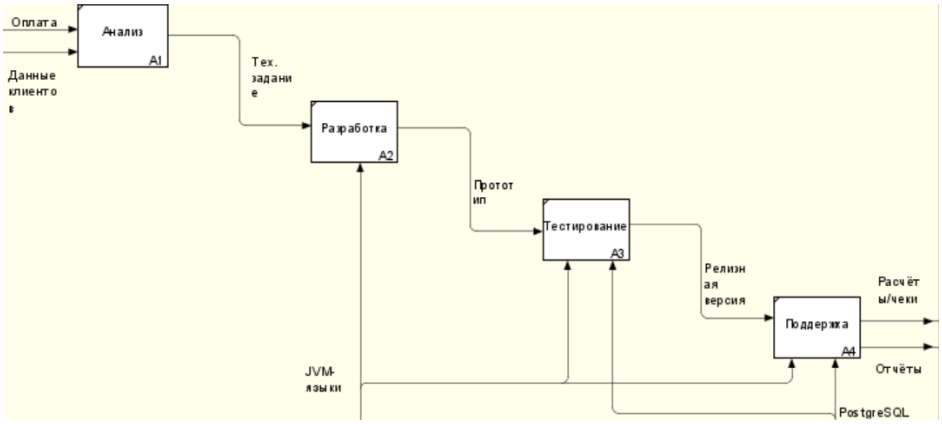


Рисунок 3 - Декомпозиция рабочего процесса организации

На рисунке представлена декомпозиция рабочего процесса организации, включающая в себя определённые этапы, начиная от анализа требований и заканчивая поддержкой и выпуском прототипов программного обеспечения. Законодательство сопровождает все этапы, процесс которых требует определённых рамок. После релиза программы публикуются и начинают выполнять свою бизнес-логику, которая обеспечивает выполнение поставленных задач. На выходе получается результат разработки\обновления в виде учётных данных, транзакций и прочих отчётов, которые будут опубликованы в личном кабинете клиента или другом месте при необходимости.

1.2 Анализ требований

Практика подразумевала разработку веб-приложения, которое будет автоматизировать работу с клиентами – было решено сделать страницу с возможностью просмотра и отправки отзывов. Диаграмма Use Case изображена на рисунке 4.

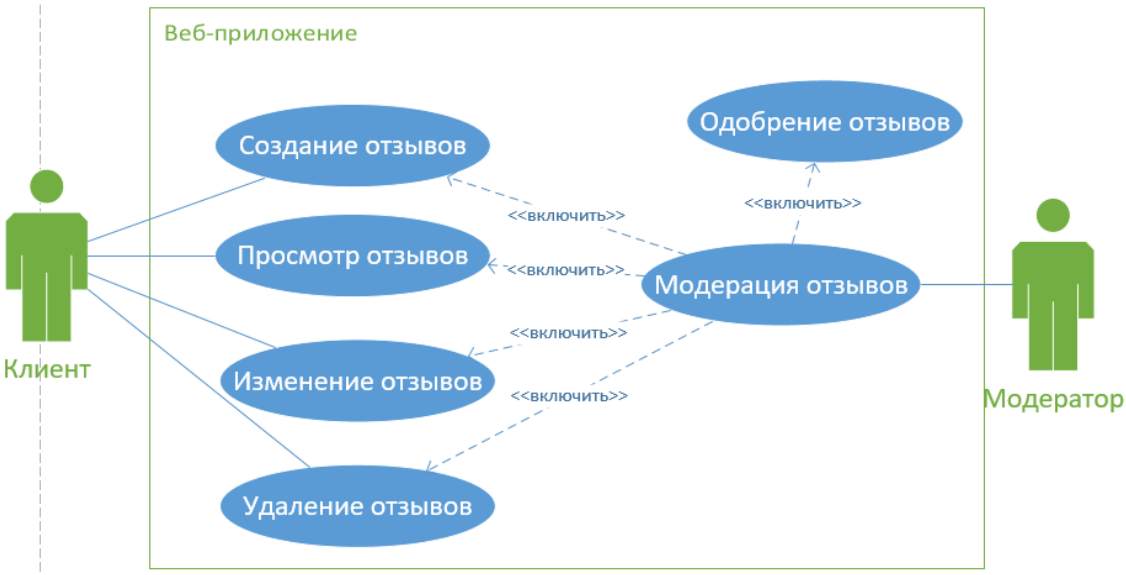


Рисунок 4 - Схема вариантов использования

На схеме вариантов использования можно увидеть основные требования к приложению:

– оставлять отзыв;

– удалять отзыв;

– просматривать отзывы;

– обновлять отзывы;

– одобрять отзывы (для уполномоченного персонала).

В соответствии с требованиями, приложение должно обладать понятным интерфейсом, быстродействием и корректным вводом/выводом. Также необходима роль модератора, способная проводить данные операции на всех отзывах.

Из требований можно вывести экономическую целесообразность проекта:

– удобство: веб-интерфейс обеспечивает отправку отзывов через браузер широкому спектру клиентов;

– безопасность: веб-интерфейс может использовать механизмы аутентификации и авторизации для защиты данных пользователей от несанкционированного доступа;

– гибкость и масштабируемость: веб-интерфейс может быть легко интегрирован с другими системами и сервисами;

– выигрыш во времени: использование веб-интерфейса позволяет разработчикам получить список требований, без дополнительных опросов;

Теперь можно дополнить модель бизнес-процесса в диаграмму в нотации idef0 (рисунок 5), добавив новый поток – обратную связь, выходящую из поддержки в анализ, обратным вызовом.

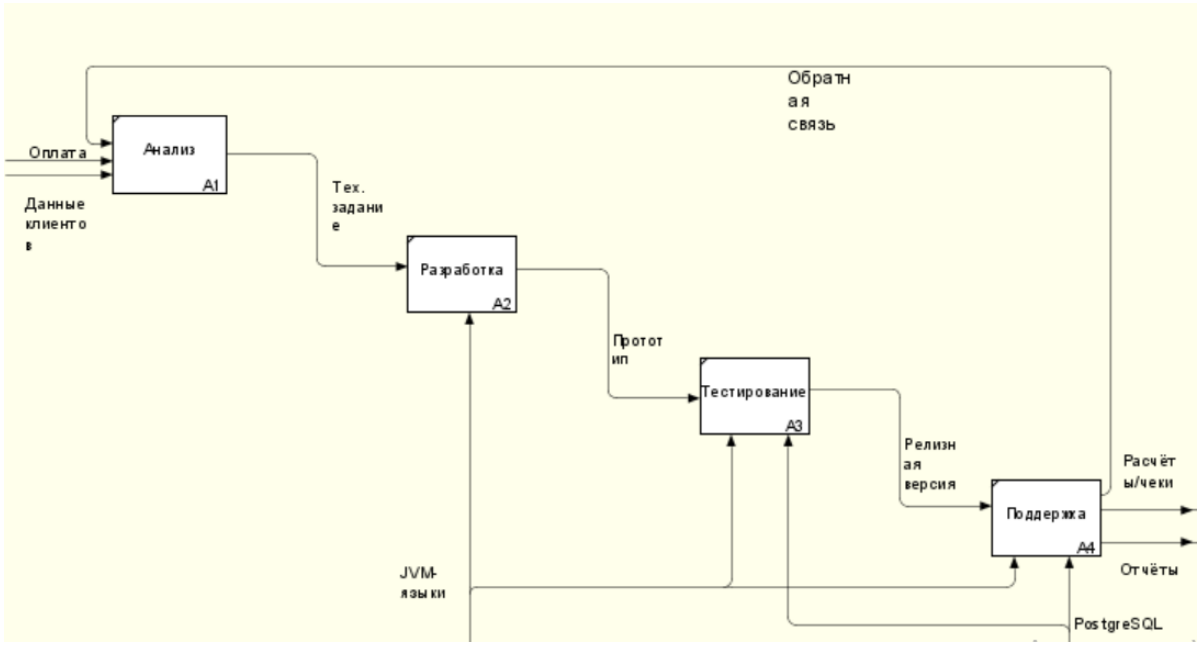


Рисунок 5 - Модель TO-BE

Работа нацелена на добавление нового потока данных – обратной связи от клиентов, которую смогут использовать служба экспертной поддержки и разработчики для улучшения проектирования обновлений и новых модулей системы [6].

Согласно аннотации IDEF0, новый поток выходит из процесса поддержки системы (из правой грани) и входит в процесс анализа (в левую грань) обратным вызовом [2]. Так клиенты смогут помогать команде поддерживать сервис в актуальном и удобном, для современного пользователя, состоянии.

В соответствии с контекстом новый канал должен использовать механизмы JVM, веб-технологии и реляционную базу данных (PostgreSQL) [8]. Так он не будет чужеродным для системы и команда компании сможет легко поддерживать данный модуль в дальнейшем.

1.3 Анализ существующих решений

Перед разработкой собственного решения важно рассмотреть уже существующие аналоги. Это позволяет выявить типичные подходы к реализации функциональности, а также определить их достоинства и недостатки с точки зрения пользователя. На рисунке 6 представлена существующая реализация раздела с отзывами на сайте компании, где отображаются ФИО, должность и место работы автора комментария.

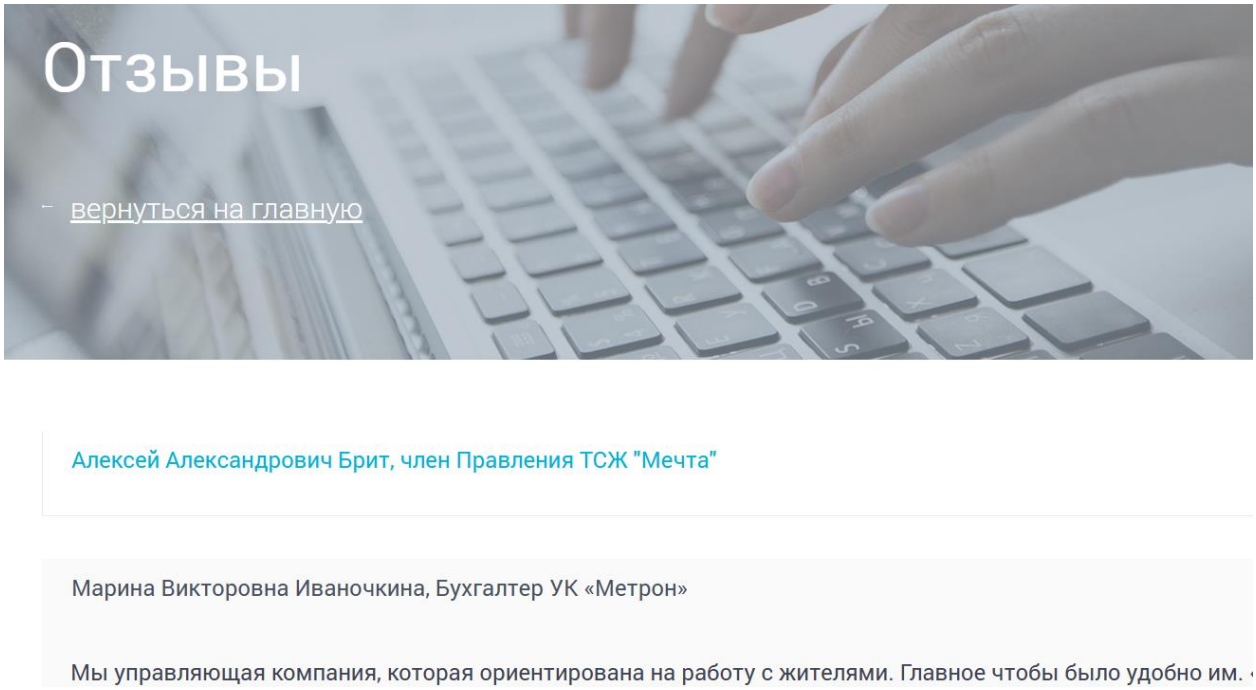


Рисунок 6 - Комментарии на странице

Компания уже имеет отзывы, направленные на общие впечатления. На странице видны ФИО, должность и место работы. Отзывы в основном оставляют представители управляющих компаний. Однако существенным недостатком является невозможность оставить отзыв напрямую, через форму, что ограничивает обратную связь. Такие отзывы могут дать однобокую картину, которая может привести к неверным решениям.

Клиенты способны дать ценную информацию, которую можно применить при улучшении, масштабировании и поддержке системы, но для этого нужна более простой и автономный веб-интерфейс.

В целом данное решение имеет следующие недостатки:

– ограниченность круга пользователей;

– отсутствие пользовательского интерфейса для отправки отзывов.

Так или иначе, общим недостатком является низкая доступность (accessibility), которая размывает впечатления о сервисах компании.

Тикетница HelpDeskEddy (рисунок 7).

Helpdesk система способствует контакту клиентов с поддержкой на всех этапах покупки товара: перед, в процессе и по завершении.

Система HelpDeskEddy распространяется по модели SaaS и является омниканальной - позволяет принимать заявки из электронной почты и разных мессенджеров (VK, Telegram), также компания предоставляет собственный виджет чата для размещения на сайте клиента. Абонентская плата зависит от того, сколько сотрудников будут обрабатывать заявки. На рисунке 7 показан интерфейс сервиса HelpDeskEddy, предоставляющего омниканальную систему обработки заявок с возможностью интеграции чата на сайт клиента.

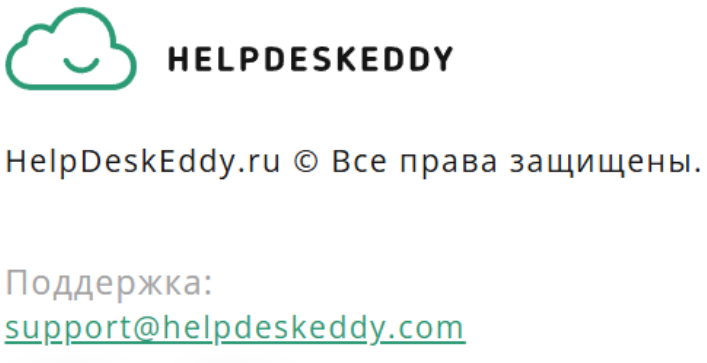


Рисунок 7 - HelpDeskEddy

Очевидным минусом является привлечение стороннего разработчика. Это осложняет обеспечение безопасности со стороны заказчика. Разработка

собственного портала для связи с клиентами обеспечит больший контроль над потоками данных компании.

1.4 Требования к функционалу и интерфейсу приложения.

По итогам анализа предметной области и требований заказчика, было составлено техническое задание, результатом которого является веб-приложение. Приложение должно обеспечивать пользователям веб-интерфейс и методы работы с базой данных, поддерживающие CRUD-операции [8].

Общие цели и задачи.

– вёрстка UI для работы с отзывами;

– реализация функций добавления, удаления, просмотра и обновления отзывов;

– обеспечить подключение к базе данных (PostgreSQL) для хранения информации о отзывах;

– обеспечение надежности, безопасности и удобства в использовании приложения.

1.4.2 Технические требования.

Требования к архитектуре приложения.

– основная модель взаимодействия - клиент-серверная архитектура с тонким клиентом;

– клиентская часть - разработана с использованием современных веб-технологий (HTML, CSS);

– серверная часть – платформа Java в издании Enterprise [17], [20]. Фреймворк Java Server Faces [8], генерирующий клиентскую часть на сервере, отправляя её клиенту в виде html. Бизнес-логика реализована на языке программирования Java, включая работу с базой данных и обработку запросов клиентов;

– база данных - PostgreSQL для хранения данных о пользователях и отзывах.

Функциональные требования.

Включают в себя CRUD-операции, распространённые при работе данными, а также возможности модерации.

– оставлять отзыв;

Необходима форма для заполнения отзыва, возможность прикрепления оценок и небольшой текстовой рецензии.

– удалять отзыв;

Предполагается кнопка для удаления пользователем ранее оставленного им отзыва. Рекомендуется подтверждение удаления для защиты от ошибок.

– просматривать отзывы;

Список всех опубликованных отзывов с указанием имени автора, оценки и текста отзыва может выводится на главную страницу. Также пользователь сможет смотреть все свои отзывы, в том числе не опубликованные.

– обновлять отзывы;

Возможность редактирования отзыва.

Пользователь должен иметь возможность изменять свою оценку.

Обновлять отзывы:

Возможность модерации отзывов для удаления запрещённого контента.

– роль модератора.

Модератор опубликовывает отзывы, которые будут появляться на странице

Нефункциональные требования.

– масштабируемость для добавления функционала и интеграции с существующими системами;

– высокая доступность и отказоустойчивость;

– интуитивно понятный интерфейс;

– обеспечение безопасности данных (шифрование соединения, защита от SQL-инъекций и т.п.).

Требования к интерфейсу.

Для выполнения поставленных задач, интерфейс веб-сайта должен предоставлять соответствующие формы и страницы. Была составлена концептуальная схема веб-сайта (рисунок 8).

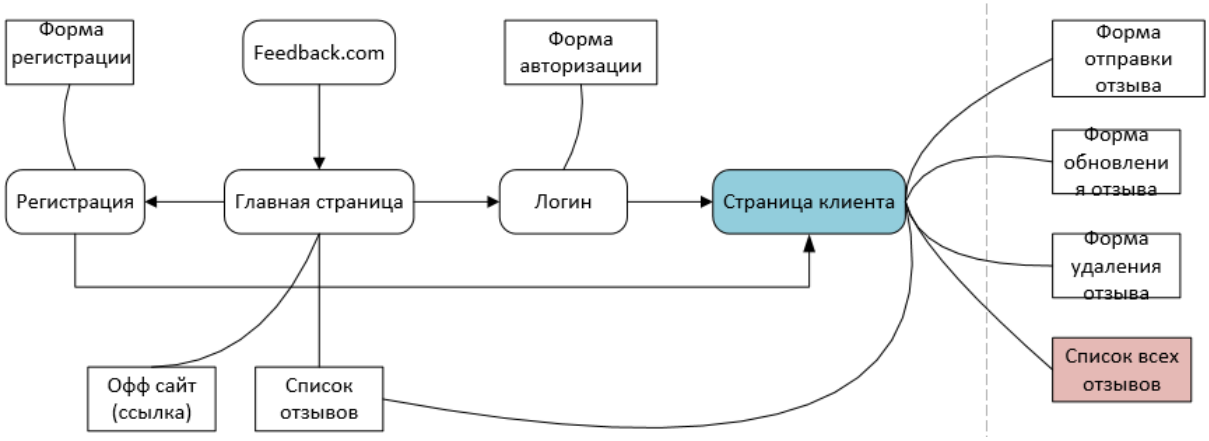


Рисунок 8 - Концептуальная схема сайта

Схема отражает основные требования заказчика к функционалу, изображая их в виде веб-страниц:

– feedback.com – точка входа в веб-приложение;

– улавная страница – начальная страница, откуда пользователь может попасть в основные разделы;

– список отзывов – элемент главной страницы, отображающий опубликованные отзывы. Также данный элемент отражается на странице пользователя;

– регистрация – переход на страницу регистрации с соответствующей формой.

Страницы могут содержать различные элементы, обозначенные прямоугольниками (с острыми краями):

– форма регистрации - элемент страницы регистрации;

– логин - страница, где пользователь входит в систему;

– форма авторизации - элемент страницы входа в систему;

– страница клиента - страница пользователя;

– форма отправки отзыва;

– форма обновления отзыва;

– форма удаления отзыва;

– офф сайт - ссылка на официальный сайт компании.

Соединительные линии:

– Стрелки – Переходы между страницами;

– Линии – Элементы, относящиеся к странице.

Синим обозначены блоки, доступные при авторизации, красным – только для пользователям-администраторам.

Вывод по главе 1.

В ходе первой главы были достигнуты важные результаты, соответствующие поставленным целям. Проведён углублённый анализ предметной области, включающий как текстовое описание, так и графическое представление с использованием диаграмм в нотациях IDEF0 и UML [2], [11]. Это позволило лучше понять специфику решаемых задач и определить ключевые компоненты будущей системы.

Анализ требований заказчика дал более чёткое представление о структуре и функциональности приложения. Выявленные требования были дополнены наглядными диаграммами, отражающими как функциональные возможности, так и особенности пользовательского интерфейса. Кроме того, было проведено изучение существующих решений, что позволило обосновать необходимость разработки собственного программного продукта.

На основе проделанной работы было сформировано техническое задание, полностью соответствующее ожиданиям заказчика и отражающее выявленные потребности [3]. Полученные результаты отвечают заявленным целям главы и создают прочную основу для последующего этапа проектирования системы.

Глава 2. Процесс разработки программного обеспечения

2.1 Архитектура программного обеспечения

Многослойная архитектура (программного обеспечения) – это подход к проектированию приложения, основанный на распределении его логики по слоям. Каждый слой выполняет определённые функции, что позволяет каждому из них быть в некоторой степени отдельным объектом, абстрагируясь от других слоёв для более детального и практичного планирования разработки, так как это разделение позволяет в то же время лучше понимать структуру всего приложения и повторно использовать код. MVC (Model, View, Controller) – один из популярнейших архитектурных шаблонов для многослойной системы. На рисунке 9 представлена схема архитектурного шаблона MVC, отражающая взаимодействие между пользователем, представлением, контроллером и моделью.

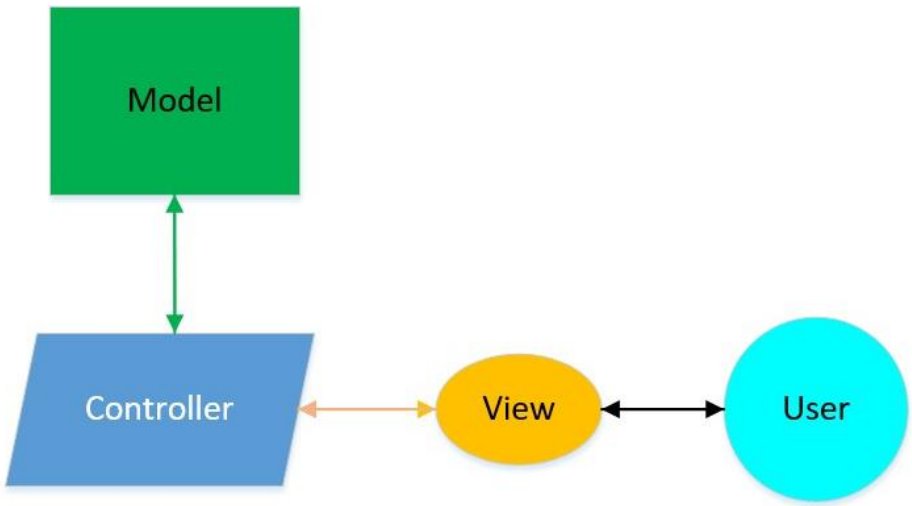


Рисунок 9 - Схема MVC

– Model. Модель, хранящая и обрабатывающая данные. Хранит в себе необходимые процедуры, вызываемые контроллером (Controller) по запросу от представления (View);

– View. Представление. Отображает модель посредством контроллера и получает запросы/данные от пользователя, передавая их контроллеру (Controller);

– Controller. Контроллер. Получая данные от представления, определяет какую процедуру вызвать в модели и какое представление отправить в ответ (пользователю для дальнейшей работы).

На рисунке 8 предоставлена диаграмма пакетов, представляющая общий вид архитектуры, с перечислением основных составляющих:

– Jakarta Enterprise Edition [15]. Набор спецификации разработки корпоративных приложений на базе Java, в частности серверной части;

– Java Server Faces. Фреймворк пользовательского веб-интерфейса. Предоставляет набор спецификаций для работы клиент-серверной архитектуры с помощью системы компонентов;

– WildFly [15]. Сервер, на котором будет развёрнуто приложение;

– Controllers. Контроллеры обеспечивают интерфейс между клиентом и бизнес-логикой. Навигация между страницами, обработка запросов итп;

– Tech. Пакет с классами, реализующими бизнес-логику приложения в виде сервисов. Они слой контроллера, взаимодействующий с моделью;

– Models. Пакет с ORM моделями;

– Persistence. Программный интерфейс, нужный для управления ORM в Java;

– Tech_sys. Пакет стандартных библиотек и модулей java, нужных для работы с веб-интерфейсом и базами данных.

На рисунке 10 представлена диаграмма пакетов, отображающая общую архитектуру системы, включая взаимодействие между контроллерами, сервисами, моделями и системными компонентами.

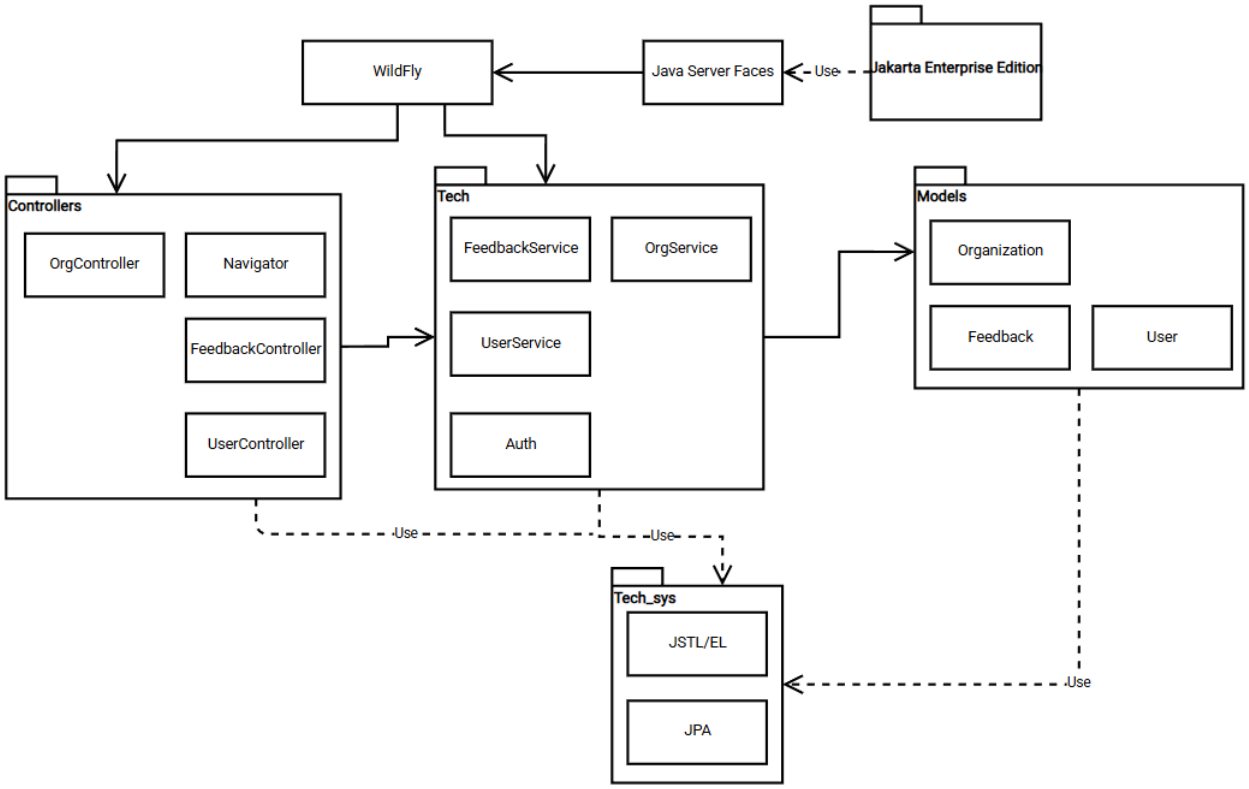


Рисунок 10 - Диаграмма пакетов общего вида архитектуры

Схемы архитектуры приложения дают общее представление о разрабатываемом программном продукте [9]. На диаграмме пакетов можно выделить слои архитектурного шаблона MVC:

– Модели данных;

– Представление, посредством фреймворка JSF и EL;

– Контроллер. В виде контроллеров для связи с представлением и сервисов для связи с моделью.

2.2 База данных

База данных должна соответствовать потребностям организации в рамках модели предметной области [5].

На рисунке 11 изображена схема данных, которая будет использована при разработке базы данных.

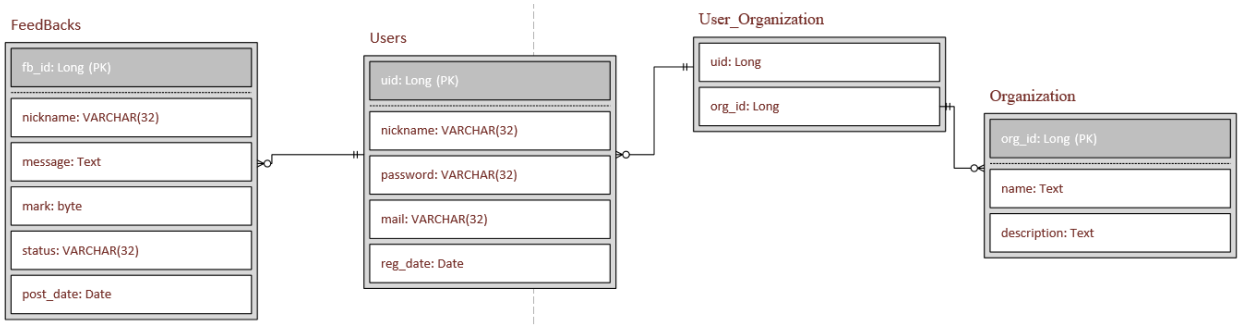


Рисунок 11 - База данных на диаграмме классов

В неё входят основные классы:

а) Feedbacks. Непосредственно отзывы, содержат следующие атрибуты:

1) Fb_id. Идентификатор комментария. Необходим для однозначной идентификации отзыва в базе данных, должен быть уникален у каждого отзыва, как и остальные идентификаторы.

2) Nickname. Владелец комментария;

3) Message. Текст комментария;

4) Mark. Оценка;

5) Status. Статус отзыва, необходим для модерации;

6) Post_date. Дата публикации.

б) Users. Данные пользователя:

1) Uid. Идентификатор пользователя. Необходим для однозначной идентификации пользователя в базе данных;

2) Nickname. Символьное имя пользователя;

3) Password. Пароль к аккаунту;

4) Mail. Электронная почта;

5) Reg_date. Дата регистрации.

в) User_organization. Вспомогательная сводная таблица связи клиента и некой организации:

1) Uid. Идентификатор пользователя, ссылается на аналогичный столбец у пользователя;

2) Org_id. Идентификатор организации.

г) Organization. Организация, к которой может относиться пользователь:

1) Org_id. Однозначный идентификатор организации;

2) Name. Название организации;

3) Description. Возможное описание организации.

2.3 Выбор технологий и инструментов для разработки

Jakarta EE - мощная платформа для разработки больших корпоративных приложений, предоставляет стандартные компоненты для разработки корпоративных решений, включая слой, слой бизнес-логики и слой доступа к данным (POJO). JEE позволит привести объекты представления, контроллера и модели к единому стилю и обеспечить больший контроль над реализацией для дальнейшей отладки и развития [17].

При этом модульность системы позволяет адаптировать проект под разные нужды и технологии Java. По сути, это фундамент веб-приложения, предоставляющий библиотеки для работы с веб-сервисами REST, запросами и жизненным циклом приложения на Java [10].

В рамках платформы поддерживаются и развиваются такие технологии как JPA, JTA, фреймворк JSF итп.

Java Persistence API (JPA) – спецификация для работы с базами данных на Java [16].

В отличие от JDBC – коннектора, позволяющего приложению подключаться к базе данных и отправлять туда запросы SQL, JPA использует

объектно-реляционный маппинг (ORM) (отображение), чтобы формировать эти SQL-запросы автоматически, с экранированием [19].

Давним предком является JDBC – коннектор, позволяющий приложению подключаться к базе данных и отправлять туда запросы SQL. После появился Hibernate, вносящий унифицированный подход к «персистентности» Java-объектов [13].

Теперь, запрос SQL создаётся и отправляется автоматически, используется отображение через поля класса, само подключение класса как запись базы данных сначала происходило через к xml файл с настройками, но современная версия JPA делает это через аннотации в самом коде класса. Этот подход стал основой для JPA. Теперь разработчики могут сосредоточится на самих запросах, а не их формах, отличающихся в каждой СУБД.

Но данная спецификация накладывает определённые требования на поля класса, чтобы JPA мог считать его и использовать при работе с базой данных:

– классы должны быть помечены аннотацией @Entity;

– классы должны иметь первичный ключ, который должен быть помечен аннотацией @Id;

– классы могут иметь дополнительные атрибуты, которые должны быть помечены аннотациями @Column;

– классы обязаны иметь методы доступа к данным - геттеры и сеттеры. Через них будут меняться значение записи.

Согласно задаче, функции JPA будет использоваться при работе с базой данных. Классы java были оформлены в соответствии со спецификацией и стали моделями данных или глубоким слоем контроллера, согласно архитектуре MVC, отвечающим за отправку запросов в базу данных.

Java Transaction API – спецификация для распределённых транзакций. Она описывает интерфейсы взаимодействия между менеджером транзакций и другими агентами, использующими транзакции, что очень важно в контексте работы с базами данных на веб-приложении, так как данная спецификация разграничивает транзакции.

Собственно, JTA обеспечивает механизмы обеспечения целостности данных в условиях конкурентного доступа, используется совместно с JPA.

Java Server Faces – фреймворк для разработки веб-приложений на платформе Java. Он позволяет создавать интерактивные веб-страницы (фронтенд) на стороне сервера, способные более тесно работать с бэкендом. В архитектуре MVC отвечает за слой представления.

Особенности JSF:

– Использование компонентов. Технология позволяет строить веб-страницы с помощью компонентов, которые можно настроить и использовать повторно;

– Механизм привязки модели. Позволяет связать компоненты интерфейса с бэкенд-логикой приложения, без отправки дополнительных запросов;

– Поддержка AJAX. Позволяет добавлять динамические элементы на веб-страницы, изменяющиеся без перезагрузки всей страницы;

– Простота интеграции. JSF легко интегрируется с другими популярными технологиями, такими как EJB или шаблонами проектирования, как MVC.

EJB – спецификация технологии и может иметь много интерпретаций. В целом, она создана для распределения логики приложения, поэтому используется в веб-разработке, так как имеет множество уже готовых библиотек и модулей, благодаря которым можно сосредоточится на логике, а не способах реализации персистентности и транзакций (при работе с базами данных).

«Бины”» – программные компоненты, как правило, классы, написанные по спецификации и выполняющие определённые задачи.

Три основных типа бинов:

– Message Driven Beans. Они не вызываются напрямую, а ждут сообщения (к примеру, от Java Message Service) – реализуют асинхронный обмен сообщений между частями системамы;

– Entity Beans (объектные бины) — определены в спецификации JPA (Java Persistence API) и используются для хранения данных, могут представлять из себя модель таблицы базы данных, в виде класса Java, и быть переданы в неё в виде записи через транзакцию;

– Session Beans (сессионные бины). Чаще используются для реализации логики, к примеру произведения транзакций – сохранение EntityBean в базу данных, как запись.

Виды Session Bean:

– Stateless. Не хранят состояние сессии с клиентом. Каждый вызов бина в рамках запроса – новый класс EJB, взятый из автоматически заполняемого пула и удаляется при бездействии. Они создаются при необходимости контейнером и хранятся в пуле для оптимизации вычислений;

– Stateful. Хранят состояние, Уникальны для каждого клиента и живут пока диалог не прекратится;

– Singleton. Один экземпляр на всё приложение;

– В архитектуре MVC, EJB часто выступают в роли контроллера, являясь посредником между моделью и другими компонентами слоя контроллера или представления.

Предпринято использовать EJB SessionBean в контроле транзакций с использованием JPA. В классе EJB SessionBean может быть определён менеджер сущностей EntityManager, транзакция EntityTransaction и метод create([аргументы]), выполняющий транзакцию в базу данных посредством этих двух компонентов. В качестве модели данных – EntityBean [14].

Session Bean должен быть использован в классах сервлета и сервиса, так как он уже имеет логику (реализацию транзакции) и осталось лишь обеспечить вызов этих методов с интерфейса пользователя или от других сервисов, сервлетов.

CDI – Context Dependency Injection. Технология внедрения зависимостей в зависимости от контекста. Может использоваться совместно с EJB. Имеет доп аннотации – например, областb видимости:

– Запроса. Почти как Stateless EJB;

– Сессии. Примерно Stateful;

– Приложения. Как Singleton.

Основатель Opensource Java EE отметил, что противопоставление EJB и CDI некорректно, так как они разрабатываются для общих целей.

«В будущих версиях Java EE мы продолжим их согласование. Выравнивание означает возможность людям делать то, что они уже могут делать, просто без аннотации или аннотации вверху» [4].

CDI более динамичен и может быть полезен при необходимости поддержки зависимостей, событий, перехватчиков, декораторов и отслеживание жизненного цикла. EJB же более неповоротлив и статичен, но может быть более прост в реализации.

2.4 Описание решаемых задач разработки

Стек технологий и инструменты разработки:

– Apache NetBeans 23. IDE для разработки кода и упрощённого менеджмента проекта;

– Версия Java 22 SDK, доступная IDE;

– Jakarta EE 11 Web, также доступная IDE;

– EL/HTML/CSS;

– PostgreSql Server. Сервер базы данных, видимый в IDE;

– Сервер WildFly 34.0.0 с подключением к PostgreSql.

Порядок работы:

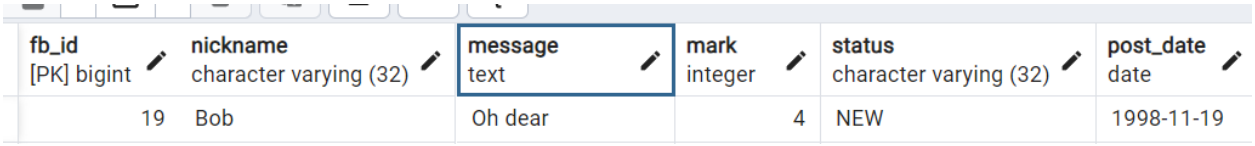
– Установить PostgreSQL и создать базу данных с необходимыми данными;

– Установить и настроить сервер WildFly для работы с PostgreSQL;

– Создать веб-приложение в netbeans.

Установка PostgreSQL Server.

Для симуляции рабочей среды, на ЭВМ был установлен PostgreSql Server с соответственной системой управления базами данных, она будет хранить и обрабатывать данные. После была создана новая база с таблицей отзывов (рисунки 12-13).



fb_id [PK] bigint	nickname character varying (32)	message text	mark integer	status character varying (32)	post_date date
19	Bob	Oh dear	4	NEW	1998-11-19

Рисунок 12 – Окно таблицы отзывов в pgAdmin

Данные таблицы:

– Fb_id. Первичный ключ с автоинкрементацией будет использоваться для уникального идентификатора отзыва;

– Nickname. Имя почта пользователя;

– Message. Сообщение некоторого размера;

– Mark. Оценка включает в себя число 2(плохо)-5(отлично);

– Status. Статус отзыва. Одобряются модераторами, помечаются как удалённые итп;

– Post_date. Дата.

Это минимальные данные, которые позволят системе соответствовать бизнес-требованиям по хранению данных об отзывах пользователей. Далее необходимо установить сервер приложения, который сможет работать с сервером базы данных.

Установка WildFly 34.0.0. Сервер WildFly был установлен в корневую папку системного диска и стандартной конфигурации вполне хватает для его работы. Однако для работы с СУБД требовалось добавить драйвер для PostgreSQL [12] и источник данных – конкретную базу в установленном PostgreSQL (рисунок 13).

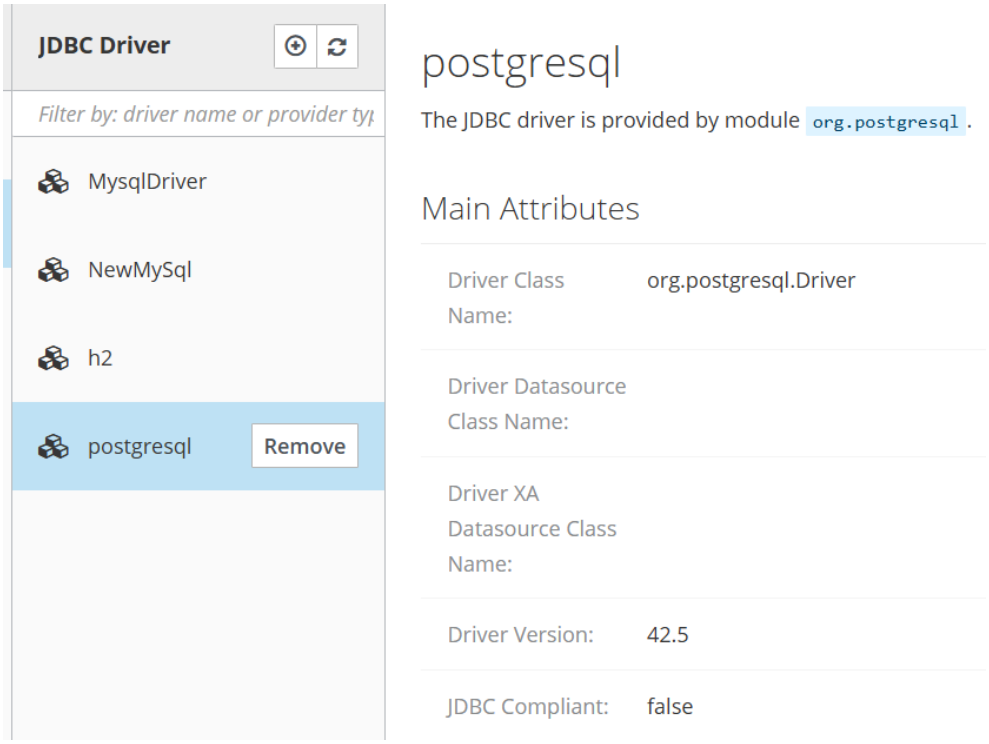


Рисунок 13 - Добавленный драйвер в меню конфигурации

Установка драйвера была проведена правильно - он будет виден на панели WildFly, в списке драйверов jdbc, что и продемонстрировано в меню «подсистемы». Это было необходимо, чтобы сервер и программы, развёрнутые на нём, могли обращаться к серверу PostgreSQL.

Следующим шагом стало добавить источник данных (рисунок 14).

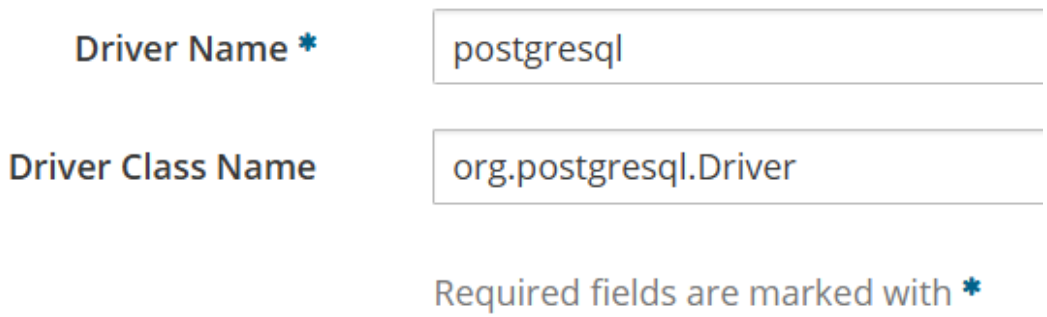


Рисунок 14 – Указание драйвера для источника данных

В интерфейсе было необходимо указать название подключения, имя ресурса, URL подключения и драйвер, добавленный ранее. URL указывает, какая именно база данных сервера PostgreSQL будет использована в качестве хранилища данных. Источник был указан серверу (рисунок 15).

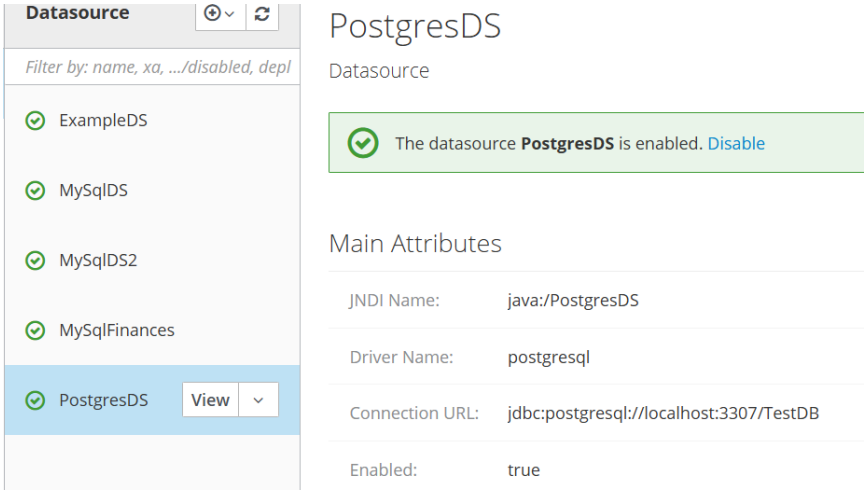


Рисунок 15 - Добавленный источник данных WildFly

Далее в меню конфигурации подсистем можно увидеть основные настройки источника данных. Была подключена база данных «TestDB» и она будет доступна для серверных приложений посредством сервера WildFly.

Для большей безопасности, сервер будет подключаться к базе данных как обычный пользователь.

Панель pgAdmin позволила создать пользователя joe, у которого нет прав супер-пользователя, но будут права на CRUD операции с таблицами отзывов и пользователей:

– создавать записи;
– читать записи;
– обновлять записи;
– удалять записи.

При создании использовалась команда CREATE ROLE. На рисунке 16 показан пример создания роли пользователя в pgAdmin с использованием SQL-команды CREATE ROLE.

```
CREATE ROLE joe WITH
  LOGIN
  NOSUPERUSER
  ENCRYPTED PASSWORD 'SCRAM-SHA-256$4(
```

Рисунок 16 - Создание роли в pgAdmin

С помощью командной строки, от имени супер-пользователя, были добавлены необходимые права для joe на таблицу feedbacks (рисунок 17):

```
TestDB=# grant insert ON feedbacks to joe;
GRANT
TestDB=# grant select ON feedbacks to joe;
GRANT
TestDB=# grant update ON feedbacks to joe;
GRANT
TestDB=# grant delete ON feedbacks to joe;
GRANT
```

Рисунок 17 - Выдача прав роли на таблицу feedbacks

В эти права входят следущие операции:

– INSERT. Внесение записей в таблицу;
– SELECT. Выборка записей из таблицы
– UPDATE. Изменение записей;
– DELETE. Удаление записей.

Аналогично предоставлен доступ к таблице users (рисунок 18):

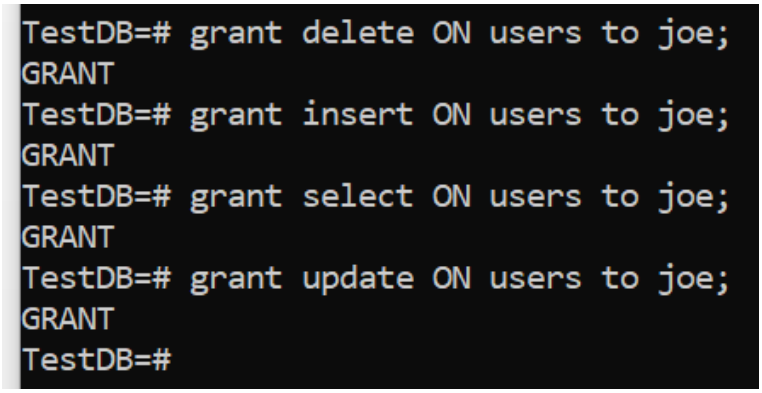


Рисунок 18 - Выдача прав роли на таблицу users

Пользователь joe имеет стандартные права, от его имени будут совершаться все операции с базой данных. Права же супер-пользователя не будут доступны серверу.

Кодирование веб-приложения.

Следующим этапом было создание проекта в netbeans, сборщиком maven. Приложение использует многослойный шаблон MVC, разделяющий логику на слои для большей модульности приложения.

Конкретная реализация шаблона в приложении:

– Представление считывает данные пользователя и передаёт их контроллеру, а также выводит нужные данные на страницу;

– Контроллер выступает посредником между представлением и моделью. Формулирует запросы к модели и выдаёт результат представлению;

– Модель меняет своё состояние и выдаёт данные по запросам контроллеру.

NetBeans IDE может работать с серверами приложений по типу WildFly. Запущенное приложение будет автоматически развёртываться на сервере, что значительно упростило разработку и тестирование (рисунок 19).

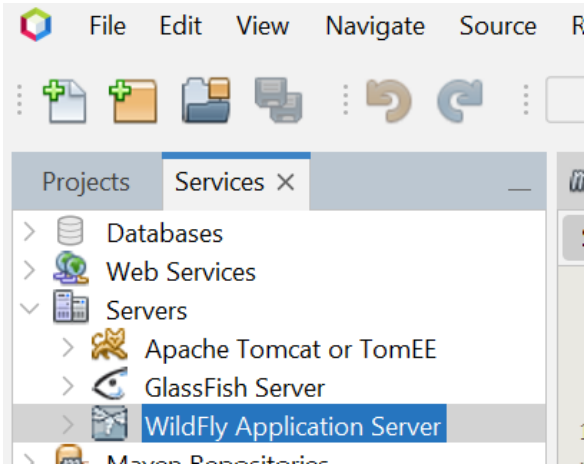


Рисунок 19 - Сервер WildFly интегрирован с NetBeans IDE

Большая часть приложения была сгенерирована IDE, в данном случае использован шаблон веб-приложения. Он имеет базовые настройки и стартовую страницу «Hello World!», которую было необходимо заменить файлом главной страницы приложения.

Ключевым шагом является настройка доступа приложения к базе данных с помощью механизмов JPA, которые хранятся в конфигурационном файле persistence.xml (рисунок 20).

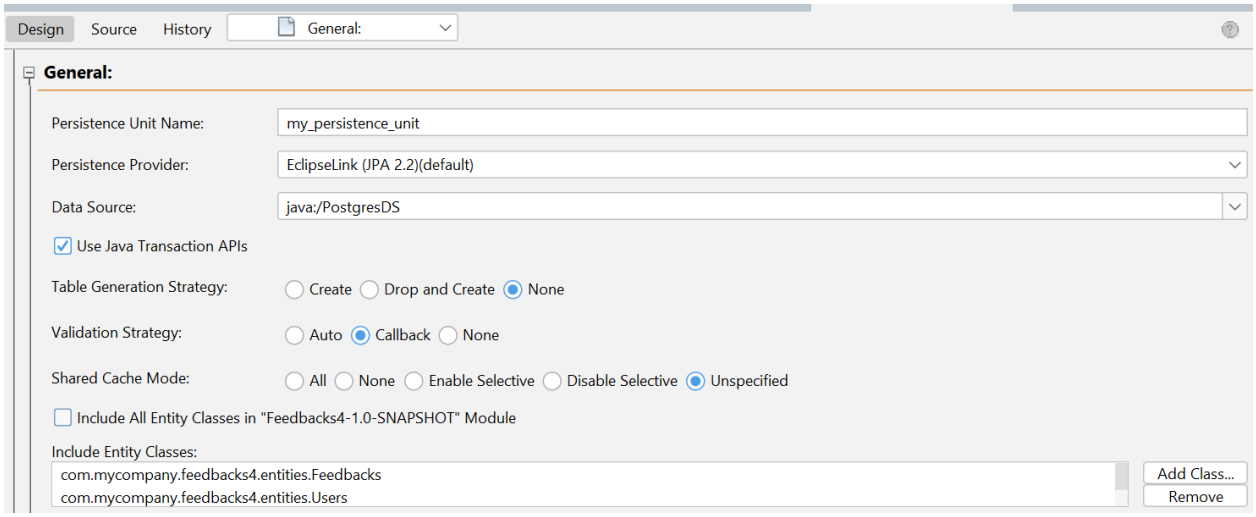


Рисунок 20 - persistence.xml

В дескрипторе persistence.xml указаны параметры JPA.

Сырая версия файла (рисунок 21) содержит те же настройки.

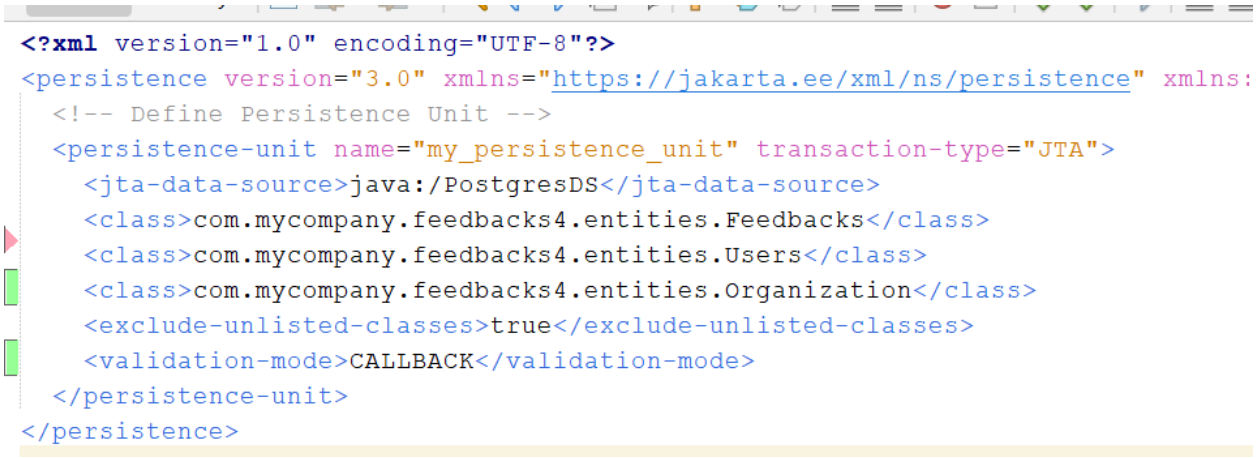


Рисунок 21 - Сырой persistence.xml

В списке «JTA Data Source» указывается соединение с именем JNDI источника данных WildFly, настроенного ранее – в данном случает это источник с базой данных PostgreSQL. Также в файл включены классы-сущности Feedbacks и Users, указан тип транзакций с использованием JTA. Остальное по умолчанию.

Компоненты бизнес-логики отзыва. Бизнес-логика приложения разделена между классами Java, в соответствии с назначением и ролью каждого компонента в системе.

Класс Feedbacks (рисунок 22) содержит сущность таблицы базы данных, которая является реализацией ORM.

Рисунок 22 - Заголовок класса-сущности

В созданном файле видны аннотации JPA [12]:

– @Entity – класс является сущностью и будет управляться контейнером сервера;

– @Table – связывает данный класс с таблицей «clients»;

– @NamedQueries({@NamedQuery}) – Готовый запрос с кратким названием или список запросов для их дальнейшего использования;

– @Id – указывает на поле с идентификатором (первичным ключём);

– @GeneratedValue – этот идентификатор будет сгенерирован автоматически базой данных;

– @Basic – указывает на обычный маппинг JPA, аннотация позволяет обозначить fetch - загрузку в ленивом режиме fetch.LAZY (fetch.EAGLE по умолчанию) и optional – может ли быть null;

– @Column – данный атрибут является столбцом таблицы с данным именем;

– @Size – максимальная длинна строки;

– @NotNull – указывает валидатору, что поле не может быть пустым.

Полный листинг в приложении 1 показывает все методы класса, а также используемые библиотеки.

Далее было необходимо создать класс, отвечающий за работу с базой данных. В нём будут описаны методы добавления данных в базу.

На рисунке 23 представлен фрагмент кода, демонстрирующий объявление Session Bean с использованием аннотации `@Stateless` для класса `FeedbackService`.

```
//@ApplicationScoped
@Stateless
public class FeedbackService {
    EntityManager em = Persistence.createEntityManagerFactory("my_persistence_unit").createEntityManager();

    /**
     * Creates a new instance of FeedbackService
     */
    public FeedbackService() {
    }
```

Рисунок 23 - Заголовок Session Bean

Класс FeedbackService – это реализация Session bean, она нужна для некоторой бизнес-логики, которую можно будет использовать в других местах. Так можно избежать написания большого количества одного и того же кода. Аннотация EJB @Stateless означает, что класс будет иметь неопределённое количество экземпляров, это может содействовать быстродействию, но повысит конкурентность, что компенсируется использованием транцакций.

В классе определены дополнительные классы, отвечающие за транзакции:

– EntityManager. нужен для подключения к базе данных. Его инициализация проводится посредством EntityManagerFactory с указанием имени persistence unit, определённого в persistence.xml;

– EntityTransaction. позволяет оградить операции друг от друга и обезопасить конкуренцию за данные.

На рисунке 24 показан метод `createfeedback`, реализующий сохранение отзыва в базе данных с использованием JPA и управления транзакциями.

```
public synchronized void createfeedback(String nick, String message, int mark) {
    if (!isNewFeedBack(nick,message)) {
        return;
    }
    Feedbacks fb = new Feedbacks();
    fb.setNickname(nick);
    fb.setMessage(message);
    fb.setMark(mark);
    fb.setPostDate(new Date());

    EntityTransaction et = em.getTransaction();
    et.begin();
    em.persist(fb);
    et.commit();

}
```

Рисунок 24 - Метод сохранения отзыва в базе

Метод createfeedback() принимает почту, сообщение и оценку, создавая внутри себя экземпляр класса Feedbacks и сохраняя его в базу методом persist() класса EntityManager.

Метод отмечен ключевым словом synchronized, которое обозначает, что метод доступен лишь одному потоку за раз. Так пользователи не смогут отправлять данные на сайт «одновременно», что пожертвует быстродействием ради надёжности.

Созданный отзыв получает данные – ник пользователя, сообщение, оценка и дату, созданную по текущей.

На рисунке 25 представлен метод `isNewFeedBack`, реализующий проверку на уникальность отзыва по адресу электронной почты.

```
public boolean isNewFeedBack(String check_mail) {
    List<Feedbacks> fbl = em.createNamedQuery("Feedbacks.findAll",Feedbacks.class).getResultList();

    for (Feedbacks f:fbl) {
        if (check_mail.equalsIgnoreCase(f.getMail())) return false;
    }

    return true;
}
```

Рисунок 25 - Метод проверки отзыва на новизну

Метод isNewFeedBack() не проводит транзакции, а просто проверяет, есть ли введённое имя пользователя в базе, поэтому использование synchronized не нужно.

На рисунке 26 показан метод `getDataList`, который формирует список всех опубликованных записей, отфильтрованных по статусу `APPROVED`.

```
public List<Feedbacks> getDataList() {
    //StringBuilder fbstr = new StringBuilder();
    List<Feedbacks> fb_all = em.createNamedQuery("Feedbacks.findAll",Feedbacks.class).getResultList();

    return fb_all.stream().filter(f->f.getStatus() != null && f.getStatus().equals("APPROVED")).toList();
    //return fb_all;
}
```

Рисунок 26 - Метод получения всех опубликованных записей списком

Метод getDataList посылает модели запрос на сбор всех записей из таблицы базы данных, после чего помещает результат в список, доступный на представлении.

На рисунке 27 представлены методы `updateFeedback` и `deleteFeedback`, реализующие обновление и удаление записей в компоненте-сервисе с использованием транзакций и проверки существования сущности.

```
public synchronized void updateFeedback(Feedbacks feedback) {
    Feedbacks fb = em.find(Feedbacks.class, feedback.getFbId());
    if (fb == null) {
        throw new NotFoundException("Feedback with id " + feedback + " not found");
    }
    EntityTransaction et = em.getTransaction();
    et.begin();
    em.merge(feedback);
    et.commit();

}

public synchronized void deletefeedback(Long id) {
    Feedbacks fb = em.find(Feedbacks.class, id);
    if (fb == null) {
        throw new NotFoundException("Feedback with id " + id + " not found");
    }
    EntityTransaction et = em.getTransaction();
    et.begin();
    em.remove(fb);
    et.commit();

}
```

Рисунок 27 - Обновление и удаление в компоненте-сервисе

Компонент «FeedbackBean» служит коммуникатором между пользовательским интерфейсом и моделью данных (рисунок 28).

```
@Named(value = "feedbackBean")
@RequestScoped
public class FeedbackBean {
    @Inject
    FeedbackService fb_serv;

    String name,message;
    int mark;
    String status;
    /**
     * Creates a new instance of FeedbackBean
     */
    public FeedbackBean() {
    }

    public void createFeedbacks(String owner) {
        //mark = 0;
        fb_serv.createfeedback(owner, message, mark);
        message = "";
        mark = 0;
    }
```

Рисунок 28 - Заголовок FeedbackBean

Заголовок также имеет аннотации:

– @Named. Название компонента, для доступа к нему со страниц xhtml;

– @RequestScoped. Компонент существует только в рамках одного запроса;

– @Inject. «Инъектор» – аннотация CDI, указывающая, что переменная ссылается на существующий в контексте экземпляр класса FeedbackService.

Метод создания отзыва просто передаёт информацию на сервис FeedbackService, обнуляя при этом свои данные, чтобы не переслать одно и то же дважды.

Для прочих CRUD-операций написаны аналогичные функции. Для получения опубликованных записей (рисунок 29).

```
public List<Feedbacks> getFeedbacks() {
    return fb_serv.getDataList();
}
```

Рисунок 29 - Получение всех записей

Вызывается метод компонента-сервиса на получение опубликованных записей.

На рисунке 30 представлены методы `updateFeedback` и `deleteFeedbacks`, реализующие обновление и логическое удаление записей путём изменения их статуса. Метод `updateFeedback` устанавливает статус 'NEW' для новой записи, тогда как метод `deleteFeedbacks` помечает запись как 'DELETED' и передаёт её на обновление в сервисный компонент `fb_serv`, без фактического удаления из базы данных.

```
public void updateFeedback(Feedbacks new_fb) {
    //mark = 0;
    new_fb.setStatus("NEW");
    fb_serv.updateFeedback(new_fb);
}

public void deleteFeedbacks(Feedbacks new_fb) {
    //mark = 0;
    new_fb.setStatus("DELETED");
    //fb_serv.deletefeedback(id);
    fb_serv.updateFeedback(new_fb);
}
```

Рисунок 30 - Обновление и удаление записей

После обновления статус отзыва меняется на «Новый», для повторного подтверждения. Удаление является сменой статуса отзыва на «Удалён».

Для регистрации и авторизации пользователей требовалось создать аналогичные компоненты.

Компоненты бизнес-логики пользователя.

UserService (рисунок 31) осуществляет CRUD операции с пользователями. Также используются аннотация @Stateless и классы JPA.

```
@Stateless
public class UserService {
    EntityManager em = Persistence.createEntityManagerFactory("my_persistence_unit").createEntityManager();

    /**
     * Creates a new instance of UserService
     */
    public UserService() {
    }
```

Рисунок 31 - Заголовок UserService

Метод checkAuthorize (рисунок 32) проверят существование пользователя в базе данных.

```
public boolean checkAuthorize(String name, String pass) {
    Query query_name = em.createNamedQuery("Users.findAll");
    List<Users> ulst = query_name.getResultList();
    for (Users u : ulst) {
        if (u.getNickname().equals(name) && u.getPassword().equals(pass)) {
            return true;
        }
    }
    return false;
}
```

Рисунок 32 - Проверка пользователя

Вспомогательная функция checkAuthorize позволяет проверять наличие пользователя в таблице Users по имени и паролю. Метод возвращает значение Boolean – истина/ложь. Сервис будет вызван следующим компонентом-контроллером для обмена данными с базой данных.

Создание пользователя (рисунок 33) также использует менеджеров транзакций и сущностей соответственно.

```
public void createUser(UserBean user) {
    Users u = new Users();
    u.setNickname(user.getName());
    u.setPassword(user.getPassword());
    u.setEmail(user.getEmail());

    EntityTransaction et = em.getTransaction();
    et.begin();
    em.persist(u);
    et.commit();
}
```

Рисунок 33 - Создание пользователя

Метод принимает экземпляр Userbean (рисунок 34), хранящий необходимую информацию.

```
@Named(value = "userBean")
@SessionScoped
public class UserBean implements Serializable{

    @Inject
    UserService user_service;

    @Size(max = 32)
    private String name;
    @Size(max = 32)
    private String password;
    @Email
    private String email;

    /**
     * Creates a new instance of UserBean
     */
    public UserBean() {
        name = "noname";
        password = "123";
        email = "noname@cc.ff";
    }
```

Рисунок 34 - Фрагмент UserBean

Класс UserBean также является компонентом-контроллером, позволяющим обмениваться данными с пользователем через представление. Но в данном классе аннотация @RequestScoped заменена на @SessionScoped, так как в отличие от контроллера отзыва - информация о пользователе должна хранится на сервере на протяжении всей сессии.

Переменная user_service ссылается на экземпляр UserService, созданный или существующий в веб-контейнере, используя аннотацию @Inject. Через неё будут вызываться различные методы компонента-сервиса (рисунок 35).

```
public boolean checkUser() {
    return user_service.checkAuthorize(name,password);
}

public void SignUpUser() {
    user_service.createUser(this);
    password = "";
    name = "";
}
```

Рисунок 35 - Дополнительные функции UserBean

Функции checkUser() и SignUpUser() ссылаются на экземпляр UserService и передают ему свою информацию – имя, пароль или свой же экземпляр (this).

Контроллер навигации. Для навигации по сайту было решено использовать компонент-контроллер Navigator, представленный соответствующим классом (рисунок 36). Его задача – отвечать на клики пользователя по кнопкам, обрабатывая информацию определённым образом, если требуется.

```
@Named(value = "navigator")
@RequestScoped
public class Navigator {
    @Inject
    FeedbackBean fbb;
    @Inject
    UserBean user;
    /**
     * Creates a new instance of Navigator
     */
    public Navigator() {
    }
```

Рисунок 36 - Заголовок Navigator

В заголовке прописана зависимость переменных и ранее определённых компонентов по спецификации CDI. «Навигатор» не хранит информацию напрямую, а лишь перенаправляет данные по нужному адресу (рисунок 37).

```
public String makeLogOut() {
    FacesContext context = FacesContext.getCurrentInstance();
    HttpServletRequest request = (HttpServletRequest) context.getExternalContext().getRequest();
    //session.removeAttribute("CSRF_TOKEN");
    user.setEmail(null);
    user.setName(null);
    user.setPassword(null);

    return "/index.xhtml";
}

public String toOffice() {
    if (user.checkUser()) {
        return "/pages/office";
    }
    return null;
}
```

Рисунок 37 - Методы выхода и логина Navigator

На рисунке видны методы:

– makeLogOut() – Очищает данные контроллера пользователя и переходит на главную страницу;

– toOffice() – Переходит в кабинет пользователя, если компонент UserBean нашёл пользователя в базе, в противном случае никуда не переходит.

Также имеется серия методов, выполняющих лишь переходы по страницам сайта (рисунок 38).

```
public String simpleMainPage() {
    return "/index";
}

public String simpleMainPageEntered() {
    return "/index_entered";
}
```

Рисунок 38 - Простые методы перехода Navigator

Они не содержат особой логики, что позволит избежать ненужных нагрузок. Необходимые данные уже буду хранится в других компонентах.

Разработка интерфейса.

После обеспечения доступа к данным, было важно добавить возможность взаимодействия с системой через интерфейс.

HTML-код и тэги EL (рисунок 39) позволяют сверстать страницу.

```
<?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.
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="jakarta.faces.html"
      xmlns:f="jakarta.faces.core"
      xmlns:c="jakarta.tags.core"
      >
    <h:head>
        <title>Welcome Page</title>
        <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"/>
    </h:head>
    <f:view>
    <h:body>
        <div class="container">
            <h:form>
            <header class="d-flex flex-wrap justify-content-center py-3 mb-4 border-bottom">
                <a href="#" class="d-flex align-items-center mb-3 mb-md-0 me-md-auto link-body-emphasis text-decorat
                    <span class="fs-4">Страница обратной связи</span> </a>
                <ul class="nav nav-pills">
                    <li class="nav-item"><a href="#" class="nav-link active" aria-current="page">Главная страница</a
                    <li class="nav-item"><h:commandLink class="nav-link" value="Регистрация" action="#{navigator.toS
                    <li class="nav-item"><h:commandLink class="nav-link" value="Вход" action="#{navigator.toSignIn()
                    <li class="nav-item"><a href="https://www.kvp24.ru/about/" class="nav-link">О компании</a></li>
                </ul>
```

Рисунок 39 - XHTML главной страницы

Все страницы являются файлами xhtml, они позволяют использовать expression language на представлении для передачи данных слою контроллера или получать данные от него. Также главная страница содержит ссылки на другие ресуры и страницы.

На странице размечены ранее описанные компоненты JSF. Они выводят список из базы данных (рисунок 40), используя технологии JPA, управляют навигацией, хранят данные итп.

```
<tbody>
    <c:forEach items="#{feedbackBean.feedbacks}" var ="j">
    <h:form>
        <tr>
            <td>#{j.fbId}</td>
            <td><h:outputText id="ffcat" value="#{j.nickname}"/></td>
            <td><h:outputText id="fmsg" value="#{j.message}"/></td>
            <td><h:outputText id="fmark" value="#{j.mark}"/></td>
        </tr>
    </h:form>
</c:forEach>
</tbody>
```

Рисунок 40 - Список отзывов на странице

В данном случае созданный экземпляр класса FeedbackService передаёт на страницу все записи в базе данных в форме таблицы.

Все блоки, использующие JSF, должны окружаться тегами <f:view> и <h:form>. XHTM более строг к вёрстке, поэтому каждый тег должен иметь начало <t> и конец </t>.

Конфигурация веб-контейнера проводится через файл web.xml (рисунок 41).

```
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Рисунок 41 - web.xml

В дескрипторе web.xml указана начальная страница – главная страница. Так приложение будет выдавать на представление главную страницу, как корневую.

Регистрация также представлена формой с полями для ввода данных (рисунок 42).

```
    </h:form>
    <div class="container align-items-center">
        <h:form>
            <p><h:inputText value="#{userBean.name}" required="true"/></p>
            <p><h:inputSecret value="#{userBean.password}" required="true"/></p>
            <p><h:commandButton class="nav-item" value="Sign In" action="#{navigator.toOffice()}"/></p>
        </h:form>
    </div>
</h:body>
</f:view>
```

Рисунок 42 - Форма регистрации

Страница регистрации использует компонент-подложку UserBean, чтобы хранить информацию о пользователе. При нажатии кнопки «Sign In» компонент-подложка навигатора осуществит регистрацию пользователя, если он не был занесён в базу данных.

При входе в систему пользователь переходит в свой кабинет (рисунок 43).

```
<tbody>
    <tr>
        <td>?</td>
        <h:form>
            <td><h:outputText value="#{userBean.name}"/></td>
            <td><h:inputText maxlength="300" value="#{feedbackBean.message}" required="true"/></td>
            <td><h:inputText type="number" id="ffdate" value="#{feedbackBean.mark}" required="true"/></td>
            <td><h:outputText id="fmark" value="NEW"/></td>
            <td><h:commandButton value="Добавить" action="#{feedbackBean.createFeedbacks(userBean.name)}"/>
        </h:form>
    </tr>
    <c:forEach items="#{feedbackBean.getUserFeedbacks(userBean.name)}" var ="j">
    <h:form>
        <tr>
            <td>#{j.fbId}</td>
            <td><h:outputText value="#{j.nickname}"/></td>
            <td><h:inputText value="#{j.message}"/></td>
            <td><h:inputText type="number" value="#{j.mark}"/></td>
            <td><h:outputText value="#{j.status}"/></td>
            <td>
                <h:commandButton value="Обновить" action="#{feedbackBean.updateFeedbacks(j)}"/>
                <h:commandButton value="Удалить" action="#{feedbackBean.deleteFeedbacks(j)}"/>
                <!-- <h:commandButton value="Одобрить" action="#{feedbackBean.approveFeedback(j)}"/> -->
            </td>
        </tr>
```

Рисунок 43 - Фрагмент страницы пользователя

Форма добавления нового отзыва соединена с общим списком записей пользователя, что улучшает читаемость и интуитивность интерфейса, без добавления отдельной страницы. Заполненные данные записываются в компоненты-контроллеры FeedBackbean и FeedBackService, служащие мостом между представлением и моделью.

Общая структура веб-страниц имеет вид (рисунок 44):

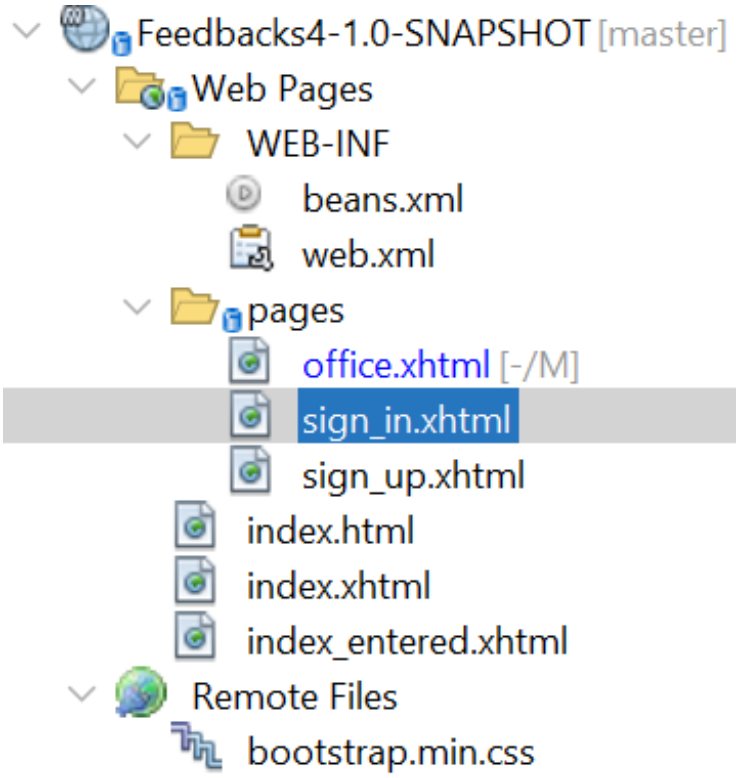


Рисунок 44 - Структура страниц

Стоит обратить внимание, что для вёрстки может использоваться удалённая библиотека bootstrap. Она влияет лишь на внешний вид страниц.

Добавление RESTful WebService.

Для упрощения управления и расширения функционала Jakarta EE позволяет настроить веб-сервисы REST. В отличие от веб-страниц, они могут обрабатывать целый спектр запросов, создавая своеобразный API [18].

На рисунке 45 показан заголовок класса `JakartaRestConfiguration`, предназначенного для настройки базового пути к REST-ресурсам с

использованием аннотации `@ApplicationPath` в рамках спецификации Jakarta RESTful Web Services.

```
*/
@ApplicationPath("resources")
public class JakartaRestConfiguration extends Application {

}
```

Рисунок 45 - Заголовок конфигуратора ресурсов

Приложения REST API находиться по пути /*корень*/resources/. Сами сервисы будут находится в модуле «resources». Они представляют из себя мини-приложения, реализующие доступ к некоторым ресурсам как API сервис.

На рисунке 46 представлен заголовок REST-ресурса `JakartaEE11Resource`, содержащего метод `ping()`, обрабатывающий GET-запросы по пути `/other` и возвращающий простой текстовый отклик."

```
@Path("other")
public class JakartaEE11Resource {

    @GET
    public Response ping(){
        return Response
                .ok("ping Jakarta EE")
                .build();
    }
}
```

Рисунок 46 - Заголовок ресурса

– @Path - указывает «корень» мини-приложения. Все запросы к этому сервису через браузер будут начинаться с «/*корнень*/resources/other»;

– @Get - указывает, что метод обрабатывает запрос типа «GET». В данном случает он просто генерирует страницу-ответ с текстом.

На рисунке 47 представлен метод findById, реализующий REST API для получения данных отзыва по его идентификатору. Метод обрабатывает GET-запрос и возвращает текст сообщения, соответствующего указанному id, в формате HTTP-ответа

```
@GET
@Path("/{id}")
public Response findById(@PathParam("id") Long fid){
    FeedbackBean fb_any = new FeedbackBean();

    List<Feedbacks> fb_list = fb_bean.getFeedbacks();
    List<String> fb_list2;


    fb_list2 = fb_list.stream().filter(fb->Objects.equals(fb.getFbId(), fid)).map(fb->fb.getMessage()).toList();


    return Response
            .ok().entity(fb_list2.getFirst())
            .build();
}
```

Рисунок 47 - API получения данных отзыва

Метод findById() обрабатывает GET запрос вида «…/other/[номер отзыва]» и отправляет ответ со строкой текста запрашиваемого отзыва.

На рисунке 48 представлен метод `create`, реализующий POST-запрос REST API для создания нового отзыва. Метод принимает объект `Feedbacks`, извлекает из него данные и передаёт их в сервисный метод `createfeedback` для последующего сохранения. «

```
@POST
@Transactional
public Response create(Feedbacks fb){
    //em.persist(fb);
    dc.createfeedback(fb.getMail(),fb.getMessage(),fb.getMark());

    return Response.ok()
            .entity(fb)
            .build();
}
```

Рисунок 48 - API создания отзыва

Метод POST с одним топ параметром @Transactional ожидает объект класса Feedbacks и отправляет его в базу данных уже упомянутым классом-сервисом FeedbackService.

Веб-сервис вместе с запросом получает JSON, который при получении и наличии соответствующего метода преобразуется в экземпляр java – описанный ранее класс Feedbacks для дальнейшего использования.

На рисунке 49 представлен метод `delete`, реализующий REST API для удаления отзыва по его идентификатору. Метод обрабатывает DELETE-запрос, вызывает соответствующий сервисный метод и возвращает подтверждение об удалении.

```
@DELETE
@Path("/{id}")
@Transactional
public Response delete(@PathParam("id") Long id) {
    fb_service.deletefeedback(id);

    return Response.ok("Deleted id " + id).build();
}
```

Рисунок 49 - API удаления отзыва

Запросы также бывают типа DELETE. Это условный тип, позволяющий логически отделить его от аналогичного GET. Метод отправляет сообщение об успешном удалении. Класс DataCollector удаляет запись по id, если она существует.

Ограниченность ресурсов не позволяет доработать REST-сервисы в полной мере, но общая работоспособность была реализована. Полный листинг некоторых классов с методами есть в приложении к работе.

2.5 Тестирование

Далее было необходимо провести тестирование разработанного веб-приложения и его развёртку на сервере WildFly [7]. Для начала были просмотрены все рабочие страницы приложения:

– главная страница;

– вход;

– регистрация;

– кабинет.

На рисунке 50 представлен внешний вид главной страницы веб-приложения, отображающей таблицу с отзывами пользователей и навигационное меню.

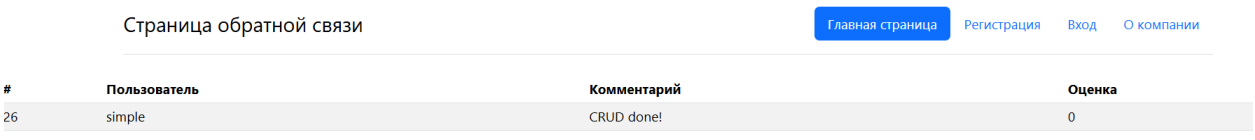


Рисунок 50 - Вид с главной страницы

На главной странице можно увидеть ссылки на другие страницы и список отзывов, полученный от session bean.

На рисунке 51 представлена страница входа в систему, содержащая поля для ввода имени пользователя и кнопки авторизации, а также навигационное меню.

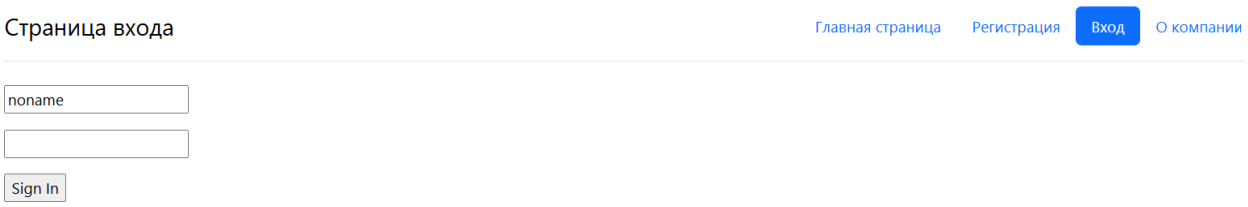


Рисунок 51 - Вид входа в систему

Вход осуществляется вводом имени и пароля уже существующего в базе пользователя.

На рисунке 52 показан интерфейс страницы регистрации, содержащий поля для ввода логина и пароля, а также кнопку подтверждения регистрации и стандартное навигационное меню.

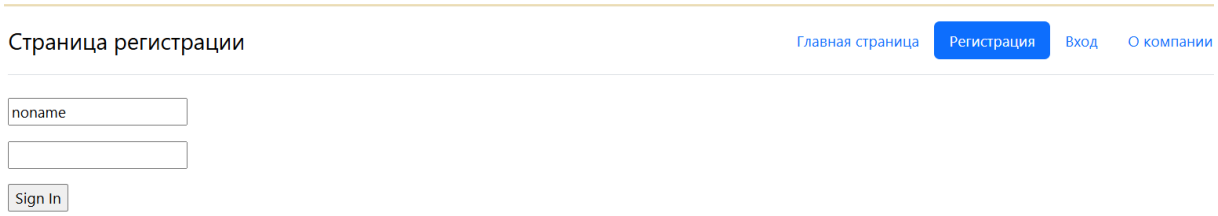


Рисунок 52 - Вид страницы регистрации

Регистрация имеет аналогичный входу интерфейс.

На рисунке 53 отображён интерфейс личного кабинета пользователя, где представлены оставленные отзывы, их статус, а также доступные действия: обновление, удаление и одобрение записи.

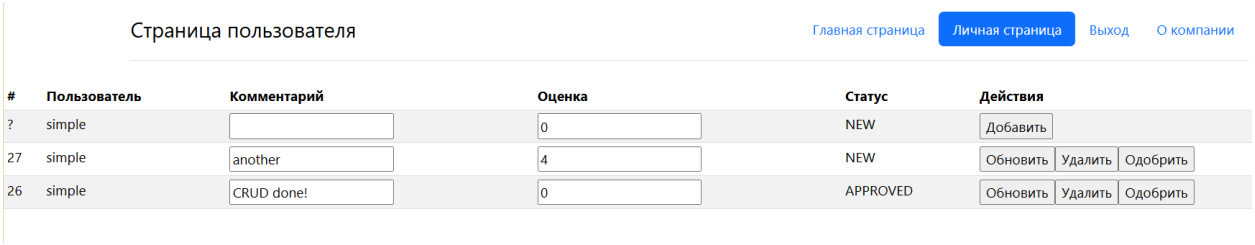


Рисунок 53 - Вид кабинета пользователя

Страница пользователя содержит не только опубликованные записи, но и ожидающие одобрения записи. В рамках тестирования на интерфейс добавлена кнопка публикации. На рисунке 54 представлена альтернативная версия главной страницы, отображающая как опубликованные, так и ожидающие одобрения записи пользователя.

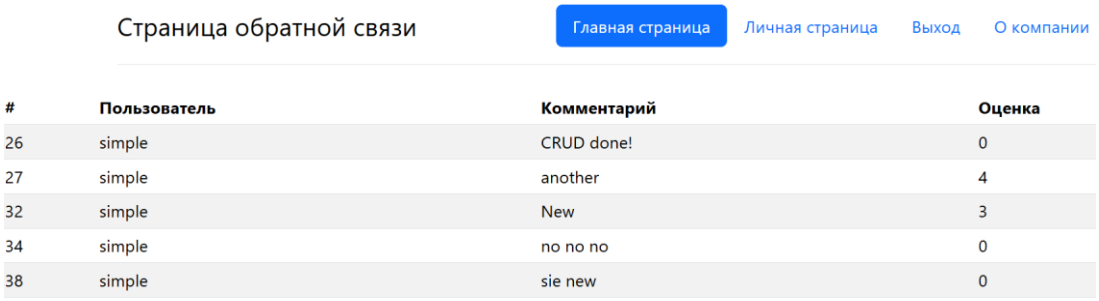


Рисунок 54 - Альтернативная главная страница

Для идентифицированного пользователя есть отдельная главная страница. Основное отличие в панели навигации.

Функциональное тестирование.

Функциональное тестирование должно проверить основные функции приложения, которые от него ожидает заказчик.

В таблице 1 представлены основные функциональные требования к странице с отзывами клиентов, которые подлежат проверке в рамках функционального тестирования.

Таблица 1 - Функциональные требования

Объект – Страничка с отзывами клиентов.	
№	Функциональные требования
1	Создание отзыва
2	Просмотр отзыва
3	Редактирование отзыва
4	Удаление отзыва

Тестовые сценарии описаны в таблице 2, так же в таблице отражены шаги выполнения сценария, вместе с ожидаемыми и полученными результатами.

Таблица 2 - План тестирования

Тестовый сценарий (test scenario)	Действия (actions)	Ожидаемый результат (expected results)	Полученный результат (actual results)
Просмотр	1. Войти на главную страницу	Комментарии видны	Новые комментарии видны и корректно отображаются
Создание	1. Войти на главную страницу 2. Зарегистрироваться 3. Заполнить форму нового отзыва	Комментарий будет виден на странице	Новые комментарии видны и корректно отображаются
Редактирование	1. Войти на главную страницу 2. Зарегистрироваться 3. Изменить данные отзыва в таблице	Комментарий изменяется	Обновлённые комментарии видны и корректно отображаются
Удаление	1. Войти на главную страницу 2. Зарегистрироваться 3. Нажать кнопку «удалить» в таблице отзывов	Комментарий удаляется	Новые комментарии видны и корректно отображаются

Тестирование показало, что приложение выполняет свои основные функции.

Нагрузочное тестирование.

Для нагрузочного тестирования было решено использовать инструмент DAST-тестирования - программу OWASP ZAP. Она проведёт множество запросов серверу для проверки стабильности системы.

На рисунке 55 показан процесс активного сканирования веб-приложения с использованием инструмента ZAP версии 2.16.1, предназначенного для выявления уязвимостей безопасности.

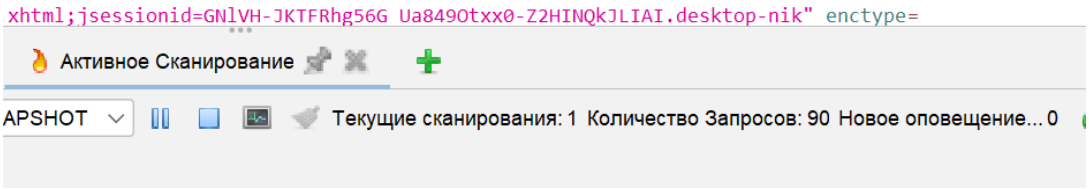


Рисунок 55 - Тестирование приложения в ZAP 2.16.1

По итогу были замечены проблемы с производительностью, но программа сохраняет работоспособность.

Тестирование на проникновение. SQL-инъекция.

Для тестирования нашего веб-приложения на уязвимости использовался специальный инструмент – SQLmap. Он проверил формы веб-сайта на уязвимость к SQL-инъекциям

Результаты тестирования (рисунок 56):

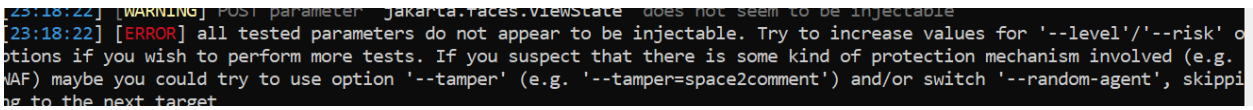


Рисунок 56 - Результат работы SQLMap

SQLMap не нашёл путей инъекции, но стоит протестировать сценарий, при котором злоумышленник знает имя пользователя, но не пароль.

Классическая SQL-инъекция рассчитана на отсутствие экранирования при передаче SQL-команд через форму на сервер базы данных.

При входе в систему, в поле пароля введена часть sql команды: q’ OR ‘1’=’1.

Выражение в программе могло приобрести вид:

SELECT name FROM user WHERE name='simple' and password='vereniki' or ‘1’=’1’;

Инъекция прибавила в запрос SELECT условие ‘1’=’1’ через логический оператор «or», что делает его истинным пр любом исходе, а значит система впустит злоумышленника в аккаунт без указание корректного пароля.

Результат (рисунок 57):

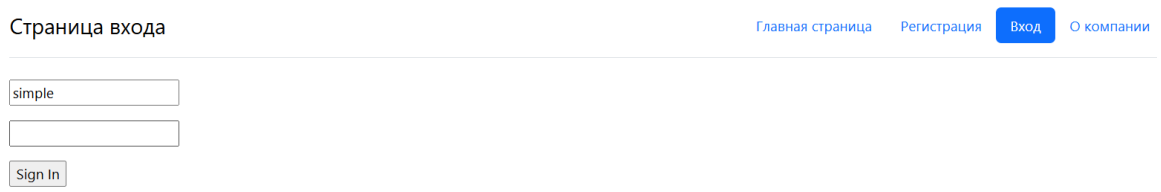


Рисунок 57 - Страница входа после SQL-инъекции

Система не пропустила условного злоумышленника, что означает защищённость формы от SQL-инъекций.

Работа с базой данных через JPA обеспечивает экранирование значений переменных при отправке SQL запросов, не позволяя внедрить в них вредоносный код.

Вывод по главе 2.

Во второй главе был представлен процесс разработки веб-приложения, основная цель которого — обеспечить удобный и надёжный механизм для сбора, хранения и обработки клиентских отзывов. На основе сформированных в предыдущей главе требований было реализовано приложение, которое позволяет пользователям оставлять комментарии, присваивать им оценку, а

также взаимодействовать с уже созданными записями. Таким образом, были реализованы ключевые возможности: создание, просмотр, обновление и удаление отзывов, а также функция модерации, позволяющая контролировать качество и релевантность публикуемого контента.

Разработка велась с соблюдением современных принципов многослойной архитектуры, включая использование шаблона MVC и разбиение системы на контроллеры, сервисы и модели. Особое внимание было уделено качеству кода и его сопровождаемости, что выразилось в логичной структуре пакетов и использовании аннотаций Jakarta EE для описания компонентов. Благодаря этому архитектура приложения получилась гибкой и расширяемой, что упрощает её сопровождение и дальнейшее развитие.

Кроме того, в рамках главы была проведена всесторонняя проверка работоспособности приложения в условиях, приближённых к реальной эксплуатации. Функциональное тестирование проводилось на основе заранее подготовленных сценариев и охватывало все основные пользовательские сценарии. Нагрузочное тестирование показало устойчивость приложения при обработке большого количества запросов, что подтверждает его пригодность для использования в условиях реальной нагрузки.

Особое внимание было уделено безопасности. Проведён анализ уязвимостей с использованием современных инструментов динамического тестирования безопасности (DAST), таких как OWASP ZAP и SQLMap. В результате были проверены все уязвимые точки взаимодействия пользователя с сервером.

Таким образом, поставленные во второй главе задачи были полностью решены. Разработанное программное обеспечение не только соответствует ранее сформулированным требованиям, но и обладает необходимыми качествами для реального применения. Полученные результаты подтверждают эффективность выбранных технических решений и создают прочную основу для дальнейшей интеграции приложения в инфраструктуру компании или его расширения с учётом новых потребностей.

Заключение

В ходе преддипломной практики были выполнены задачи, поставленные для достижения цели работы:

– Проведен анализ предметной области и требований заказчика, на основании которого составлено техническое задание;

– Выбран оптимальный стек технологий для реализации проекта;

– Спроектирована архитектура системы, с учетом требований масштабируемости и надежности;

– Реализован ключевой функционал приложения, включая интерфейс отправки отзывов, систему хранения и обработки данных;

– Проведено тестирование рабочего прототипа программного продукта в реальных условиях.

Разработанное решение полностью соответствует требованиям заказчика и обеспечивает:

– удобный канал получения обратной связи от пользователей;

– систематизированную публикацию поступающих отзывов;

– возможность дальнейшего анализа пользовательского опыта.

Перспективы развития проекта включают несколько ключевых направлений:

– совершенствование пользовательского интерфейса за счет разработки фирменного стиля, соответствующего корпоративным стандартам компании, и оптимизации UX/UI-решений для различных категорий пользователей;

– расширение функциональных возможностей системы, в частности внедрению гибкой системы ролевого доступа и разработке API для интеграции с другими сервисами компании;

– усовершенствование аналитического функционала, включая внедрение инструментов автоматической обработки отзывов с

использованием технологий NLP (Natural Language Processing) и разработку комплексной системы формирования аналитических отчетов.

Дополнительно рассматривается возможность реализации мобильной версии интерфейса и интеграции с популярными мессенджерами для повышения удобства подачи обратной связи.

Все планируемые улучшения будут реализованы с сохранением принципов модульности архитектуры, что обеспечит плавное масштабирование системы по мере роста бизнес-потребностей компании. Архитектурные решения проекта реализованы с учетом принципов модульности и масштабируемости, что позволяет легко расширять функционал системы в соответствии с развитием бизнес-процессов компании. Применение современных технологий разработки гарантирует стабильную работу решения при возрастающих нагрузках.

Реализованный проект имеет значительный потенциал для дальнейшего развития и может стать важным элементом системы клиентоориентированного сервиса компании «Квартплата 24». Внедрение решения позволит не только улучшить качество обратной связи с пользователями, но и получить ценные данные для совершенствования продуктовой линейки компании.

Список используемой литературы и используемых источников

1. Арлоу Д., Нейштадт И. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. - 2-е изд. - Санкт-Петербург: Символ-Плюс, 2007. - 624 с.
2. Бойцов А.К., Колмогорова С.С. Практическое моделирование на UML. - Санкт-Петербург: Реноме, 2023. - 210 с.
3. Вигерс К., Битти Д. Разработка требований к программному обеспечению. - 3-е изд. - Санкт-Петербург: БХВ-Петербург, 2014. - 576 с.
4. Городко С.И., Снисаренко С.В. Современные технологии программирования. - Минск: БГУИР, 2017. - 320 с.
5. Жилиндина О.В. Проектирование баз данных. - Благовещенск: АмГУ, 2023. - 41 с.
6. Ковалев С., Ковалев В. Настольная книга аналитика. Практическое руководство по проектированию бизнес-процессов и организационной структуры. - Москва: 1С-Паблишинг, 2020. - 384 с.
7. Куликов С.С. Тестирование программного обеспечения. Базовый курс. - 3-е изд. - Минск: Четыре четверти, 2020. - 432 с.
8. Лаврищева Е.М., Петрухин В.А. Методы и средства инженерии программного обеспечения. - Москва: МФТИ, 2007. - 256 с.
9. Макконел С. Совершенный код. Практическое руководство по разработке программного обеспечения. - Т. 2. - Санкт-Петербург: БХВ-Петербург, 2018. - С. 70-80.
10. Мардан А., Барклунд М. React быстро. - Санкт-Петербург: Питер, 2024. - 288 с.
11. Моисеев А.Н., Литовченко М.И. Основы языка UML. - Томск: Издательство Томского государственного университета, 2023. - 150 с.
12. Integrating with a PostgreSQL database. [Электронный ресурс]. - Режим доступа: https://www.wildfly.org/guides/database-integrating-with-postgresql (дата обращения: 25.11.24).

13. Install and Configure MySQL JDBC Driver on JBoss Wildfly | Craftsman Nadeem. [Электронный ресурс]. URL: https://reachmnadeem.wordpress.com/2021/05/13/install-and-configure-mysql-jdbc-driver-on-jboss-wildfly/ (дата обращения: 05.12.24).

14. JPA's EntityManager createQuery() vs createNamedQuery() vs createNativeQuery() [Электронный ресурс] URL: stackoverflow.com/questions/33798493/ (дата обращения: 10.12.24).

15. Jakarta EE 10 Official Documentation. [Электронный ресурс] URL: https://jakarta.ee/release/10/ (дата обращения: 08.12.24).

16. Java Persistence API (JPA) 2.2 Specification. [Электронный ресурс] URL: https://jakarta.ee/specifications/persistence/2.2/persistence_2.2.pdf (дата обращения: 05.02.25).

17. Java with Jakarta EE. [Электронный ресурс] URL: https://wiki.rakovets.by/java/jakarta-ee/ (дата обращения: 025.03.25).

18. Jennifer Niederst Robbins. Learning Web Design: A Beginner's Guide. 4th edition - O'Reilly Media - 2012

19. Mike K., Merrick S. Pro JPA. 1th edition - Apress Berkeley, CA – 2010

20. Schildt H. Java. The Complete Reference.12th ed. – New York: McGraw-Hill, 2021.

Приложение А

Класс сущности Feedbacks

```
@Entity
@Table(name = "feedbacks")
@NamedQueries({
    @NamedQuery(name = "Feedbacks.findAll", query = "SELECT f FROM
Feedbacks f"),
    @NamedQuery(name = "Feedbacks.findByFbId", query = "SELECT f FROM
Feedbacks f WHERE f.fbId = :fbId"),
    @NamedQuery(name = "Feedbacks.findByNickname", query = "SELECT f
FROM Feedbacks f WHERE f.nickname = :nickname"),
    @NamedQuery(name = "Feedbacks.findByMessage", query = "SELECT f
FROM Feedbacks f WHERE f.message = :message"),
    @NamedQuery(name = "Feedbacks.findByMark", query = "SELECT f FROM
Feedbacks f WHERE f.mark = :mark"),
    @NamedQuery(name = "Feedbacks.findByStatus", query = "SELECT f FROM
Feedbacks f WHERE f.status = :status"),
    @NamedQuery(name = "Feedbacks.findByPostDate", query = "SELECT f
FROM Feedbacks f WHERE f.postDate = :postDate")})
public class Feedbacks implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "fb_id")
    private Long fbId;
    @Size(max = 32)
    @Column(name = "nickname")
    private String nickname;
    @Size(max = 2147483647)
    @Column(name = "message")
    private String message;
    @Column(name = "mark")
    private Integer mark;
    @Size(max = 2147483647)
    @Column(name = "status")
    private String status = "NEW";
    @Basic(optional = false)
    @NotNull
    @Column(name = "post_date")
    @Temporal(TemporalType.DATE)
```

```
private Date postDate;

public Feedbacks() {
}

    public Feedbacks(Long fbId) {
        this.fbId = fbId;
    }

    public Feedbacks(Long fbId, Date postDate) {
        this.fbId = fbId;
        this.postDate = postDate;
    }

    public Long getFbId() {
        return fbId;
    }

    public void setFbId(Long fbId) {
        this.fbId = fbId;
    }

    public String getNickname() {
        return nickname;
    }

    public void setNickname(String nickname) {
        this.nickname = nickname;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public Integer getMark() {
        return mark;
    }

    public void setMark(Integer mark) {
```

```
this.mark = mark;
    }

public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public Date getPostDate() {
        return postDate;
    }

    public void setPostDate(Date postDate) {
        this.postDate = postDate;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (fbId != null ? fbId.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        if (!(object instanceof Feedbacks)) {
            return false;
        }
        Feedbacks other = (Feedbacks) object;
        if ((this.fbId == null && other.fbId != null) || (this.fbId != null &&
!this.fbId.equals(other.fbId))) {
            return false;
        }
        return true;
    }
    @Override
    public String toString() {
        return "com.mycompany.feedbacks4.entities.Feedbacks[ fbId=" + fbId + " ]";
 }}
```

Приложение Б

Класс сущности FeedbackService

```
@Stateless
public class FeedbackService {
   EntityManager em =
Persistence.createEntityManagerFactory("my_persistence_unit").createEntityMana
ger();

   /**
    * Creates a new instance of FeedbackService
    */
   public FeedbackService() {
   }

   public synchronized void createfeedback(String nick, String message, int mark)
{
      if (!isNewFeedBack(nick,message)) {
         return;
      }
      Feedbacks fb = new Feedbacks();
      fb.setNickname(nick);
      fb.setMessage(message);
      fb.setMark(mark);
      fb.setPostDate(new Date());

      EntityTransaction et = em.getTransaction();
      et.begin();
      em.persist(fb);
      et.commit();
   }
   public synchronized void updateFeedback(Feedbacks feedback) {
      Feedbacks fb = em.find(Feedbacks.class, feedback.getFbId());
      if (fb == null) {
         throw new NotFoundException("Feedback with id " + feedback + " not
found");
      }
      EntityTransaction et = em.getTransaction();
      et.begin();
      em.merge(feedback);
      et.commit();
   }
```

```
public synchronized void deletefeedback(Long id) {
        Feedbacks fb = em.find(Feedbacks.class, id);
        if (fb == null) {
            throw new NotFoundException("Feedback with id " + id + " not found");
        }
        EntityTransaction et = em.getTransaction();
        et.begin();
        em.remove(fb);
        et.commit();

    }

    public List<Feedbacks> getDataList() {
        //StringBuilder fbstr = new StringBuilder();
        List<Feedbacks> fb_all =
em.createNamedQuery("Feedbacks.findAll",Feedbacks.class).getResultList();

        return fb_all.stream().filter(f->f.getStatus() != null &&
f.getStatus().equals("APPROVED")).toList();
        //return fb_all;
    }

    public List<Feedbacks> getUserFeedbacksList(String nickname) {
        //StringBuilder fbstr = new StringBuilder();
        List<Feedbacks> fb_all =
em.createNamedQuery("Feedbacks.findByNickname",Feedbacks.class).setParamet
er("nickname", nickname).getResultList();
        return fb_all.stream().filter(f->f.getStatus() != null &&
!f.getStatus().equals("DELETED")).toList();
            }
    public boolean isNewFeedBack(String check_name, String check_msg) {
        List<Feedbacks> fbl =
em.createNamedQuery("Feedbacks.findAll",Feedbacks.class).getResultList();

        for (Feedbacks f:fbl) {
            if (check_name.equalsIgnoreCase(f.getNickname()) &&
check_msg.equals(f.getMessage())) return false;
        }

        return true;
    }
}
```

Приложение В

Класс сущности FeedbackBean

```
@Named(value = "feedbackBean")
@RequestScoped
public class FeedbackBean {
    @Inject
    FeedbackService fb_serv;

    String name,message;
    int mark;
    String status;
    /**
     * Creates a new instance of FeedbackBean
     */
    public FeedbackBean() {
    }

    public void createFeedbacks(String owner) {
        //mark = 0;
        fb_serv.createfeedback(owner, message, mark);
        message = "";
        mark = 0;
    }

    public List<Feedbacks> getFeedbacks() {
        return fb_serv.getDataList();
    }

    public void updateFeedback(Feedbacks new_fb) {
        //mark = 0;
        new_fb.setStatus("NEW");
        fb_serv.updateFeedback(new_fb);
    }

    public void deleteFeedbacks(Feedbacks new_fb) {
        //mark = 0;
        new_fb.setStatus("DELETED");
        //fb_serv.deletefeedback(id);
        fb_serv.updateFeedback(new_fb);
    }

    public List<Feedbacks> getUserFeedbacks(String nickname) {
```

```
return fb_serv.getUserFeedbacksList(nickname);
    }

    public void approveFeedback(Feedbacks new_fb) {
        new_fb.setStatus("APPROVED");
        fb_serv.updateFeedback(new_fb);
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public int getMark() {
        return mark;
    }

    public void setMark(int mark) {
        this.mark = mark;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }
}
```