Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет инженерно-экономический
Кафедра экономической информатики
Дисциплина «Программирование сетевых приложений»

«К ЗАЩИТЕ ДОПУСТИТЬ»
Руководитель курсового проекта
старший преподаватель
_____ Д.А. Сторожев
___.____.2023

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему:
«WEB-ПРИЛОЖЕНИЯ ДЛЯ РЕСТОРАННОГО БИЗНЕСА С ВОЗМОЖНОСТЬЮ ФОРМИРОВАНИЯ ОНЛАЙН ЗАКАЗОВ И АНАЛИТИКИ О РАБОТЕ ОРГАНИЗАЦИИ ОБЩЕПИТА»

БГУИР КП 1-40 05 01-10 009 ПЗ

Выполнил студент группы 084371
Карпеко Владислав Михайлович

_____
(подпись студента)

Курсовой проект представлен на проверку ___. ____.2023

_____
(подпись студента)

Минск 2023

РЕФЕРАТ

БГУИР КП 1-40 05 01-10 009 ПЗ

Карпеко, В.М. Web-приложения для ресторанного бизнеса с возможностью формирования онлайн заказов и аналитики о работе организации общепита / В. М. Карпеко. – Минск : БГУИР, 2023. − 62 с.

Пояснительная записка 62 с., 31 рис., 8 источников, 3 приложения

WEB-ПРИЛОЖЕНИЯ ДЛЯ РЕСТОРАННОГО БИЗНЕСА С ВОЗМОЖНОСТЬЮ ФОРМИРОВАНИЯ ОНЛАЙН ЗАКАЗОВ И АНАЛИТИКИ О РАБОТЕ ОРГАНИЗАЦИИ ОБЩЕПИТА

Цель проектирования: разработка Web-приложения для ресторанного бизнеса с возможностью формирования онлайн заказов и аналитики о работе организации общепита.

Методология проведения работы: в процессе решения поставленных задач были использованы методы компьютерной обработки данных, был использован метод анализа рынка общепитов и также в ходе данного курсового проектирования был использован метод абстрагирования для анализа работы общепита, чтобы наиболее точно и корректно можно было выполнить условия и техническое задание проекта.

Результаты работы: выполнен анализ литературных источников, спроектировано и написано приложение, разработана графическая часть проекта, проведен собственный анализ результатов, анализирован рынок работ общепитов, изучены виды аналитики о работе общепита и способы ее подсчета, были разработаны различных видов диаграммы и схемы:

– UML;

– IDEF0;

– модели представления программного средства;

– блок-схемы алгоритмов.

Область применения результатов: Web-приложения для ресторанного бизнеса с возможностью формирования онлайн заказов и аналитики о работе организации общепита может применяться в ресторанных областях и сферах предприятий, способствует облегчения взаимодействия клиентов с бизнесом.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данного курсового проекта является разработка Web-приложения для ресторанного бизнеса с возможностью формирования онлайн заказов и аналитики о работе организации общепита.

Объектом курсового проекта являются ресторанный бизнес.

А в свою очередь предмет исследования – автоматизация заказов.

Задачи, которые нужно решить во время курсового проекта:

– проектирование системы;

– разработка системы;

– тестирование системы;

– написание пояснительной записки;

– проектирование схем и диаграмм.

Ресторанный бизнес в своем современном воплощении возник не так давно. Всего несколько десятилетий назад в стране вообще не было частных ресторанов. Перспективы развития этой отрасли поражали, а первопроходцы постигали всё на собственном опыте.

Сегодняшним владельцам заведений общепита не нужно «изобретать велосипед». Схемы построения бизнеса уже готовы и апробированы, но рынок перенасыщен предложениями. Приходится сражаться за каждого клиента и место под солнцем. Открыть ресторан, кафе или бар получается у многих, но добиться от него прибыли, а иногда просто выжить на фоне тяжелой экономической ситуации и неожиданных форс-мажоров — у единиц. Все зависит от профессионализма.

Было время, когда мне настолько нравилось посещать рестораны, что я даже не брал еду на вынос. Теперь это в прошлом благодаря множеству сервисов, готовых предоставить полное ресторанное обслуживание на дому. Некоторые из них просто предлагают доставку из ресторанов разного уровня – от демократичных до элитных, а другие полностью переворачивают концепцию ресторана с ног на голову. Это позволяет и ресторанам, и шеф-поварам значительно увеличить количество клиентов.

Сервис Caviar, приобретённый компанией Square в 2014 году за 90 миллионов долларов, позволяет оформить доставку через сайт или приложение из тех ресторанов, где прежде доставка была недоступна. А такие кулинарные стартапы, как Sprig, Munchery и Spoonrocket, за считанные минуты доставляют полезные блюда от шеф-поваров прямо к дверям покупателей. Основанный в Сан-Франциско стартап Din в буквальном смысле воссоздаёт ваши любимые ресторанные блюда, а также может доставить ингредиенты на дом (вплоть до каждого яйца и кусочка масла), чтобы вы сами смогли их приготовить.

Более того, такие стартапы, как Kitchensurfing и Kitchit, устроят ресторан прямо у вас дома, предоставив шеф-повара, необходимые ингредиенты для приготовления блюда и средства для уборки на кухне. Оба сервиса подберут

для вас повара, который придёт к вам и приготовит ужин для любого количества гостей

Наше поколение живет в таком времени, в котором почти все люди владеют той или иной информацией. Поэтому роль информационных технологий огромна в жизни каждого из нас. И, порой невозможно представить нашу жизнь без этих технологий.

На сегодняшний день, с помощью всех современных устройств наша жизнь стала намного проще, а главное – удобнее! Ведь всего пару десятков лет назад человечество и мечтать не могло о том, что может позволить себе сейчас.

Информационные технологии – это процессы, которые используют совокупность средств и методов сбора, обработки и передачи данных (первичной информации) для получения информации нового качества о состоянии объекта, процесса или явления (информационного продукта).

Так же следует отметить появление программ, которые помогают банковским работникам, экономистам, бухгалтерам, проектировщикам в их тяжёлой, повседневной работе. Кроме того, появились детекторы лжи, способные выявлять ложь человека. Это очень помогает в расследовании серьезных преступных дел. Навигационные системы позволяют человеку ориентироваться в том месте, которое он не знает, и всегда проложить маршрут домой. А появление новейших медицинских аппаратов значительно подняло уровень медицины по всему миру - резко понизился показатель смертности, что является огромнейшим плюсом. С помощью них в современном мире человек может вылечиться практически от любой болезни.

Но, к сожалению, не все так идеально как хотелось бы…И у каждого новшества имеются свои недостатки. Так, человечество становится более зависимым от техники.

1 АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО СРЕДСТВА

1.1 Описание предметной области

В современной ресторанной индустрии появляются все новые и новые приложения и сервисы, направленные на улучшение качества обслуживания. Рестораны постоянно предлагают альтернативные решения для заказа, оплаты и бронирования… в общем, что угодно, кроме, разве что, аромата и вкуса еды онлайн. В последнее время многие из новых технологий произвели настоящий фурор на рынке – но как определить, какие из них останутся популярными, а какие со временем сойдут на нет?

Главные технологических тенденции ресторанной индустрии, которые позволяют улучшить качество обслуживания и увеличить число клиентов:

– бронирование;
– оплата заказа;
– ресторан на дому;
– «настольные» технологии;
– аавтоматизация.

Давайте разберем каждый отдельный элемент более детальнее.

Бронирование. Стремительный рост числа стартапов, меняющих привычную процедуру бронирования, вероятно, является самой спорной технологической тенденцией за последнее время.

И хотя позиции сервиса бронирования ресторанов OpenTable, приобретенного компанией Priceline за 2,6 млрд долларов, все ещё сильны, множество стартапов готовы составить ему конкуренцию. Такие сайты для оплаты бронирования, как Resy и Table8, заранее бронируют столики в самых элитных и «труднодоступных» ресторанах, а затем продают бронь пользователям. Цена за столик составляет около 20 долларов, но может меняться в зависимости от спроса (подобно модели ценообразования Uber).

Следующее приложение – Reserve – выполняет функции портье. Reserve не осуществляет бронь конкретных столиков в ресторанах, но с его помощью пользователи могут попасть в популярные рестораны в удобное для них время. Стоимость такой услуги зависит от времени и популярности заведения. Reserve бронирует для вас столик, передаёт ресторатору ваше имя и фотографию и оплачивает счёт, используя ранее введенные вами данные. В отличие от других платёжных приложений, с Reserve вам не придётся доставать свой телефон, чтобы оплатить заказ.

Ещё одним нововведением в сфере бронирования является покупка «билетов» в ресторан по аналогии с билетами в театр. Благодаря данной услуге клиенты смогут заранее оплатить все, что будет входить в их заказ. Технологии, несомненно, помогают воплотить в жизнь подобную идею, однако какого-либо сложного технического оснащения для этого не требуется, так как основная цель подобного проекта – избавить клиентов от проблем,

связанных связанных с цифровыми сервисами бронирования. Самая известная система онлайн-бронирования «билетов» в ресторан – Tock, которую представили Грант Акац (Grant Achatz) и Ник Коконас (Nick Kokonas), шеф-повара ресторанов Alinea и Next в Чикаго. Сервис уже запущен ими в собственных ресторанах, а в скором будущем и другие заведения смогут приобрести на него лицензию. Принцип не сильно отличается от покупки билета на концерт: вы выбираете дату и время, а затем оплачиваете заказ. Tock ограничивает возможность перепродать чужое бронирование и позволяет свести к минимуму ситуации, когда клиенты бронируют столик, а затем не приходят, в результате чего ресторан терпит убытки.

Несмотря на то, что некоторые технологии в сфере бронирования могут приносить операторам неплохую прибыль (речь идёт о сервисах, которые сотрудничают с ресторанами и платят им долю прибыли), простота онлайн-бронирования может привести к тому, что сторонние компании будут «навариваться» на популярности ресторанов. Такие сервисы, как Killer Rezzy и ныне прекративший существование ReservationHop, заранее бронируют места на популярные даты, а затем перепродают бронь с наценкой. В таком случае рестораны не имеют никакой выгоды, более того, они даже могут не знать, что цена на бронь была завышена. (Справедливости ради, стоит отметить, что на данный момент Killer Rezzy сотрудничает с определенными ресторанами по той же схеме, что Resy и Table8, но всё-таки пока предлагает бронирование и в другие заведения).

Сервисы стремятся привлечь к сотрудничеству рестораны и продолжают улучшать свои приложения, поэтому в будущем конкуренция станет еще более жёсткой. Я полагаю, что на смену «спекулятивному» бронированию придут новые, более выгодные для ресторанов сервисы и решения. В будущем ещё больше ресторанов введёт систему бронирования билетов, следуя примеру своих успешных предшественников – таких, как ресторан Alinea в Чикаго, в котором зародился и был впервые опробован сервис Tock. Ещё одним отличным примером является ресторан Lazy Bear в Сан-Франциско. Конечно, не стоит недооценивать амбиции и планы Гранта Акаца и Ника Коконаса касательно Tock – они объявили о своих намерениях обойти OpenTable, который по-прежнему является лидером в сфере онлайн-бронирования.

Оплата заказа. Даже такое, казалось бы, простое действие, как скачивание приложения, может вызвать затруднения – особенно, когда вы находитесь в тускло освещённом ресторане. Приложение Cover позволяет разделить чек с вашими друзьями, которые все ещё не установили это приложение (им всего лишь нужно будет скачать его в течение семи дней). Со временем эти приложения станут ещё проще и удобнее в использовании, так что абсолютно каждый сможет с лёгкостью расплачиваться телефоном.

Создатели мобильных приложений для оплаты станут больше внимания уделять разделению чеков. Так, например, TabbedOut уже сейчас даёт пользователям возможность разделить общую сумму на доли в долларах или

в процентах, а Cover позволяет разделить чек сразу между несколькими посетителями – правда пока ещё лишь на равные, а не произвольные части. В скором будущем многие из этих приложений смогут предложить более «гибкие» опции деления счета. Безусловно, количество людей, использующих эти технологии в погоне за новыми тенденциями, возрастёт. (А я не могу дождаться, когда смогу оплатить ужин Биткойнами).

Операторы по приёму платежей служат надёжным источником информации о потребителях. Например, с их помощью можно выяснить объёмы потраченных средств или предпочтения клиентов. Таким образом, процессинговые компании будут способствовать лучшему пониманию ресторанами поведения потребителей. Анализ информации о платежах и заказах клиентов позволит ресторанам отслеживать главные тенденции и поможет правильно составить меню и установить цены.

Ресторан на дому. На данный момент сервисы доставки еды ещё недостаточно развиты. Постепенно потребители освоят эти новые технологии, которые, в свою очередь, станут более удобными и простыми в использовании. Шеф-повар Дэвид Чанг, владеющий ресторанной сетью Momofuku, намерен расширить деятельность компании по доставке еды, инвестировав в стартап Maple. В отличие от других подобных сервисов, Maple будет доставлять своим клиентам еду собственного приготовления по рецептам, разработанным «Кулинарным советом директоров». Однако тот факт, что потребители постепенно привыкают питаться ресторанной едой за пределами ресторанов, позволяет этой новой отрасли сосредоточиться либо на самом блюде, либо на ресторанном.

«Настольные» технологии. Рестораны быстрого питания и так называемые рестораны «fastcas» (fast-casual) устанавливают на столах планшеты, которые позволяют клиентам не только оформить заказ, но и развлечься. Запуск приложения в сети ресторанов Taco Bell стал самым крупным рекламным и маркетинговым событием в истории компании, и это говорит о многом, учитывая её подкованность в вопросах цифровой связи. Появилась масса приложений для оформления и оплаты заказов с собой/на вынос. Доставка пиццы никогда не будет прежней – крупные национальные сети запускают приложения для оформления заказов. Международная сеть ресторанов Dominos установила голосовое сопровождение, так что теперь вы можете просто озвучить свой заказ приложению (вы даже можете сделать заказ через Twitter). В настоящее время всё больше ресторанов используют в своей работе сенсорные экраны, так, например, известные сети Applebees и Chilis предлагают посетителям воспользоваться планшетами для оформления заказов и оплаты чеков.

Технология взаимодействия с посетителями станет более распространенной. Благодаря экспериментам ресторанов с планшетами и смартфонами клиенты могут самостоятельно отправить свой заказ, что позволяет избежать ошибок персонала. Таким образом, повара готовят именно то, что хотели клиенты, а последние остаются довольны обслуживанием.

Сегодня каждый ребенок умеет пользоваться iPad, так что в эру цифровых технологий оформление заказов через сенсорный экран (в соответствующих заведениях) станет нормой.

Автоматизация. Системы автоматизации ресторана позволяют владельцам заведений решать целый ряд проблем: контролировать сотрудников, снижая вероятность обмана с их стороны, облегчать бухгалтерский учет и повышать скорость обслуживания клиентов.

В настоящий момент существует два типа систем автоматизации ресторана: автономные и облачные. Каждый из них имеет свои минусы — автономные системы не защищают от обмана владельца сотрудниками, а облачные зависят от наличия доступа к интернету и не дают большой гибкости в работе.

Есть и гибридный вариант системы автоматизации — так работает, к примеру Jowi. Локальный модуль устанавливается в ресторане, а затем данные синхронизируются и загружаются на удаленные серверы. Это позволяет сохранить работоспособность системы даже при «падении» интернета — когда связь восстановится, данные на сервере просто обновятся.

Система является модульной — есть части для менеджеров зала, официантов, поваров, бухгалтеров. Например, когда официант принимает заказ, система тут же выводит список блюд на экран повару, а когда блюдо будет готово (менеджер может включать обратный отсчет времени приготовления), официант получит уведомление на телефон или планшет. Jowi умеет в режиме реального времени измерять себестоимость порции в зависимости от изменения цен на продукты и, анализируя приходные накладные, рассчитывать коэффициент наценки и конечную цену блюда. Для удобства бухгалтеров любые действия в системе можно «провести» прошедшей датой — в реальной жизни такая необходимость возникает часто.

Системы автоматизации ресторанного бизнеса в будущем станут еще более функциональными, а число использующих их заведений очевидным образом будет расти. В них будет реализована и функциональность для решения связанных задач — например, бронирование столиков и сбор отзывов посетителей. Кроме того, появятся и решения по автоматизации и внедрение дополнительного контроля новых аспектов функционирования ресторана — например, скорости реагирования на нажатие кнопки вызова официанта [1].

1.2 Разработка функциональной модели предметной области

Проведённый анализ предметной области даёт возможность разработать функциональную модель процесса автоматизации работы ресторанного бизнеса на основе методологии IDEF0.

Достаточно часто собственники или руководители верхнего уровня ставят задачу разобраться с текущей ситуацией и выяснить целесообразность существующей на предприятии организационной структуры, четко определить функции подразделений, понять, кто за них отвечает, с какой

эффективность выполняется работа и т. п. Такая постановка задачи означает, что нужно подвергнуть анализу деятельность всего предприятия, в первую очередь руководителей крупных структурных подразделений. Модель в IDEF0 может помочь в этом случае четко расписать выполняемые функции, структурировать их по подразделениям. Построенную модель можно подвергнуть анализу и предложить изменения как в части структуры подразделений, так и по составу выполняемых ими функций. Модель может быть использована для обсуждения с собственниками и последующего принятия решений по реорганизации компании. При постановке указанной выше задачи никоим образом не говорится о процессном подходе или других технологиях управления. Задача сугубо конкретная. Способы ее решения должны быть максимально просты, и в то же время эффективны. При таком подходе можно строить модель в IDEF0, опираясь на схему организационной структуры предприятия.

Модели в IDEF0, построенные на основе организационной структуры предприятия, хорошо отражают его текущее состояние с точки зрения структуры и выполняемых функций. Однако процессы оказываются выделенными фактически только на уровне небольших отделов — там, где уже нет никаких подразделений, а существует разделение обязанностей между отдельными сотрудниками. Для анализа и реорганизации деятельности предприятия в целом такая степень детализации, как правило, оказывается излишней — чрезмерный объем информации затрудняет принятие решений. Межфункциональные (или «сквозные») бизнес-процессы предприятия при таком подходе оказываются невыделенными. Поэтому, если возникает необходимость анализа и реорганизации предприятия именно с точки зрения оптимизации крупных, межфункциональных бизнес-процессов, необходимо использовать другие подходы, например метод построения моделей в IDEF0 на основе цепочек создания ценности.

Для этого необходимо выявить цепочку последовательно выполняемых бизнес-процессов, на выходе которой появляются продукты (услуги) предприятия. В такую цепочку процессов, как правило, включаются не только процессы предприятия, но и процессы, выполняемые его внешними контрагентами [2].

Первая заключается в использовании данных стандартов и получаемых моделей для управления организацией. Первоначальное использование IDEF, как и введение понятий "бизнес-процесс", "стандарт моделирования деятельности компании", "методология описания" и т. д., в российской компании чаще всего связано с внедрением на предприятии информационной системы. Можно даже сказать, что информационные технологии сейчас в принципе выступают мощным "локомотивом" изменений, который приводит в движение все остальные части компании. Почему именно информационные технологии? Во-первых, потому что с изменением бизнес-среды перед предприятием встают не только новые оперативные вопросы, но и появляются новые стратегические задачи развития, решение которых требует новую

информацию, причем качественно новую, отражающую не только состояние, но и само строение бизнес-системы. Во-вторых, в информационных системах отражаются самые последние технические достижения, а также опыт и знания в предметных областях менеджмента. В-третьих, информационная система объединяет все подразделения компании, позволяя автоматизировать многие функции по сбору и обработке информации. Но вместе с тем, цель автоматизации основных процессов текущей операционной деятельности, которая ставится перед внедрением ИС, и предопределяет результаты аналитиков и разработчиков, сводящиеся к описанию бизнес-процессов на уровне конечных исполнителей, практически без учета деятельности руководителей. Обычно последующее за этим шагом решение непосредственно включить некоторые функции управления в информационную систему и необходимость в каждом процессе видеть место руководителя приходит позднее, когда создание моделей без учета деятельности руководителей входит в привычку. Таким образом, можно сделать первое умозаключение, что само по себе моделирование бизнес-процессов или наличие информационной системы ничего не значат с точки зрения поддержки процесса управления, пока руководителем явно не будет поставлена цель "включить меня в процесс".

Вторая проблема является следствием первой и связана с представлением разработанных моделей менеджерам компании. Дело в том, что полученные схемы и описания наиболее удобны разработчикам информационных систем и аналитикам, но не всегда выразительны и наглядны. Особенно если необходимо сделать презентацию сложного процесса с целью обсуждения возможных его улучшений и сравнить два варианта ("as is" и "to be"). То есть проблема состоит не в какой-либо методологической сложности IDEF и не в том, что менеджеры не способны ее изучить (или не хотят этого делать), а именно в выразительности схем при обсуждении и принятии решений большим количеством участников в сжатые сроки (в объеме презентации) [3].

После перечисленного выше была разработана четырехуровневая функциональная модель процессов предметной области рисунок 1.1.

На рисунке 1.2 представлена контекстная диаграмма верхнего уровня.

Рисунок 1.1 – Контекстная диаграмма функциональной модели

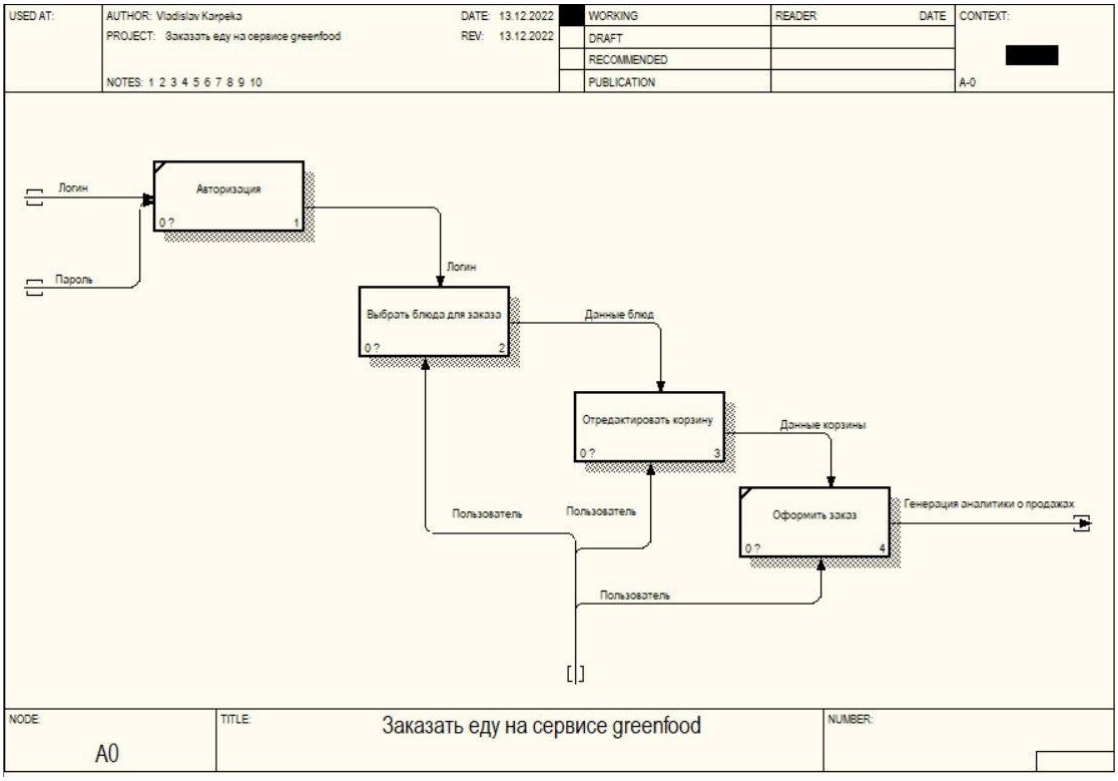

Рисунок 1.2 – Декомпозиция контекстной диаграммы

После декомпозиции контекстной диаграммы получаем следующие блоки:

- логин;

- выбор блюда для заказа;
- редактирование корзины;
- оформить заказ.

1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований

Ключевая цель премирования заключается в стимулировании заинтересованности работников качественно и оперативно выполнять поставленные задачи, соблюдать трудовую дисциплину, выполнять планы.

Модель данных должна быть реализована на основе ORM-технологии (Hibernate/JPA). Бизнес-логика приложения (серверная часть) должна быть реализована на основе фреймворка Spring или EJB (по согласованию с руководителем). Клиентская часть может быть реализована на веб-технологиях: JSP/Servlets/JSF/React/Angular.

Рекомендуется задействовать функционал стороннего сервиса, представленного в виде публичного REST API.

Конкретные версии фреймворков и технологий, применяемых для реализации программного средства, должны быть актуальными на начало 2022 года.

После получения технического задания и анализа требований к программному продукту, были выбраны следующие технологии.

Серверная часть:

– java 17;

– maven 3;

– spring boot 3;

– postreSQL 14.5;

– hibernate 6.

Клиентская часть:

– typescript;

– reactJs.

Выбор перечисленных выше технологий лежит на собственном опыте и также опираясь на современные тенденции и выбор программистами этих технологий в качестве инструментов для достижения целей клиента. Немного о них.

Oracle представила Java 17, новейшую версию языка программирования и платформа для разработки в мире №1. Java 17 отличается новым уровнем производительности, стабильности и сопровождается множеством обновлений в области безопасности. Новый релиз также содержит 14 JEP (JDK Enhancement Proposal), которые ведут к дальнейшим улучшениям платформы и языка программирования Java и помогают повысить производительность труда разработчиков.

Java 17 представляет собой новейший релиз из категории LTS (long-term support), который выходит в рамках стандартного шестимесячного цикла

релизов Java и является результатом тесного сотрудничества инженеров Oracle и других участников сообщества Java на базе OpenJDK Community и Java Community Process (JCP). Предыдущая версия JDK 11 LTS была выпущена три года назад, и за это время система была улучшена за счет 70 JEP от членов сообщества.

Oracle JDK 17 и будущие релизы JDK попадают под действие полностью бесплатной лицензии, которая будет действовать вплоть до следующего года после выхода следующего релиза LTS. Oracle также продолжит выпускать Oracle OpenJDK под действием открытой лицензии General Public License (GPL), на тех же условиях, которые были утверждены в 2017 году.

Oracle сотрудничает с сообществом разработчиков Java и JCP в работе над графиком выхода релизов LTS, чтобы предложить организациям больше гибкости в переходе на новые версии Java LTS. Таким образом, Oracle предполагает выпустить новый релиз LTS Java 21 в сентябре 2023, меняя частоту выхода LTS с трех лет до двух лет.

Пользователи с подписками Oracle LTS и Java SE могут мигрировать на Java 17 в наиболее комфортном для них режиме. Oracle продолжит предоставлятьэтим заказчикам обновления безопасности, гарантировать производительность и присылать исправления для Java 17 вплоть до сентября 2029.

Функциональность системы сборки Maven шире, чем компилятора исходного кода. В процессе работы приложения Apache Maven вызывает компилятор и при этом автоматически управляет зависимостями и ресурсами, например:

– загружает подходящие версии пакетов;

– размещает изображения, аудио- и видеофайлы в нужных папках;

– подгружает сторонние библиотеки.

Автоматическая сборка приложения особенно важна на этапах разработки, отладки и тестирования — Maven помогает собрать код и ресурсы в исполняемое приложение без IDE (среды разработки). При этом система сборки отличается гибкостью:

– может использоваться в IDE — Eclipse, IntelliJ IDEA, NetBeans и других;

– не зависит от операционной системы;

– не требует установки — архив с программой можно распаковать в любой директории;

– все необходимые параметры имеют оптимальные настройки по умолчанию;

– упрощает организацию командной работы и документирование;

– запускает библиотеки для модульного тестирования;

– обеспечивает соблюдение стандартов;

– имеет огромное количество плагинов и расширений.

Также существуют две другие системы сборки Java-приложений — Ant и Gradle, однако Maven пользуется наибольшей популярностью и является стандартом индустрии [4].

Из-за громоздкой конфигурации зависимостей настройка Spring для корпоративных приложений превратилась в весьма утомительное и подверженное ошибкам занятие. Особенно это относится к приложениям, которые используют также несколько сторонних библиотек.

Каждый раз, создавая очередное корпоративное Java-приложение на основе Spring, вам необходимо повторять одни и те же рутинные шаги по его настройке:

– в зависимости от типа создаваемого приложения (Spring MVC, Spring JDBC, Spring ORM и т. д.) импортировать необходимые Spring-модули;

– импортировать библиотеку web-контейнеров (в случае web-приложений);

– импортировать необходимые сторонние библиотеки (например, Hibernate, Jackson), при этом вы должны искать версии, совместимые с указанной версией Spring;

– конфигурировать компоненты DAO, такие, как: источники данных, управление транзакциями;

– конфигурировать компоненты web-слоя, такие, как: диспетчер ресурсов, view resolver;

– определить класс, который загрузит все необходимые конфигурации.

Авторы Spring решили предоставить разработчикам некоторые утилиты, которые автоматизируют процедуру настройки и ускоряют процесс создания и развертывания Spring-приложений, под общим названием Spring Boot.

Spring Boot — это полезный проект, целью которого является упрощение создания приложений на основе Spring. Он позволяет наиболее простым способом создать web-приложение, требуя от разработчиков минимум усилий по его настройке и написанию кода.

Spring Boot обладает большим функционалом, но его наиболее значимыми особенностями являются: управление зависимостями, автоматическая конфигурация и встроенные контейнеры сервлетов.

Чтобы ускорить процесс управления зависимостями, Spring Boot неявно упаковывает необходимые сторонние зависимости для каждого типа приложения на основе Spring и предоставляет их разработчику посредством так называемых starter-пакетов (spring-boot-starter-web, spring-boot-starter-data-jpa и т. д.).

Starter-пакеты представляют собой набор удобных дескрипторов зависимостей, которые можно включить в свое приложение. Это позволит получить универсальное решение для всех, связанных со Spring технологий, избавляя программиста от лишнего поиска примеров кода и загрузки из них требуемых дескрипторов зависимостей (пример таких дескрипторов и стартовых пакетов будет показан ниже).

Второй превосходной возможностью Spring Boot является автоматическая конфигурация приложения.

После выбора подходящего starter-пакета Spring Boot попытается автоматически настроить Spring-приложение на основе добавленных вами jar-зависимостей.

Например, если вы добавите Spring-boot-starter-web, Spring Boot автоматически сконфигурирует такие зарегистрированные бины, как DispatcherServlet, ResourceHandlers, MessageSource.

Каждое Spring Boot web-приложение включает встроенный web-сервер. Посмотрите на список контейнеров сервлетов, которые поддерживаются «из коробки».

Разработчикам теперь не надо беспокоиться о настройке контейнера сервлетов и развертывании приложения на нем. Теперь приложение может запускаться само, как исполняемый jar-файл с использованием встроенного сервера.

Если вам нужно использовать отдельный HTTP-сервер, для этого достаточно исключить зависимости по умолчанию. Spring Boot предоставляет отдельные starter-пакеты для разных HTTP-серверов [5].

Создание автономных web-приложений со встроенными серверами не только удобно для разработки, но и является допустимым решением для приложений корпоративного уровня и становится все более полезно в мире микросервисов. Возможность быстро упаковать весь сервис (например, аутентификацию пользователя) в автономном и полностью развертываемом артефакте, который также предоставляет API — делает установку и развертывание приложения значительно проще.

Объектно-реляционная модель, или ORM, позволяет создать программную «виртуальную» базу данных из объектов. Объекты описываются на языках программирования с применением принципов ООП. Java Hibernate — популярное воплощение этой модели.

Hibernate построен на спецификации JPA 2.1 — наборе правил, который описывает взаимодействие программных объектов с записями в базах данных. JPA поясняет, как управлять сохранением данных из кода на Java в базу. Но сама по себе спецификация — только теоретические правила, а в «чистой» Java ее реализации нет. Hibernate — одна из самых популярных реализаций JPA на рынке.

Фреймворк бесплатный, с открытым исходным кодом, который может просмотреть любой программист. По-русски название читается как «хибернейт».

Hibernate пользуются Java-разработчики, которые работают с базами данных или с обработкой информации для последующего переноса в базу. Фреймворк используют при создании информационных систем: приложений, крупных программ и сетей, которые работают с информацией и базами данных. Существует аббревиатура CRUD, означающая Create, Read, Update, Delete: создавать, читать, обновлять и удалять. Это четыре действия, которые

должна уметь выполнять информационная система, работающая с базой. Задача Hibernate при создании такого приложения — сократить количество низкоуровневого кода и облегчить работу программиста с БД.

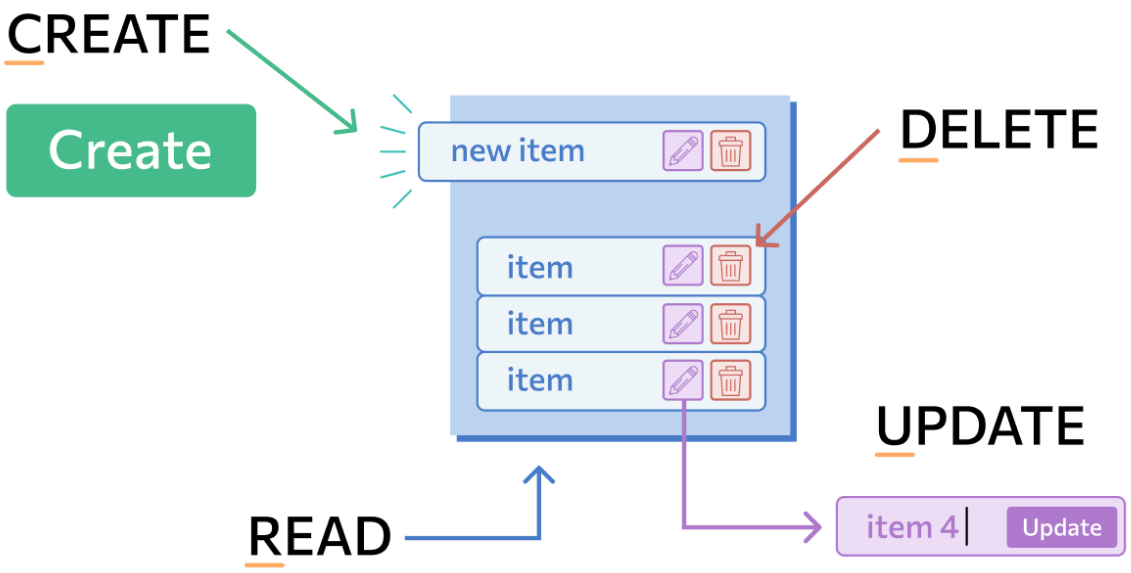

Рисунок 1.3 – CRUD операции.

Hibernate:

– ускоряет и облегчает написание кода;

– позволяет создать удобную модель для отображения базы данных в коде;

– дает возможность быстро и в читаемом виде записывать информацию из кода в базу.

В автоматизации бизнеса выделяют следующие требования:

– функциональные;

– нефункциональные.

Рассмотрим их в деталях.

При разработке программного обеспечения функциональные требования определяют функции, которые должно выполнять все приложение или только один из его компонентов. Функция состоит из трех шагов: ввод данных — поведение системы — вывод данных. Он может вычислять, манипулировать данными, выполнять бизнес-процессы, устанавливать взаимодействие с пользователем или выполнять любые другие задачи.

Другими словами, функциональное требование — это то, ЧТО приложение должно или не должно делать после ввода некоторых данных.

Функциональные требования важны, поскольку они показывают разработчикам программного обеспечения, как должна вести себя система. Если система не соответствует функциональным требованиям, значит, она не работает должным образом.

Нефункциональные требования определяют стандарты производительности и атрибуты качества программного обеспечения, например удобство использования системы, эффективность, безопасность, масштабируемость и т. д.

В то время как функциональные требования определяют, что делает система, нефункциональные требования описывают, КАК система это делает. Например, веб-приложение должно обрабатывать более 15 миллионов пользователей без какого-либо снижения производительности, или веб-сайт не должен загружаться более 3 секунд.

Если приложение не соответствует нефункциональным требованиям, оно продолжает выполнять свои основные функции, однако не сможет обеспечить удобство для пользователя.

Нефункциональные требования важны, поскольку они помогают разработчикам программного обеспечения определять возможности и ограничения системы, которые необходимы для разработки высококачественного программного обеспечения. Следовательно, нефункциональные требования так же важны, как и функциональные требования для успешного внедрения продукта.

Исходя из вышеперечисленного –давайте рассмотрим функциональные требования к Web-приложению для ресторанного бизнеса с возможностью формирования онлайн заказов и аналитики о работе организации общепита:

– авторизация пользователя;
– режим для пользователя;
– режим для администратора;
– просмотр блюд;
– добавление блюд в заказ;
– удаление блюд из заказа;
– оформление заказа на определенный адрес;
– просмотр комментариев пользователей;
– добавление комментариев;
– удаление комментариев;
– просмотр аналитики;
– добавление блюд в список блюд при наличии учетной записи администратора.

1.4 Разработка информационной модели предметной области

Информационная система – это коммуникационная и вычислительная система, которая предназначена для сбора, хранения, обработки и передачи информации. Она также снабжает работников различного ранга той информацией, которая необходима им для реализации функций управления [6].

Информационная модель системы отражает информацию о предметной области, данные о которой должны храниться в проектируемой базе данных.

В данном курсовом проекте предметной областью является автоматизация работы медицинских учреждений.

В процессе информационного моделирования были выделены следующие сущности:

- пользователи;
- заказы;
- отзывы;
- корзины.

Сущность «Пользователи» служит для предоставления возможности входа в систему. Наличие такой сущности позволяет разграничить пользователей системы по типу (администратор или простой пользователь) и тем самым предоставить каждому пользователю необходимый ему функционал. Атрибутами этой сущности являются логин, пароль, имя, роль, данные о статусе аккаунта и уникальный ID для каждого пользователя.

Пользователь, когда он находится в режиме администратор, имеет полный функционал:

- создание пользователя;
- просмотр пользователей;
- обновление пользователя;
- удаление пользователя;
- создание заказа;
- просмотр аналитики за год;
- обновление заказа;
- удаление заказа;
- добавление блюд;
- просмотр отзывов.

В свою очередь обычный пользователь может только просматривать данные и работать с заказом и отзывами своими.

Сущность «заказы» нужна для хранения информации о заказах, с которым работает пользователь.

Сущность «отзывы» нужна для хранения и отображения отзывов пользователей о сервисе.

Сущность «корзины» нужна для хранения информации о блюдах, который находятся у пользователя в корзине.

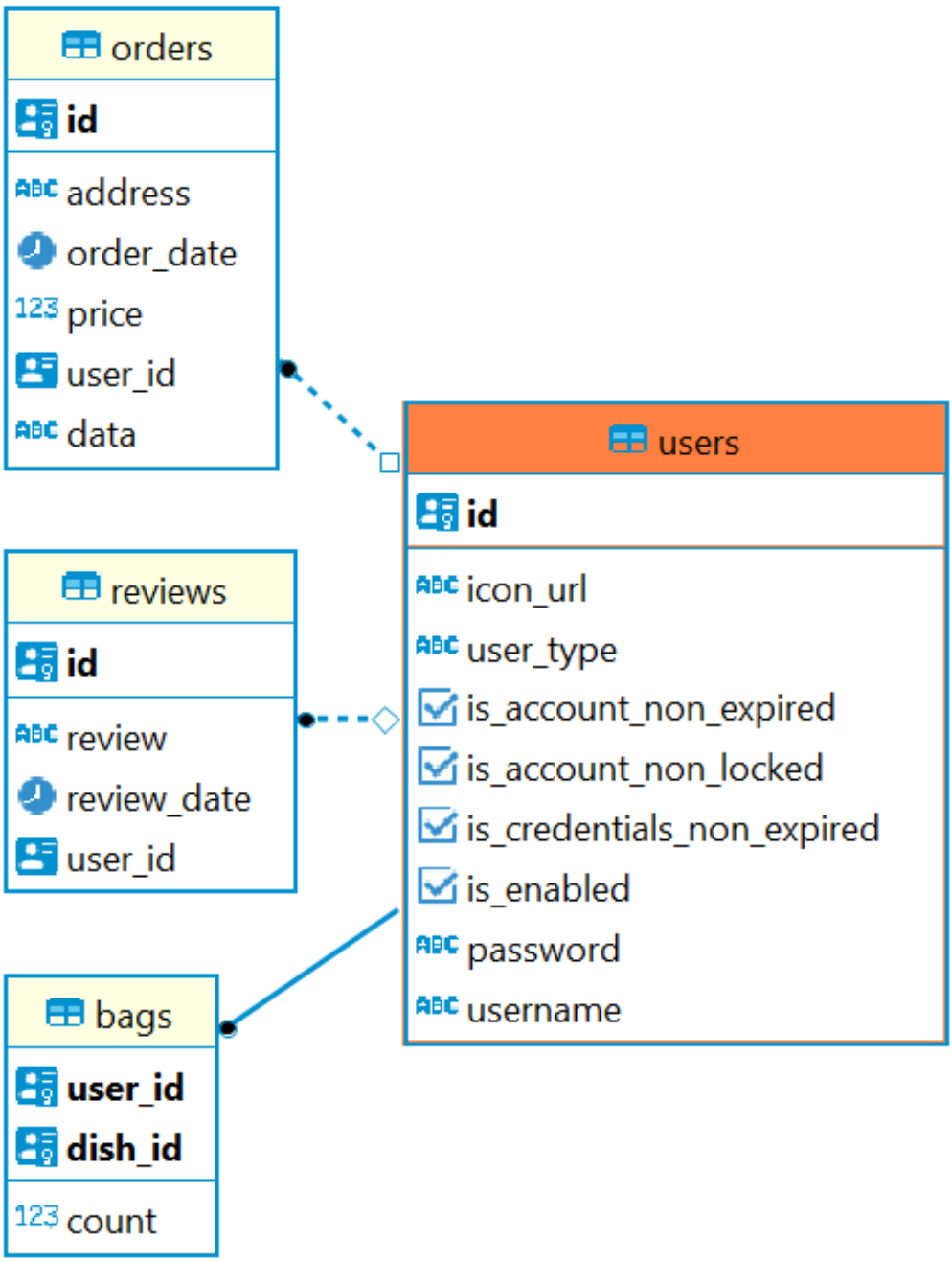

Рисунок 1.4 – Информационная модель базы данных

С учётом обозначенного взаимодействия сущностей смоделируем их взаимодействие в формате IDEF1.X и приведём эту модель к третьей нормальной форме. В результате последовательного приведения получается модель, соответствующая условиям третьей нормальной формы – не ключевой атрибут сущности функционально зависит только от всего первичного ключа и ни от чего другого (рисунок 1.4).

1.5 UML-модели представления программного средства и их описание

Модели UML могут создаваться и могут использоваться с различными целями. Иногда бывает даже так, что автор создавал модель с одной определенной целью, а используется она другими людьми совершенно неожиданным для автора образом. Таким образом, назначение модели не является чем-то постоянным и трудно изменяемым, как цвет кожи или разрез глаз. Трансформации назначения моделей вполне возможны, но, тем не менее, практика моделирования подсказывает, что модели дают больший эффект, если при моделировании принимать во внимание назначение модели.

Рисунок 1.5 – UML-модели представления программного средства и их описание

2 ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Постановка задачи

В данном курсовом проекте поставлена цель разработать Web-приложения для ресторанного бизнеса с возможностью формирования онлайн заказов и аналитики о работе организации общепита. Для этого необходимо разработать такое приложение, которое предоставит возможность многим пользователям вносить и просматривать необходимую информацию. Пользоваться им смогут различные клиенты данного приложения.

Предлагается разработать приложение в архитектуре клиент-сервер. Так как необходимость воспользоваться приложением может возникать сразу у нескольких пользователей, сервер должен поддерживать возможность одновременно обрабатывать запросы этих пользователей. Реализация серверного приложения будет осуществлена в виде spring boot приложения на основе REST HTTP общения с клиентом.

Для реализации данного проекта необходимо выполнить следующие задачи:

– изучить предметную область;

– реализовать клиент-серверное приложение;

– настроить базу данных PostgreSQL;

– связать базу данных с сервером;

– создать удобный интерфейс для пользователя;

– разработать собственную иерархию классов;

– реализовать не менее двух паттернов проектирования;

– использовать сокрытие данных, перегрузку методов, переопределение методов, сериализацию, абстрактные типы данных, статические методы;

– предусмотреть обработку исключительных ситуаций;

– предоставить пользователю аналитическую информацию в виде диаграмм;

– протестировать полученное приложение.

Для максимального удобства клиентское приложение должно быть реализовано в виде web-приложения с графическим интерфейсом пользователя. Функционал администратора отличается от функционала простого пользователя, поэтому это будет отражено в клиентском приложении. Администратору будут доступны все функции для работы со списками блюд. Простой пользователь сможет просматривать всю информацию, но не вносить изменения [6].

2.2 Архитектурные решения

Для программной реализации системы в данном курсовом проекте был выбран объектно-ориентированный язык программирования Java.

Как было описано выше для данного курсового проекта необходимо было реализовать два приложения одно для серверной части, другое для клиентской, для общения между серверами был выбран REST реализованный поверх HTTP. Структура приложения изображена на рисунке 2.1.

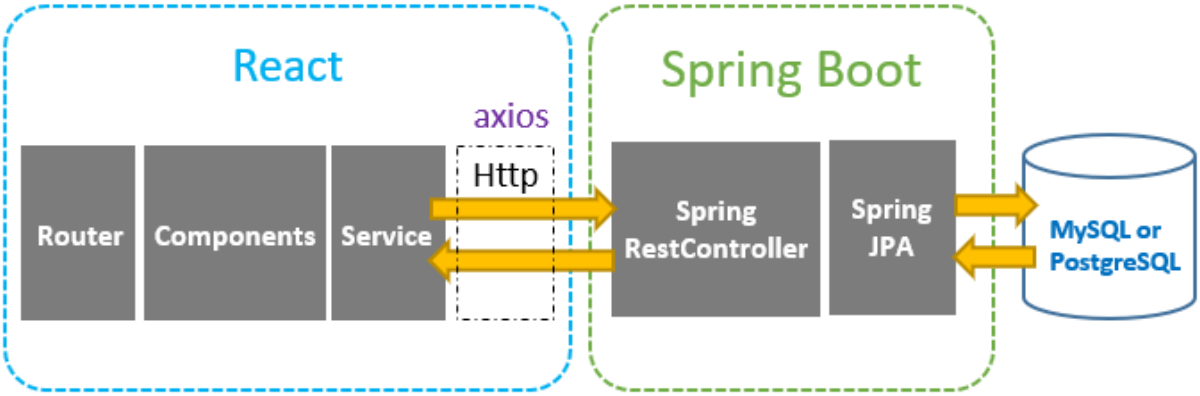

Рисунок 2.1 – структура приложения.

Representational State Transfer (REST) в переводе — это передача состояния представления. Технология позволяет получать и модифицировать данные и состояния удаленных приложений, передавая HTTP-вызовы через интернет или любую другую сеть.

Если проще, то REST API — это когда серверное приложение дает доступ к своим данным клиентскому приложению по определенному URL. Далее разберем подробнее, начиная с базовых понятий.

Application Programming Interface (API), или программный интерфейс приложения — это набор инструментов, который позволяет одним программам работать с другими. API предусматривает, что программы могут работать в том числе и на разных компьютерах. В этом случае требуется организовать интерфейс API так, чтобы ПО могло запрашивать функции друг друга через сеть.

Также API должно учитывать, что программы могут быть написаны на различных языках программирования и работать в разных операционных системах.

REST API позволяет использовать для общения между программами протокол HTTP (зашифрованная версия — HTTPS), с помощью которого мы получаем и отправляем большую часть информации в интернете.

HTTP довольно прост. Посмотрим на его работу на примере. Допустим, есть адрес http://website.com/something. Он состоит из двух частей: первая —

это адрес сайта или сервера, то есть http://website.com. Вторая — адрес ресурса на удаленном сервере, в данном примере — /something.

Вбивая в адресную строку URL-адрес http://website.com/something, мы на самом деле идем на сервер website.com и запрашиваем ресурс под названием /something. «Пойди вот туда, принеси мне вот то» — и есть HTTP-запрос. Пример изображён на рисунке 2.2.

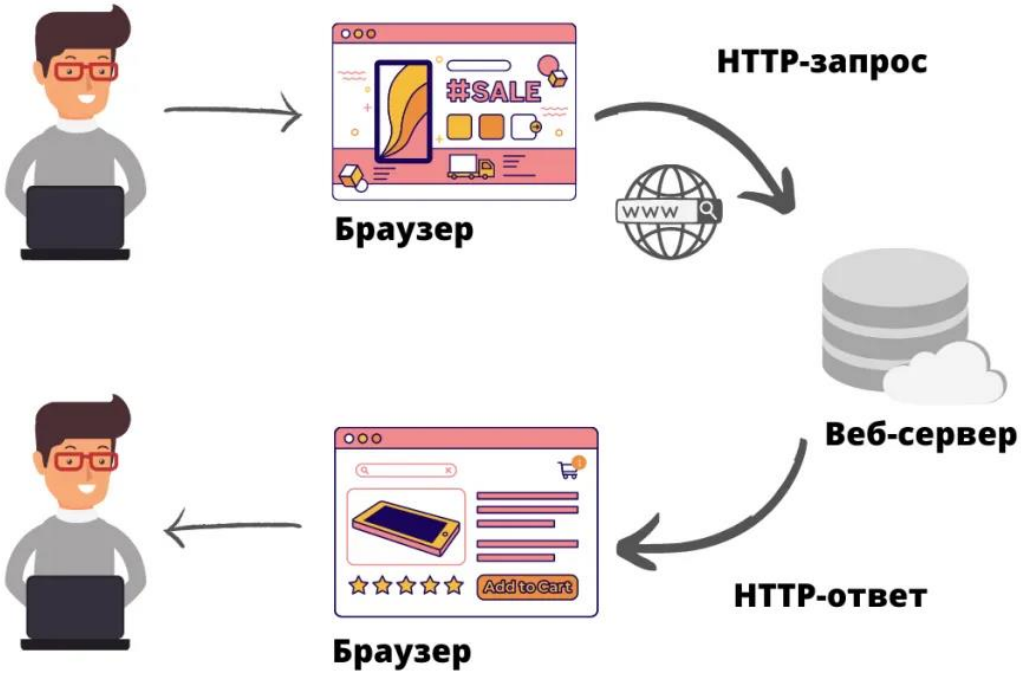

Рисунок 2.2 – пример HTTP-запроса к серверу

Чтобы ресурс, который вы запрашиваете, выполнял нужные действия, используют разные способы обращения к нему. Например, если вы работаете со счетами с помощью ресурса /invoices, который мы придумали выше, то можете их просматривать, редактировать или удалять.

В API-системе четыре классических метода:

– GET — метод чтения информации. GET-запросы всегда только возвращают данные с сервера, и никогда их не меняют и не удаляют. В бухгалтерском приложении GET /invoices вы открываете список всех счетов;

– POST — создание новых записей. В нашем приложении POST /invoices используется, когда вы создаете новый счет;

– PUT — редактирование записей. Например, PUT /invoices вы исправляете номер счета, сумму или корректируете реквизиты;

– DELETE — удаление записей. В нашем приложении DELETE /invoices удаляет старые счета, которые контрагенты уже оплатили.

Таким образом, мы получаем четыре функции, которые одна программа может использовать при обращении к данным ресурса, в примере — это ресурс для работы со счетами /invoices.

Архитектура REST API — самое популярное решение для организации взаимодействия между различными программами. Так произошло, поскольку HTTP-протокол реализован во всех языках программирования и всех операционных системах, в отличие от проприетарных протоколов.

Чаще всего REST API применяют:

– для связи мобильных приложений с серверными;

– для построения микросервисных серверных приложений. Это архитектурный подход, при котором большие приложения разбиваются на много маленьких частей;

– Для предоставления доступа к программам сторонних разработчиков. Например, Stripe API позволяет программистам встраивать обработку платежей в свои приложения [7].

Для данного курсового проекта было выбрано использовать микро сервисную архитектуру, а не монолит.

Одни из критериев было то, что это условие данного курсового проекта, второй момент – это то, что, на текущий момент микро сервисная архитектора развивается быстро и является очень популярным решением в web мире. Но также нужно упомянуть, что микросервисы – это не «серебряная пуля». Это решение нужно применять с умом и взвесив все за и против.

Немного о данном подходе.

Микросервисная архитектура — это подход, который помогает не только ускорить разработку продукта, но и сделать ее гибкой и управляемой: проект из неделимого целого превращается в систему связанных между собой блоков — сервисов. Впервые о микросервисах заговорили ещё в 2000-х, но концепция архитектуры сформировалась только к началу 2010-х. К 2014 году технологию внедрили такие крупные компании, как Netflix, Amazon и Twitter. Сегодня микросервисный подход используют гораздо активнее. В 2020 году в отчёте Cloud Microservices Market Research рынок облачных микросервисов оценили в 831,45 млн долларов США. К 2026 году его масштабы могут увеличиться более чем в три раза.

Приложение с микросервисной архитектурой разделено на небольшие не зависящие друг от друга компоненты — микросервисы. У каждого из них своя бизнес-задача: например, управлять каталогом, хранить и обновлять содержимое корзины или проводить оплату заказа.

Для наглядности покажем, как выглядит архитектура микросервисов в Yandex Cloud рисунок 2.3.

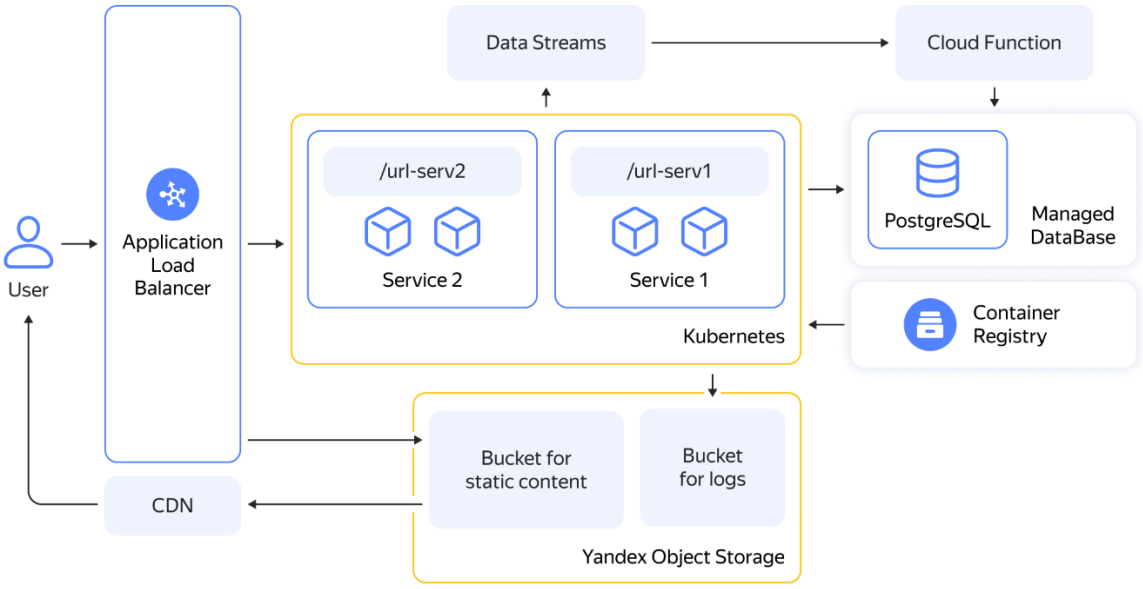

Рисунок 2.3 – архитектура микросервисов в Yandex Cloud.

Конечно, микросервисный подход не лишён недостатков. При кажущейся простоте и логичности деления большого продукта на самостоятельные сервисы разработка распределённой системы — процесс сложный и с технической, и с организационной точек зрения. Плюсы могут обернуться минусами:

– сбой одного сервиса не приведёт к полному отказу приложения, но любая распределённая система имеет и другие слабые места: потенциальные проблемы связи её элементов друг с другом, сетевые задержки, возможная неконсистентность данных;

– Вы сэкономите, если будете платить только за те ресурсы, которые потребляют микросервисы, но должны будете предусмотреть расходы на внедрение облачных технологий, отдельное развёртывание каждого нового сервиса и его покрытие отдельными тестами и мониторингами;

– Контролировать качество решения отдельных бизнес-задач проще и эффективнее, чем оценивать систему в целом, но настроить рабочие процессы большой команды разработчиков не так уж легко.

Также хотелось бы затронуть ахритектуру написания микросервиса на сервеной части.

В качестве паттерна для разработки был выбран MVC и был успешно реализован, что видно на рисунке 2.4.

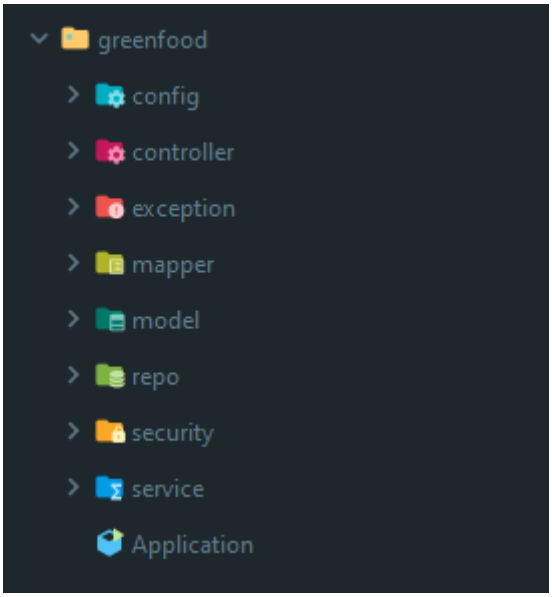

Рисунок 2.4 – структура серверной части

MVC расшифровывается как «модель-представление-контроллер» (от англ. model-view-controller). Это способ организации кода, который предполагает выделение блоков, отвечающих за решение разных задач. Один блок отвечает за данные приложения, другой отвечает за внешний вид, а третий контролирует работу приложения.

Компоненты MVC:

– Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента model будет определять список задач и отдельные задачи;

– Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования;

– Контроллер — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя. По сути, это мозг MVC-приложения.

Для удобства развертывания приложения был использован подход контейнеризации, в частности Docker и Docker-compose.

Docker — это платформа, которая позволяет упаковать в контейнер приложение со всем окружением и зависимостями, а затем доставить и запустить его в целевой системе.

Приложение, упакованное в контейнер, изолируется от операционной системы и других приложений. Поэтому разработчики могут не задумываться, в каком окружении будет работать их приложение, а инженеры по эксплуатации — единообразно запускать приложения и меньше заботиться о системных зависимостях.

Docker разработали в 2008 году. Изначально это был внутренний проект компании dotCloud, которую впоследствии переименовали в Docker Inc. В 2013 году dotCloud открыла исходный код Docker для сообщества.

Популярность Docker продолжает расти, потому что его поддерживает большое сообщество. Платформа попала в мейнстрим на волне популярности DevOps, быстрых конвейеров доставки и автоматизации.

Docker Compose — это средство для определения и запуска приложений Docker с несколькими контейнерами. При работе в Compose вы используете файл YAML для настройки служб приложения. Затем вы создаете и запускаете все службы из конфигурации путем выполнения одной команды.

Docker-compose для приложения изображён на рисунке 2.5.

34 lines (33 sloc) | 705 Bytes

```yaml
version: '3.8'
services:
  db:
    image: postgres:14.5-alpine
    container_name: db
    environment:
      - POSTGRES_DB=test
      - POSTGRES_USER=test
      - POSTGRES_PASSWORD=test
    ports:
      - '5432:5432'
  backend:
    container_name: backend
    build:
      context: ./backend
      dockerfile: Dockerfile
    environment:
      - SPRING_DATASOURCE_DRIVER_CLASS_NAME=org.postgresql.Driver
      - SPRING_DATASOURCE_URL=jdbc:postgresql://db:5432/test
    ports:
      - "8080:8080"
    depends_on:
      - db

  frontend:
    container_name: frontend
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    depends_on:
      - db
      - backend
```

Рисунок 2.5 – Docker-compose файл для данного проекта

2.3 Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства

Как было описано выше для серверной части, как паттерн реализации был выбран MVC, и следовательно основная часть реализации была предоставлена в сервисных классах.

В основном это простая реализация CRUD операции, но есть и интересные моменты, о которых поговорим дальше.

Для реализации создания нового элемента, который будет положен в корзину для заказа пользователя. Его логика следуюжая. Проверка на наличие текущего блюда в заказе, если оно есть, то увелечение счетчика этого заказа на один, если нет, то проверить на наличие нужного блюда и пользователя в базе и при успешных ответах создать новый этемент в корзине. Реализация на рисунке 2.6.

```
@Override
@Transactional
public void createBag(UUID userId, UUID dishId) {
  Bag bag = bagService.findByUserIdAndDishId(userId, dishId);

  if (bag != null) {
    bag.setCount(bag.getCount() + 1);
    repository.save(mapper.dtoToEntity(bag));
  } else {
    userService.getUserById(userId);
    dishService.getDishById(dishId);

    repository.save(new BagEntity(userId, dishId, 1));
  }
}
```

Рисунок 2.6 – добавление в корзину нового элемента.

Для авторизации был написан метод, который изображен на рисунке 2.7.

```java
@Override
public User login(UserCredentials userCredentials) {
  User user = userService.getUserByUsername(userCredentials.getUsername());

  if (passwordEncoder.matches(userCredentials.getPassword(), user.getPassword())) {
    return user;
  }

  throw new RuntimeException(WRONG_PASSWORD_MESSAGE);
}
```

Рисунок 2.7 – авторизация.

Также интересный момент был в реализации создания и оформления заказа. Так как хотелось достичь минимального захламления базы дынных не актуальными данными по положению элементов в корзине, и чтобы не добавлять лишние поля в таблицы был реализован метод, который получает все элементы из корзины по идентификатору пользователя, добавляет их в заказ в виде JSON формата, сохраняет заказ и удаляет не актуальные элементы из корзины. Реализация представлена на рисунке 2.8.

```java
@Override
@Transactional
@SneakyThrows
public Order createOrder(Order order) {
  OrderEntity orderForCreate = mapper.dtoToEntity(order);
  List<Bag> bagList = bagService.findAllByUserId(order.getUserId());

  orderForCreate.setData(bagList);
  orderForCreate.setOrderDate(OffsetDateTime.now());
  orderForCreate.setPrice(bagService.sumBags(order.getUserId()));

  Order createdOrder = mapper.entityToDto(orderRepository.save(orderForCreate));

  bagService.deleteAllBagsByUserId(order.getUserId());

  return createdOrder;
}
```

Рисунок 2.8 – создание заказа.

2.4 Проектирование пользовательского интерфейса

Для проектирование пользовательского интерфейса в сфере высоких технологий выделяют определенную должность – это UI/UX дизайнер.

Для данного курсового проекта был выбран дизайн, который уже был разработан и выложен в открытый доступ с помощь платформы «dribbble» и дизайн изображен на рисунке 2.9.

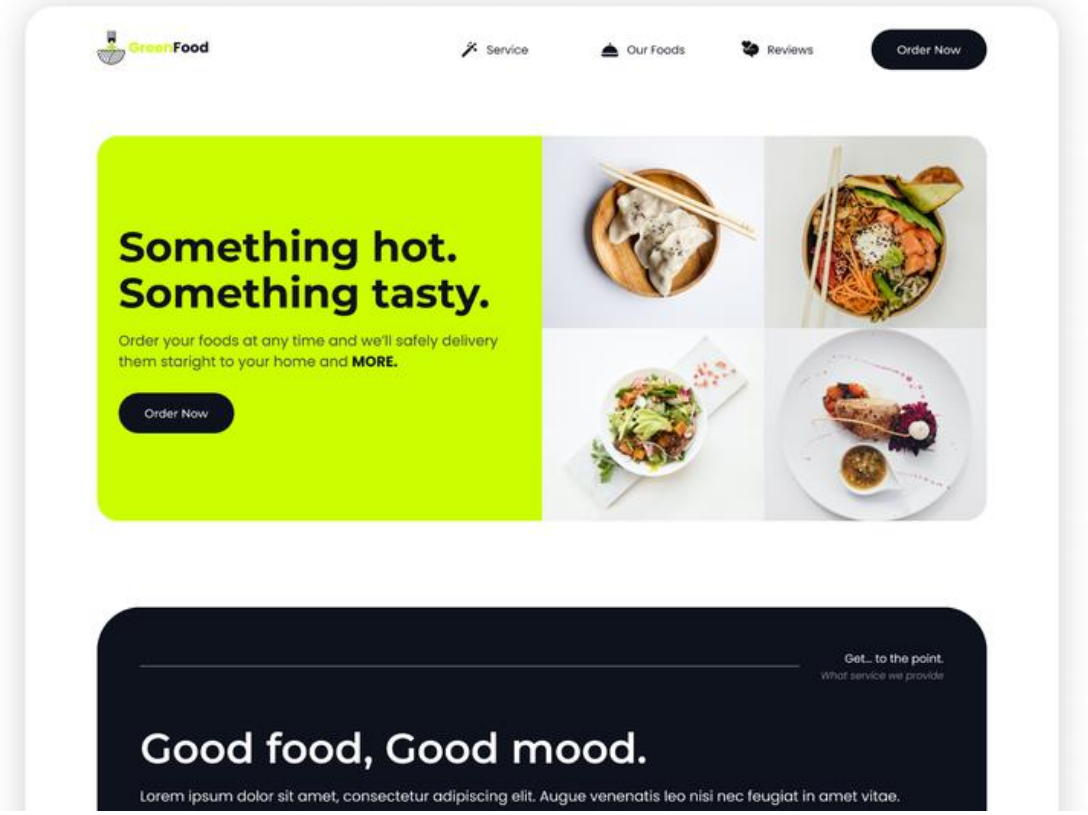

Рисунок 2.9 – дизайн курсового проекта.

Также необходимо упомянуть о том, что такое UI/UX.

UX-дизайн (User Experience — «пользовательский опыт») отвечает за то, как интерфейс работает.

UI-дизайн (User Interface — «пользовательский интерфейс») отвечает за то, как интерфейс выглядит.

Одна часть не может существовать без другой. Сейчас трудно себе представить, что можно разрабатывать интерфейс и не задумываться о том, как он будет выглядеть, и наоборот — оформлять UI в отрыве от проектирования UX.

Рисунок 2.10 – Проектирование визуальной части дизайна

Дизайнер, который работает сразу над UX и UI, чаще всего называется дизайнером интерфейсов, либо UX/UI-дизайнером. Он ведёт весь процесс разработки интерфейса от этапа получения задачи до тестирования прототипа.

Дизайнер пользовательских интерфейсов гибко планирует свою работу. Это значит, что для решения конкретной задачи он может подобрать разный набор действий.

Базово можно разделить работу дизайнера на четыре этапа: исследования, проектирование, визуальный дизайн и проверка решения.

Для подробного изучения задачи дизайнер интерфейсов использует различные инструменты UX-исследования: интервьюирует будущих пользователей, изучает конкурентов и предметную область, анализирует продукт и существующие проблемы.

Дизайнер ищет инсайты. То есть разбирается, что именно нужно пользователю и подойдёт ли ему тот способ решения задачи, который определил заказчик. Дизайнер интерфейсов обязательно общается не только с заказчиком, но и с потенциальным пользователем продукта.

На этом этапе будут созвоны, встречи, уточнения задачи, новые детали. В конце этапа они соберутся в единую картину пожеланий, требований и ограничений.

Представим, что дизайнер получил задачу: спроектировать мобильное приложение и лендинг для сервиса клинико-диагностической лаборатории.

Пользователи этого приложения должны иметь возможность заказать и оплатить анализы, вызвать врача на дом, посмотреть старые результаты анализов и добавить в приложение ещё одного пользователя, например ребёнка.

В ходе обсуждения задачи с заказчиком дизайнер поинтересуется, почему нельзя добавить эти функции в личный кабинет на сайте, почему важно вынести их в отдельное приложение. От заказчика дизайнер узнает, что приложение — это гипотеза. Если большая часть пользователей начнёт

заказывать анализы через приложение, его будут дорабатывать, добавлять новые функции и расширять приложение.

На этапе подготовительных исследований дизайнер интерфейсов визуализирует алгоритм задачи. Здесь он может применить разные инструменты, например:

– пошаговый сценарий пользователя (User Flow), чтобы наглядно увидеть путь пользователя от регистрации до покупки;

– карту пользовательского опыта (CJM, Customer Journey Map), чтобы понимать, где ключевые боли пользователя и как их решить.

Рисунок 2.11 – фрагмент схемы User Flow для «Смартлаба».

Перед тем как приступить к этапу проектирования интерфейса, дизайнер заранее определяет метрики, по которым будет измеряться эффективность дизайна.

В брифе заказчика из примера выше был запрос на два формата: мобильное приложение и лендинг — одностраничный сайт, который расскажет про функции приложения. Для каждого формата будет своя оценка эффективности, например у лендинга это количество скачиваний приложения.

После подготовительного этапа, когда все данные собраны, дизайнер визуализирует структуру продукта или сервиса в виде наброска.

Самый быстрый черновой набросок, который показывает идею, называется вайрфреймом. Обычно проектирование макета идёт по методу

«прогрессивного джипега», когда сначала делается структура целиком, а детали прорабатываются постепенно.

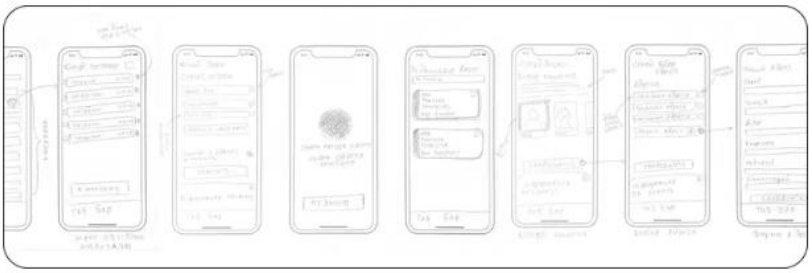

Рисунок 2.12 – быстрый скетч от руки.

Уже на этапе скетча можно отправить спецификацию разработчикам, которые будут программировать приложение. Это нужно, чтобы понять, все ли задумки дизайнера они смогут реализовать.

Вайрфрейм детализируется и становится реалистичным прототипом в процессе согласования с клиентом и параллельного тестирования с разработчиками. На пути из точки А в точку Б макет проходит несколько итераций. UX-дизайнер презентует прототип клиенту, тестирует и улучшает его.

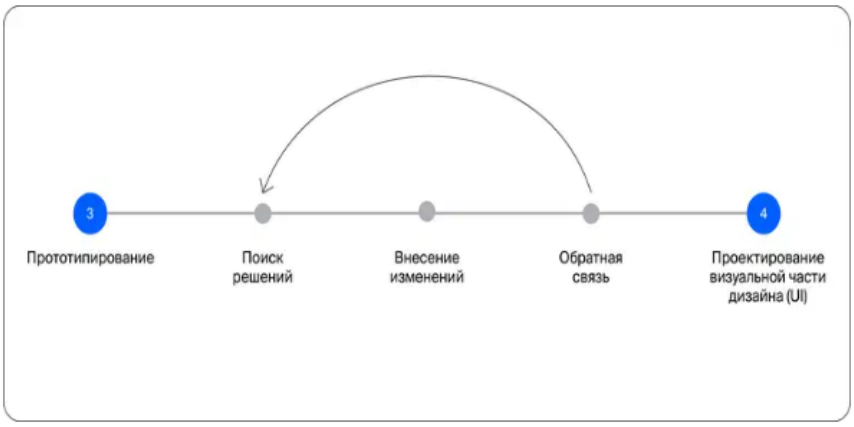

Рисунок 2.13 – итерации проектирования.

У UX-дизайна есть принципы, которые помогают построить эффективный дизайн интерфейса. Начинающим дизайнерам можно ориентироваться на 10 принципов Якоба Нильсена, гуру пользовательского дизайна.

Дизайнер интерфейсов переходит к этапу UI-дизайна как к очередной задаче. Чтобы выполнить часть UI, он должен обладать навыками визуального дизайна.

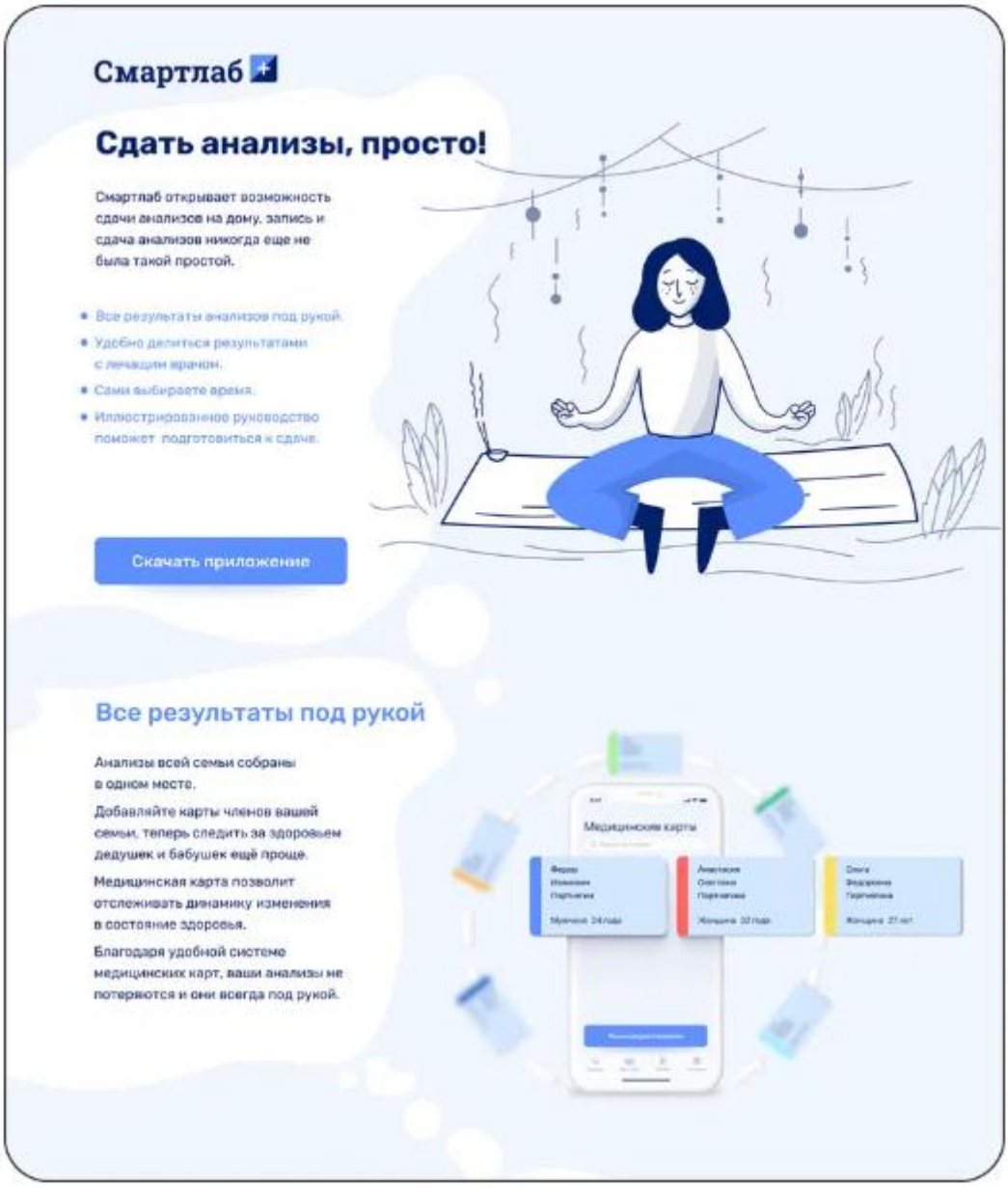

Рисунок 2.14 – фрагмент UI-макета интерфейса лендинга «Смартлаб».

На этом этапе дизайнер решает, как будут выглядеть элементы, подбирает картинки, шрифты, цвета — общий стиль. Если понадобится, то UI-дизайнер создаст анимированные объекты.

В UI-части дизайнер следит не только за общей красотой интерфейса, но и за тем, чтобы пользователь верно считывал элементы интерфейса во всех состояниях экрана, и чтобы интерфейс соответствовал стандартам доступности. Это одновременно относится и к визуальной части, и к части взаимодействия с интерфейсом. То есть в части UI дизайнер продолжает думать про UX интерфейса.

Дизайнер интерфейсов тестирует свой макет с помощью различных инструментов. Самый простой — коридорное тестирование, оно помогает быстро найти грубые ошибки. Дизайнер продумывает сценарий опроса и

опрашивает нескольких возможных пользователей. Потом собирает и анализирует обратную связь, а если нужно, возвращается на предыдущие этапы. Для более сложного тестирования используют юзабилити исследования, АВ-тесты.

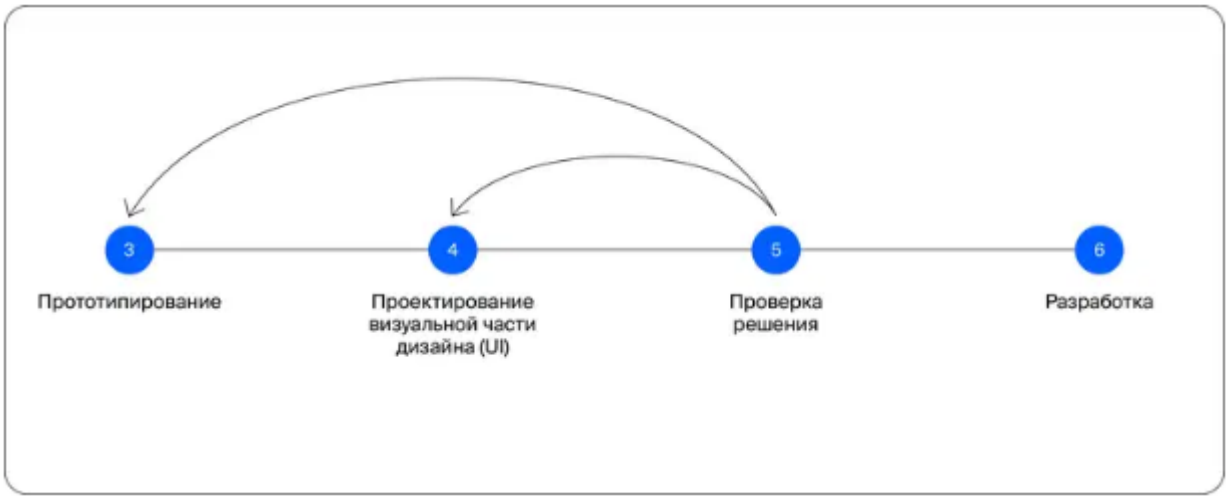

Рисунок 2.15 – тестирование макета на уровне UI или на уровне UX.

В финале дизайнер дорабатывает прототип интерфейса и отдаёт его веб-разработчикам. Он по-прежнему включён в задачу, контролирует процесс вплоть до сдачи готового продукта заказчику [8].

Дизайнер интерфейсов — это адвокат пользователя. На каждом этапе проектирования дизайнер думает сразу и о том, как интерфейс будет работать (UX), и о том, как будет выглядеть (UI). Это забота не об интерфейсе, а о пользователе, будет ли в итоге ему удобно. Да, дизайнер учитывает интересы бизнеса, возможности и ограничения технологий, но его основная цель — разработать интерфейс, который решит задачу пользователя.

2.5 Обоснование выбора компонентов и технологий для реализации программного средства

Как было описано выше в пункте 2.1 используются самые последние и новые технологии и фреймворки, который непосредственно относятся для реализации необходимого функционала для Web-приложения для ресторанного бизнеса с возможностью формирования онлайн заказов и аналитики о работе организации общепита.

3 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА (РАЗРАБОТКА НЕ МЕНЕЕ ЧЕМ ПО 2 ТЕСТ-КЕЙСА НА КАЖДЫЙ ВАРИАНТ ИСПОЛЬЗОВАНИЯ И ИХ РЕАЛИЗАЦИЯ НА JUNIT ИЛИ ДРУГИХ ФРЕЙМВОРКАХ ТЕСТИРОВАНИЯ ПО)

В мире программирования многие аспекты были уставлены и описаны уже давно, это также касается тестирования.

Одни из популярных тестовых принципов это:

– пирамида тестов;

– FIRST.

Давайте поговорим более детально про них.

Пирамида тестирования, также часто говорят уровни тестирования, это группировка тестов по уровню детализации и их назначению. Эту абстракцию придумал Майк Кон и описал в книге «Scrum: гибкая разработка ПО» (Succeeding With Agile. Software Development Using Scrum).

Пирамиду разбивают на 4 уровня (снизу вверх), например, по ISTQB:

– модульное тестирование (юнит);

– интеграционное тестирование;

– системное тестирования;

– приемочное тестирование;

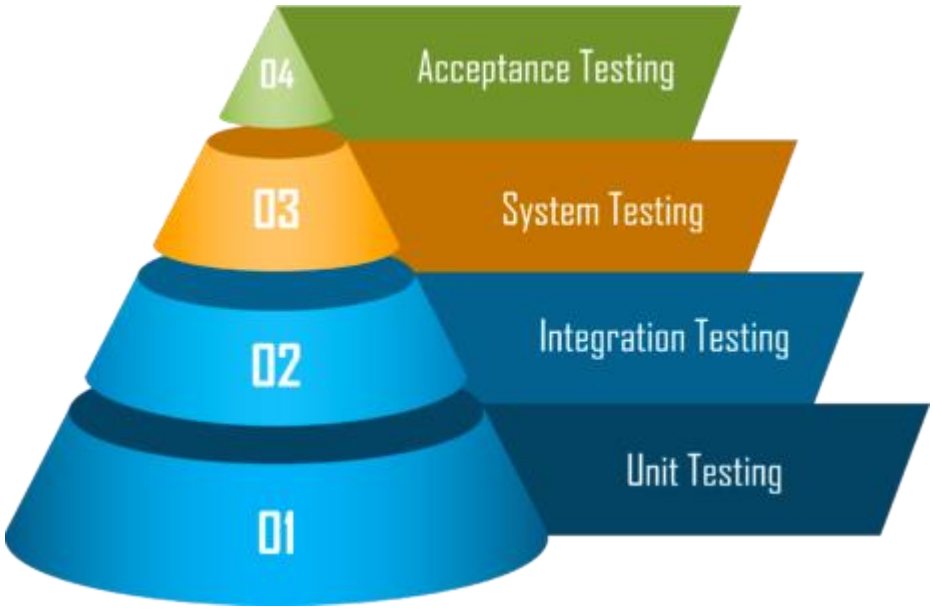

Рисунок 3.1 – пирамида тестов на 4 уровня.

Но можно встретить варианты, где 3 уровня. В этой модели объединяют интеграционный и системный уровни:

– модульное тестирование (юнит);

– интеграционное тестирование (включает в себя системное);

– приемочное тестирование.

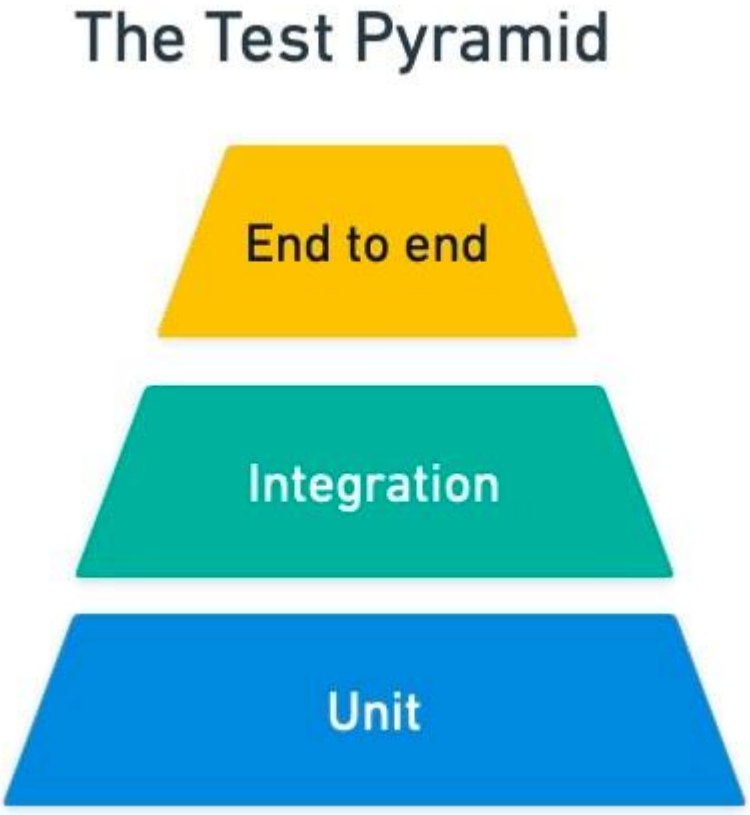

Рисунок 3.2– пирамида тестов на 3 уровня.

Можно сказать, что разработка ПО — это движение по пирамиде снизу вверх. Важно отметить, что тест (ручной, на высоких уровнях, или автотест, на низких уровнях), должен быть на том же уровне, что и тестируемый объект. Например, модульный тест (проверяющий функции, классы, объекты и т.п.) должен быть на компонентном уровне. Это неправильно, если на приемочном уровне запускается тест, который проверят минимальную единицу кода.

Тесты уровнем выше не проверяют логику тестов уровнем/уровнями ниже.

Чем выше тесты уровнем, тем они:

– сложней в реализации, и соответственно, дороже в реализации;

– важнее для бизнеса и критичней для пользователей;

– замедляют скорость прохождения тестовых наборов, например регресса.

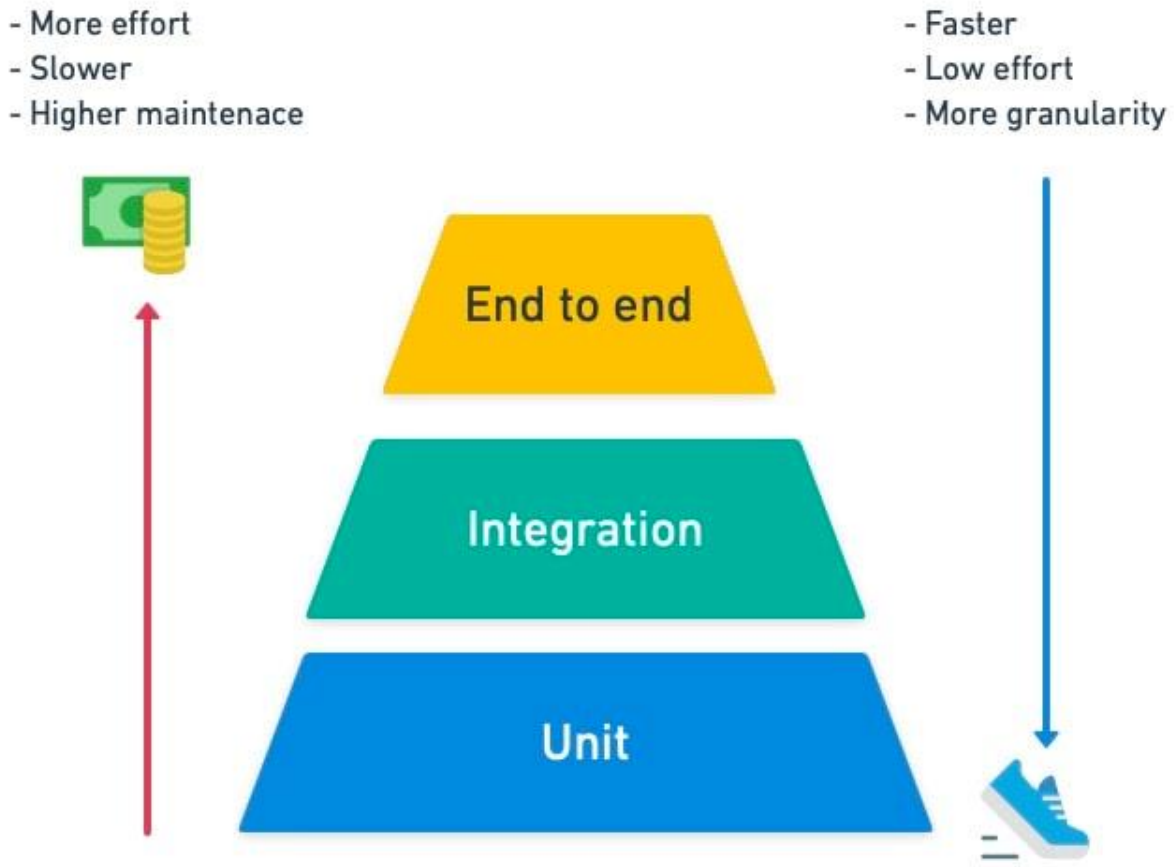

Рисунок 3.3 – описание пирамиды тестов.

Компонентный уровень - чаще всего называют юнит тестированием. Реже называют модульным тестированием. На этом уровне тестируют атомарные части кода. Это могут быть классы, функции или методы классов.

Юнит тесты находят ошибки на фундаментальных уровнях, их легче разрабатывать и поддерживать. Важное преимущество модульных тестов в том, что они быстрые и при изменении кода позволяют быстро провести регресс (убедиться, что новый код не сломал старые части кода).

Тест на компонентном уровне:

– всегда автоматизируют;

– модульных тестов всегда больше, чем тестов с других уровней;

– юнит тесты выполняются быстрее всех и требуют меньше ресурсов;

– практически всегда компонентные тесты не зависят от других модулей (на то они и юнит тесты) и UI системы.

В 99% разработкой модульных тестов занимается разработчик, при нахождении ошибки на этом уровне не создается баг-репортов. Разработчик находит баг, правит, запускает и проверяет (абстрактно говоря это разработка через тестирование) и так по новой, пока тест не будет пройден успешно.

На модульном уровне разработчик (или автотестер) использует метод белого ящика. Он знает, что принимает и отдает минимальная единица кода, и как она работает.

Интеграционный уровень проверят взаимосвязь компоненты, которую проверяли на модульном уровне, с другой или другими компонентами, а также интеграцию компоненты с системой (проверка работы с ОС, сервисами и службами, базами данных, железом и т. д.). Часто в английских статьях называют service test или API test.

В случае с интеграционными тестами редко когда требуется наличие UI, чтобы его проверить. Компоненты ПО или системы взаимодействуют с тестируемым модулем с помощью интерфейсов. Тут начинается участие тестирования. Это проверки API, работы сервисов (проверка логов на сервере, записи в БД) и т. п.

Отдельно отмечу, что в интеграционном тестировании, выполняются как функциональные (проверка по ТЗ), так и нефункциональные проверки (нагрузка на связку компонент). На этом уровне используется либо серый, либо черный ящик.

В интеграционном тестировании есть 3 основных способа тестирования (представь, что каждый модуль может состоять еще из более мелких частей).

Снизу вверх (Bottom Up Integration): все мелкие части модуля собираются в один модуль и тестируются. Далее собираются следующие мелкие модули в один большой и тестируется с предыдущим и т. д. Например, функция публикации фото в соц. профиле состоит из 2 модулей: загрузчик и публикатор. Загрузчик, в свою очередь, состоит из модуля компрессии и отправки на сервер. Публикатор состоит из верификатора (проверяет подлинность) и управления доступом к фотографии. В интеграционном тестировании соберем модули загрузчика и проверим, потом соберем модули публикатора, проверим и протестируем взаимодействие загрузчика и публикатор.

Сверху вниз (Top Down Integration): сначала проверяем работу крупных модулей, спускаясь ниже добавляем модули уровнем ниже. На этапе проверки уровней выше данные, необходимые от уровней ниже, симулируются. Например, проверяем работу загрузчика и публикатора. Руками (создаем функцию-заглушку) передаем от загрузчика публикатору фото, которое якобы было обработано компрессором.

Большой взрыв ("Big Bang" Integration): собираем все реализованные модули всех уровней, интегрируем в систему и тестируем. Если что-то не работает или недоработали, то фиксим или дорабатываем.

О системном уровне говорили в интеграционном. Тут отметить только то, что:

– системный уровень проверят взаимодействие тестируемого ПО с системой по функциональным и нефункциональным требованиям;

– важно тестировать на максимально приближенном окружении, которое будет у конечного пользователя.

Тест-кейсы на этом уровне подготавливаются:

– по требованиям;

– по возможным способам использования ПО.

На системном уровне выявляются такие дефекты, как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т. д.

На этом уровне используют черный ящик. Интеграционный уровень позволяет верифицировать требования (проверить соответствие ПО прописанным требованиям).

Приемочное тестирование также часто называют E2E тестами (End-2-End) или сквозными. На этом уровне происходит валидация требований (проверка работы ПО в целом, не только по прописанным требованиям, что проверили на системном уровне).

Проверка требований производится на наборе приемочных тестов. Они разрабатываются на основе требований и возможных способах использования ПО.

Отмечу, что приемочные тесты проводят, когда (1) продукт достиг необходимо уровня качества и (2) заказчик ПО ознакомлен с планом приемки (в нем описан набор сценариев и тестов, дата проведения и т. п.).

Приемку проводит либо внутреннее тестирование (необязательно тестировщики) или внешнее тестирование (сам заказчик и необязательно тестировщик).

Важно помнить, что E2E тесты автоматизируются сложнее, дольше, стоят дороже, сложнее поддерживаются и трудно выполняются при регрессе. Значит таких тестов должно быть меньше.

Многим разработчикам известны принципы проектирования, которые благодаря Роберту Мартину получили звучное название S.O.L.I.D. Многим из этих принципов уже не один десяток лет (принцип подстановки Лисков впервые был озвучен более двадцати лет назад), они были опробованы на миллионах строк кода, тысячами разработчиками (при этом добрая половина из них применяла эти принципы даже не имея понятия об этом:)). Конечно, слепое следование любым принципам никогда ни к чему хорошему не приводило и приводить не будет, но тем не менее в них описаны разумные вещи, о которых нужно как минимум знать, да и понимать их тоже будет совсем не лишним.

Помимо принципов проектирования существуют и другие, незаслуженно менее известные принципы, которые положены в основу написания качественных тестов. На каждом шагу говорится о качестве кода, продуманности дизайна или архитектуры, но при этом довольно слабо уделяется внимание читабельности и сопровождаемости тестов. А ведь объем кода в тестах по-хорошему может быть (а точнее должен быть) не меньшим, чем объем кода; возможно именно из-за некачественного кода модульных тестов они так редко поддерживаются в актуальном состоянии, что очень быстро приводит к тому, что они устаревают, становятся неактуальными и вообще остаются на “обочине” процесса разработки.

Принципы написания качественных тестов придуманы не на пустом месте. Большинство опытных разработчиках знают о них точно так же, как и о принципах проектирования без помощи “Дядюшки” Боба, но как и в случае с принципами проектирования именно Боб Мартин объединил пять принципов тестирования, в результате чего получилось звучное название Б.Н.П.О.С. (или F.I.R.S.T., если вам звучность русскоязычного названия не по душе): Fast (Быстрота), Independent (Независимость), Repeatable (Повторяемость), Self-Validating (Очевидность) (*), Timely (Своевременность)).

Итак, каждый модульный тест должен обладать следующими характеристиками.

Быстрота (Fast). Тесты должны выполняться быстро. Все мы знаем, что разработчики люди, а люди ленивы, поскольку эти выражения являются “транзитивными”, то можно сделать вывод, что люди тоже ленивы. А ленивый человек не захочет запускать тесты при каждом изменении кода, если они будут долго выполняться.

Независимость (Independent). Результаты выполнения одного теста не должны быть входными данными для другого. Все тесты должны выполняться в произвольном порядке, поскольку в противном случае при сбое одного теста каскадно “накроется” выполнение целой группы тестов.

Повторяемость (Repeatable). Тесты должны давать одинаковые результаты не зависимо от среды выполнения. Результаты не должны зависеть от того, выполняются ли они на вашем локальном компьютере, на компьютере соседа или же на билд-сервере. В противном случае найти концы с концами будет весьма не просто.

Очевидность (Self-Validating). Результатом выполнения теста должно быть булево значение. Тест либо прошел, либо не прошел и это должно быть легко понятно любому разработчику. Не нужно заставлять людей читать логи только для того, чтобы определить прошел тест успешно или нет.

Своевременность (Timely). Тесты должны создаваться своевременно. Несвоевременность написания тестов является главной причиной того, что они откладываются на потом, а это “потом” так никогда и не наступает. Даже если вы и не будете писать тесты перед кодом (хотя этот вариант уже доказал свою жизнеспособность) их нужно писать, как минимум параллельно с кодом.

Исходя из описанного выше были написаные следующие тесты на рисунках 3.4–3.6 для данного приложения.

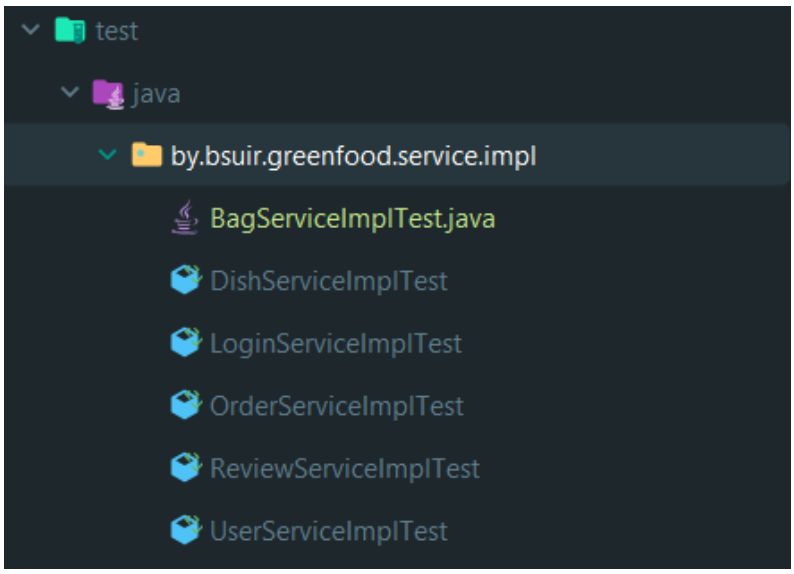

Рисунок 3.4 – юнит тесты для сервисного слоя.

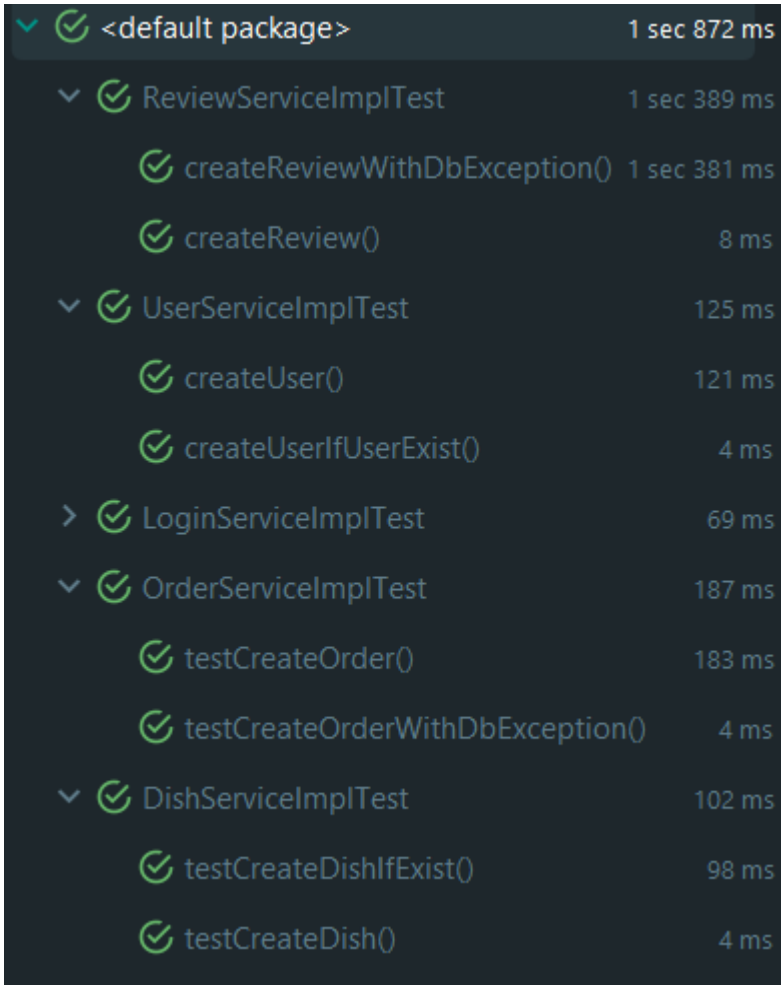

Рисунок 3.5 – результаты выполнения юнит тестов для сервисного слоя.

Рисунок 3.6 – реализация юнит тестов для логина.

4 ИНСТРУКЦИЯ ПО РАЗВЕРТЫВАНИЮ ПРИЛОЖЕНИЯ И СКВОЗНОЙ ТЕСТОВЫЙ ПРИМЕР, НАЧИНАЯ ОТ АВТОРИЗАЦИИ, ДЕМОНСТРИРУЯ РЕАЛИЗАЦИЮ ВСЕХ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Как было описано в пункте 2.2 для развертывания приложения используется контейнеризация, а именно Docker compose.

Перейдем к тестовому варианта использования для заказа еды.

Главный экран изображен на рисунке 4.1.

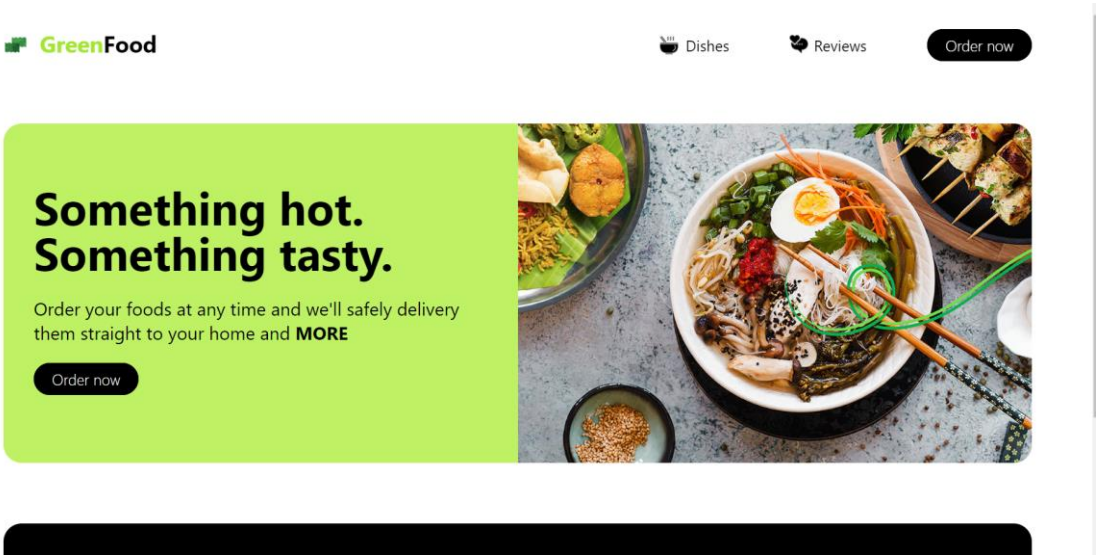

Рисунок 4.1 – главный экран.

Экран авторизации изображен на рисунке 4.2.

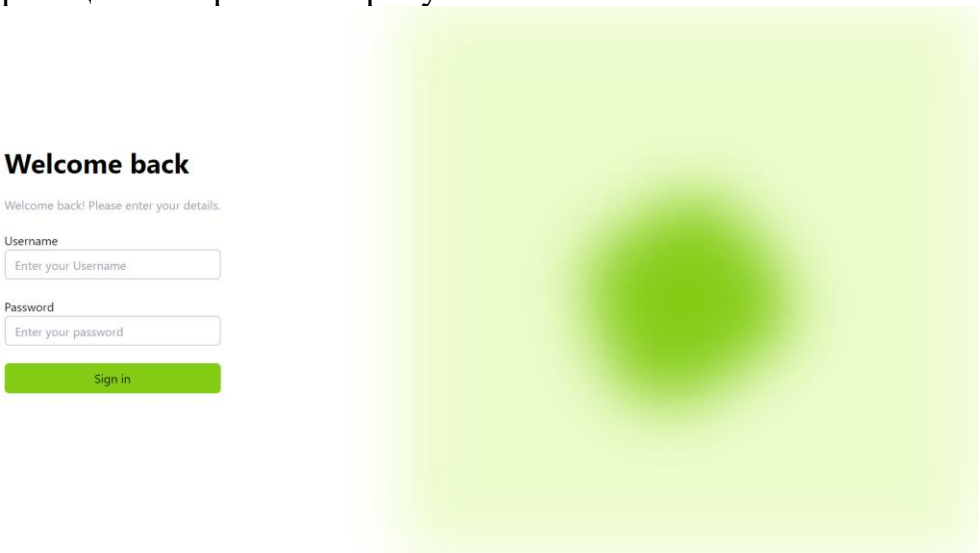

Рисунок 4.2 – экран авторизации.

После введения логина и пароля происходит редирект на главный экран после чего переходим на экран блюд, изображен на рисунке 4.3.

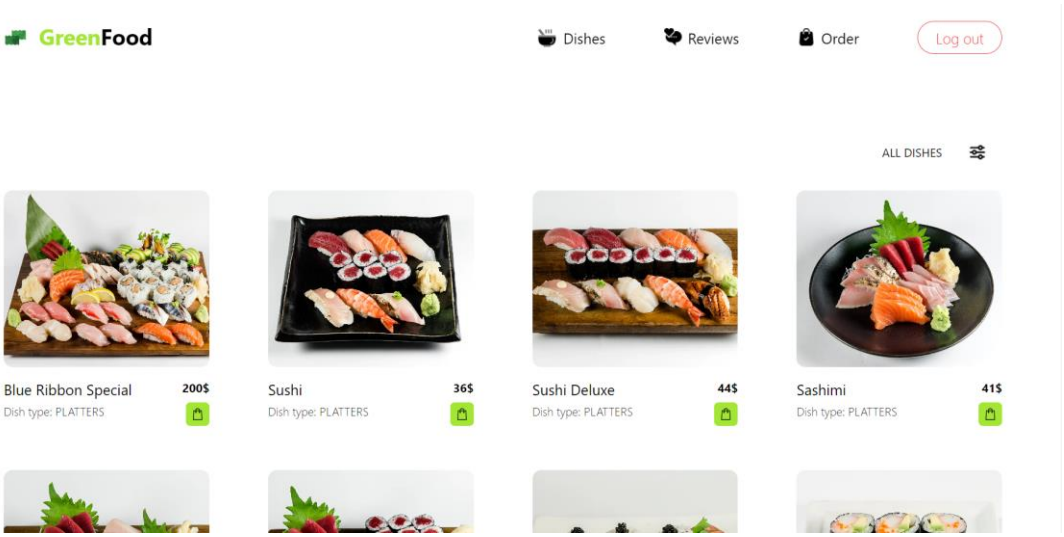

Рисунок 4.3 – экран блюд.

Для добавления блюда в корзину нажимаем на зеленую кнопку корзины и переходим на экран заказа, изображенного на рисунке 4.4.

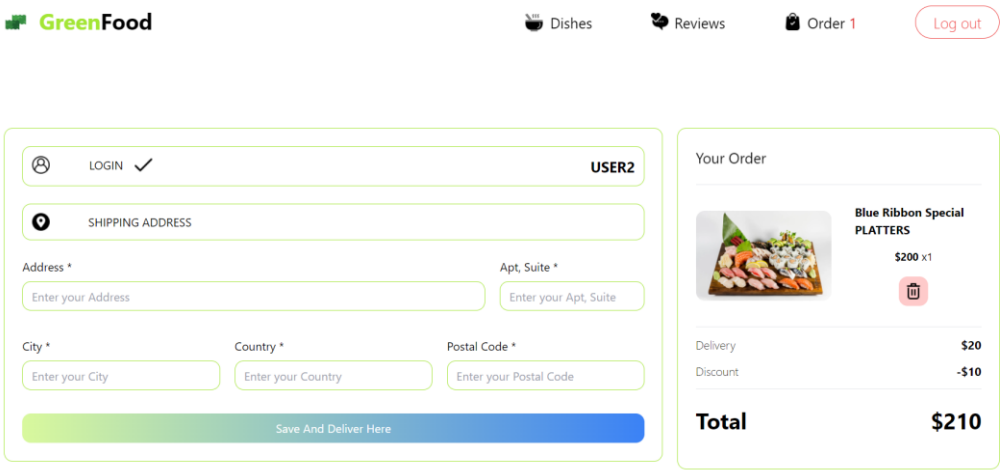

Рисунок 4.4 – экран заказа.

Вводим все данные и нажимаем на кнопку заказа, после чего мы увидим анимацию, которая говорит о успешном заказа, изображена на рисунке 4.5 и после этого происходит редирект на главный экран.

Рисунок 4.5 – анимация, которая говорит о успешном заказа.

Это основной ход действия для данного приложения, но у него еще есть очень много функционала:

– авторизация пользователя;
– режим для пользователя;
– режим для администратора;
– просмотр блюд;
– добавление блюд в заказ;
– удаление блюд из заказа;
– оформление заказа на определенный адрес;
– просмотр комментариев пользователей;
– добавление комментариев;
– удаление комментариев;
– просмотр аналитики;
– добавление блюд в список блюд при наличии учетной записи администратора.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы было создано программное средство, для ресторанного бизнеса с возможностью формирования онлайн заказов и аналитики о работе организации общепита.

В ходе создания программного средства были подробно изучены особенности внедрения информационных технологий в сферу ресторанного бизнеса. Полученная информация позволила построить функциональную модель IDEF0, которая наглядно отображает процесс работы ресторана. Были показаны и описаны диаграммы UML, с помощью которых было выполнено проектирование системы. Также была рассмотрена архитектура созданного программного средства. Помимо этого, в ходе выполнения проекта было составлено руководство пользователю, где понятным и доступным языком описывается принцип работы программы. В завершение работы было проведено тестирование разработанной системы, подтвердившее работоспособность созданного программного средства.

Основные функции программного средства реализованы в соответствии с выявленными особенностями предметной области. Был разработан довольно широкий функционал для работы с информацией, которая содержится в базе данных. Стиль интерфейса программы создавался с упором на массовость потребления и использования, который позволит любому пользователю легко и просто использовать данное программное средство без лишних временных затрат.

Бизнес-логика системы даёт возможность систематизировать данные в удобной форме. Полученные результаты могут быть очень эффективны при подведении итогов работы учреждения, а также при определении дальнейшей стратегии развития.

Одним из отличий данного программного средства является надёжная и безопасная база данных, а также структурированная система учёта информации. Благодаря этому вероятность искажения информации минимальна.

В будущем возможно рассмотрение вопроса о расширении функционала программы или же усовершенствования имеющегося. Это обеспечит расширение спектра применения разработанного программного средства.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Национальный правовой интернет-портал Республики Беларусь [Электронный ресурс]. – Электронные данные. – Режим доступа: http://pravo.by/document/?guid=3871&p0=v19302435.

[2] Академия профессионального развития [Электронный ресурс]. – Электронные данные. – Режим доступа: https://academy-prof.ru/blog/informacionnye-tehnologii-v-medicine.

[3] Блинов, И. Н. Java. Промышленное программирование: практическое пособие / И. Н. Блинов, В. С. Романчик. – Мн.: Универсал-Пресс, 2012.

[4] Буч, Г. UML. Классика CS / Г. Буч, Дж. Рамбо, А. Якобсон; пер. с англ. – 2-е изд. – СПб.: Питер, 2006.

[5] Методология IDEF0 [Электронный ресурс]. – Режим доступа: https://itteach.ru/bpwin/metodologiya-idef0

[6] Проектирование, разработка и сопровождение баз данных с использованием CASE – средств [Электронный ресурс]. – Режим доступа: https://www.bsuir.by/m/12_100229_1_90142.pdf

[7] YouTube [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.youtube.com/user/PlurrimiTube/featured.

[8] JavaRush [Электронный ресурс]. – Электронные данные. – Режим доступа: https://javarush.ru.

ПРИЛОЖЕНИЕ А
(обязательное)
Отчет о проверке на заимствования в системе «Антиплагиат»

ПРИЛОЖЕНИЕ Б
(обязательное)
Листинг кода алгоритмов, реализующих основную бизнес-логику

```
package by.bsuir.greenfood.service.impl;

import by.bsuir.greenfood.mapper.BagMapper;
import by.bsuir.greenfood.model.dto.Bag;
import by.bsuir.greenfood.model.dto.Dish;
import by.bsuir.greenfood.model.entity.BagEntity;
import by.bsuir.greenfood.repo.BagRepository;
import by.bsuir.greenfood.service.BagService;
import by.bsuir.greenfood.service.DishService;
import by.bsuir.greenfood.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;
import java.util.UUID;

/**
 * DESCRIPTION
 *
 * @author Vladislav_Karpeka
 * @version 1.0.0
 */
@Service
@RequiredArgsConstructor
public class BagServiceImpl implements BagService {

  private static final String BAG_NOT_FOUND_MESSAGE = "Bag not found!";

  private final BagMapper mapper;
  private final BagRepository repository;
  private final UserService userService;
  private final DishService dishService;

  @Autowired
  private BagService bagService;

  @Override
  @Transactional
  public void createBag(UUID userId, UUID dishId) {
    Bag bag = bagService.findByUserIdAndDishId(userId, dishId);

    if (bag != null) {
      bag.setCount(bag.getCount() + 1);
      repository.save(mapper.dtoToEntity(bag));
    } else {
      userService.getUserById(userId);
      dishService.getDishById(dishId);

      repository.save(new BagEntity(userId, dishId, 1));
    }
  }

  @Override
  @Transactional
  public int countBags(UUID userId) {
```

```
    return bagService.findAllByUserId(userId).stream()
        .map(Bag::getCount)
        .reduce(0, Integer::sum);
  }

  @Override
  @Transactional
  public Double sumBags(UUID userId) {
    return bagService.findAllByUserId(userId).stream()
        .map(bag -> {
          Dish dish = dishService.getDishById(bag.getDishId());
          return bag.getCount() * dish.getPrice();
        })
        .reduce(0.0, Double::sum);
  }

  @Override
  @Transactional
  public List<Bag> findAllByUserId(UUID userId) {
    return repository.findAllByUserId(userId).stream()
        .map(mapper::entityToDto)
        .toList();
  }

  @Override
  @Transactional
  public Bag findByUserIdAndDishId(UUID userId, UUID dishId) {
    return repository.findByUserIdAndDishId(userId, dishId)
        .map(mapper::entityToDto)
        .orElse(null);
  }

  @Override
  public void deleteAllBagsByUserId(UUID userId) {
    repository.deleteAllByUserId(userId);
  }

  @Override
  @Transactional
  public void deleteBag(UUID userId, UUID dishId) {
    Bag bag = Optional.ofNullable(bagService.findByUserIdAndDishId(userId,
dishId))
        .orElseThrow(() -> new RuntimeException(BAG_NOT_FOUND_MESSAGE));

    if (bag.getCount() == 1) {
      repository.deleteByUserIdAndDishId(userId, dishId);
    } else {
      bag.setCount(bag.getCount() - 1);
      repository.save(mapper.dtoToEntity(bag));
    }
  }
}

package by.bsuir.greenfood.service.impl;

import by.bsuir.greenfood.mapper.DishMapper;
import by.bsuir.greenfood.model.dto.Dish;
import by.bsuir.greenfood.model.entity.DishEntity;
import by.bsuir.greenfood.model.enums.DishType;
import by.bsuir.greenfood.repo.DishRepository;
import by.bsuir.greenfood.service.DishService;
import jakarta.persistence.EntityNotFoundException;
import lombok.RequiredArgsConstructor;
```

```
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;
import java.util.UUID;

/**
 * DESCRIPTION
 *
 * @author Vladislav_Karpeka
 * @version 1.0.0
 */
@Service
@RequiredArgsConstructor
public class DishServiceImpl implements DishService {

  private final DishMapper mapper;

  private final DishRepository dishRepository;

  @Override
  public Dish createDish(Dish dish) {
    Optional<Dish> createdDish = getDishByName(dish.getName());
    if (createdDish.isPresent()) {
      return createdDish.get();
    }

    DishEntity dishEntity = mapper.dtoToEntity(dish);
    return mapper.entityToDto(dishRepository.save(dishEntity));
  }

  @Override
  public Dish updateDish(Dish dish) {
    getDishById(dish.getId());

    DishEntity dishToUpdate = mapper.dtoToEntity(dish);
    return mapper.entityToDto(dishRepository.save(dishToUpdate));
  }

  @Override
  public List<Dish> getDishes(DishType dishType) {
    if (dishType == null) {
      return dishRepository.findAll().stream()
          .map(mapper::entityToDto)
          .toList();
    }

    return dishRepository.findAllByDishType(dishType).stream()
        .map(mapper::entityToDto)
        .toList();
  }

  @Override
  public Optional<Dish> getDishByName(String name) {
    return dishRepository.findDishByName(name)
        .map(mapper::entityToDto);
  }

  @Override
  @Transactional
  public Dish getDishById(UUID id) {
    return dishRepository.findById(id)
```

```
        .map(mapper::entityToDto)
        .orElseThrow(() -> new EntityNotFoundException("User not found with
id " + id));
  }

  @Override
  public void deleteDishById(UUID id) {
    dishRepository.deleteById(id);
  }
}
```

```
package by.bsuir.greenfood.service.impl;

import by.bsuir.greenfood.model.dto.UserCredentials;
import by.bsuir.greenfood.model.dto.User;
import by.bsuir.greenfood.service.LoginService;
import by.bsuir.greenfood.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

/**
 * DESCRIPTION
 *
 * @author Vladislav_Karpeka
 * @version 1.0.0
 */
@Service
@RequiredArgsConstructor
public class LoginServiceImpl implements LoginService {

  private static final String WRONG_PASSWORD_MESSAGE = "Wrong password!";

  private final UserService userService;
  private final PasswordEncoder passwordEncoder;

  @Override
  public User login(UserCredentials userCredentials) {
    User user = userService.getUserByUsername(userCredentials.getUsername());

    if (passwordEncoder.matches(userCredentials.getPassword(),
user.getPassword())) {
      return user;
    }

    throw new RuntimeException(WRONG_PASSWORD_MESSAGE);
  }
}
```

```
package by.bsuir.greenfood.service.impl;

import by.bsuir.greenfood.mapper.OrderMapper;
import by.bsuir.greenfood.model.dto.Bag;
import by.bsuir.greenfood.model.dto.Order;
import by.bsuir.greenfood.model.entity.OrderEntity;
import by.bsuir.greenfood.repo.OrderRepository;
import by.bsuir.greenfood.service.BagService;
import by.bsuir.greenfood.service.OrderService;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.AllArgsConstructor;
import lombok.SneakyThrows;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
```

```
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.time.OffsetDateTime;
import java.util.List;
import java.util.UUID;

/**
 * DESCRIPTION
 *
 * @author Vladislav_Karpeka
 * @version 1.0.0
 */
@Service
@AllArgsConstructor
public class OrderServiceImpl implements OrderService {

  private static final ObjectMapper MAPPER = new ObjectMapper();

  private final OrderMapper mapper;

  private final OrderRepository orderRepository;
  private final BagService bagService;

  @Override
  @Transactional
  @SneakyThrows
  public Order createOrder(Order order) {
    OrderEntity orderForCreate = mapper.dtoToEntity(order);
    List<Bag> bagList = bagService.findAllByUserId(order.getUserId());

    orderForCreate.setData(bagList);
    orderForCreate.setOrderDate(OffsetDateTime.now());
    orderForCreate.setPrice(bagService.sumBags(order.getUserId()));

    Order createdOrder =
mapper.entityToDto(orderRepository.save(orderForCreate));

    bagService.deleteAllBagsByUserId(order.getUserId());

    return createdOrder;
  }

  @Override
  public Order updateOrder(Order order) {
    getOrderById(order.getId());

    OrderEntity orderForUpdate = mapper.dtoToEntity(order);
    return mapper.entityToDto(orderRepository.save(orderForUpdate));
  }

  @Override
  public Order getOrderById(UUID id) {
    return orderRepository.findById(id).map(mapper::entityToDto)
        .orElseThrow(() -> new UsernameNotFoundException("Order not found
with id " + id));
  }

  @Override
  public List<Order> getOrdersByOwnerId(UUID id) {
    return
orderRepository.findAllByOwnerId(id).stream().map(mapper::entityToDto).toList
();
```

```
  }

  @Override
  public List<Order> getOrders() {
    return
orderRepository.findAll().stream().map(mapper::entityToDto).toList();
  }

  @Override
  public void deleteOrder(UUID id) {
    orderRepository.deleteById(id);
  }
}
```

```
package by.bsuir.greenfood.service.impl;

import by.bsuir.greenfood.mapper.ReviewMapper;
import by.bsuir.greenfood.model.dto.Review;
import by.bsuir.greenfood.model.entity.ReviewEntity;
import by.bsuir.greenfood.repo.ReviewRepository;
import by.bsuir.greenfood.service.ReviewService;
import lombok.AllArgsConstructor;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.time.OffsetDateTime;
import java.util.Comparator;
import java.util.List;
import java.util.UUID;

/**
 * DESCRIPTION
 *
 * @author Vladislav_Karpeka
 * @version 1.0.0
 */
@Service
@AllArgsConstructor
public class ReviewServiceImpl implements ReviewService {

  private final ReviewMapper mapper;

  private final ReviewRepository reviewRepository;

  @Override
  public Review createReview(Review review) {
    ReviewEntity reviewForCreate = mapper.dtoToEntity(review);

    reviewForCreate.setReviewDate(OffsetDateTime.now());

    return mapper.entityToDto(reviewRepository.save(reviewForCreate));
  }

  @Override
  public Review updateReview(Review review) {
    getReviewById(review.getId());

    ReviewEntity reviewForUpdate = mapper.dtoToEntity(review);
    return mapper.entityToDto(reviewRepository.save(reviewForUpdate));
  }

  @Override
```

```
  public Review getReviewById(UUID id) {
    return reviewRepository.findById(id).map(mapper::entityToDto)
        .orElseThrow(() -> new UsernameNotFoundException("Review not found
with id " + id));
  }

  @Override
  public List<Review> getReviewsByCommenterId(UUID id) {
    return
reviewRepository.findAllByCommenterId(id).stream().map(mapper::entityToDto).t
oList();
  }

  @Override
  public List<Review> getReviews() {
    return reviewRepository.findAll().stream()
        .sorted(Comparator.comparing(ReviewEntity::getReviewDate).reversed())
        .map(mapper::entityToDto)
        .toList();
  }

  @Override
  public void deleteReview(UUID id) {
    reviewRepository.deleteById(id);
  }
}
```

```
package by.bsuir.greenfood.service.impl;

import by.bsuir.greenfood.mapper.UserMapper;
import by.bsuir.greenfood.model.dto.User;
import by.bsuir.greenfood.model.entity.UserEntity;
import by.bsuir.greenfood.repo.UserRepository;
import by.bsuir.greenfood.service.UserService;
import lombok.AllArgsConstructor;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.UUID;

/**
 * DESCRIPTION
 *
 * @author Vladislav_Karpeka
 * @version 1.0.0
 */
@Service
@AllArgsConstructor
public class UserServiceImpl implements UserService, UserDetailsService {

  private final UserMapper mapper;

  private final PasswordEncoder passwordEncoder;

  private final UserRepository repository;

  @Override
```

```
  public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
    return repository.findByUsername(username)
        .orElseThrow(() -> new UsernameNotFoundException("User not found with
username " + username));
  }

  @Override
  public User createUser(User user) {
    getUserByUsername(user.getUsername());

    UserEntity userForCreate = mapper.dtoToEntity(user);

userForCreate.setPassword(passwordEncoder.encode(userForCreate.getPassword())
);

    return mapper.entityToDto(repository.save(userForCreate));
  }

  @Override
  public User updateUser(User user) {
    getUserById(user.getId());

    UserEntity userForUpdate = mapper.dtoToEntity(user);
    return mapper.entityToDto(repository.save(userForUpdate));
  }

  @Override
  @Transactional
  public User getUserByUsername(String username) {
    return repository.findByUsername(username).map(mapper::entityToDto)
        .orElseThrow(() -> new UsernameNotFoundException("User not found with
username " + username));
  }

  @Override
  @Transactional
  public User getUserById(UUID id) {
    return repository.findById(id).map(mapper::entityToDto)
        .orElseThrow(() -> new UsernameNotFoundException("User not found with
id " + id));
  }

  @Override
  public List<User> getUsers() {
    return repository.findAll().stream().map(mapper::entityToDto).toList();
  }

  @Override
  public void deleteUser(UUID id) {
    repository.deleteById(id);
  }
}
```

ПРИЛОЖЕНИЕ В
(обязательное)
Листинг скрипта генерации базы данных

```
create table users (
id uuid not null,
icon_url varchar(355) not null,
user_type varchar(255) not null ,
is_account_non_expired boolean not null,
is_account_non_locked boolean not null,
is_credentials_non_expired boolean not null,
is_enabled boolean not null,
password varchar(255) not null ,
username varchar(255) not null ,
primary key (id));

create table dishes (
id uuid not null,
icon_url varchar(255) not null,
dish_type varchar(255) not null,
description varchar(255) not null,
name varchar(255) not null,
price float8 not null,
primary key (id));

create table bags (
count int4 not null,
user_id uuid not null,
dish_id uuid not null,
primary key (user_id, dish_id));

create table orders (
id uuid not null,
address varchar(255) not null,
order_date timestamp not null,
price float8 not null,
user_id uuid not null,
data jsonb,
primary key (id));

create table reviews (
id uuid not null,
review varchar(65535) not null,
review_date timestamp not null,
user_id uuid not null,
primary key (id));

alter table if exists users add constraint UK_r43af9ap4edm43mmtq01oddj6
unique (username);

alter table if exists dishes add constraint UK_r12af9ap4edm43mmtq45oddj9
unique (name);
alter table if exists dishes add constraint UK_r12af9ap4edm43mmtq45oddj1
unique (icon_url);

alter table if exists bags add constraint UK_r12af9ap4edm43mmtq22oddj1
foreign key (user_id) references users;
alter table if exists bags add constraint UK_r12af9ap4edm43mmtq33oddj9
foreign key (dish_id) references dishes;

alter table if exists orders add constraint FK32ql8ubntj5uh44ph9659tiih
foreign key (user_id) references users;
```

```
alter table if exists reviews add constraint FKcgy7qjc1r99dp117y9en6lxye
foreign key (user_id) references users;

insert into users (id, icon_url, user_type, is_account_non_expired,
is_account_non_locked, is_credentials_non_expired, is_enabled, password,
username) values
 ('e03a1618-f42f-4579-b006-e54f02b207ba',
'https://st2.depositphotos.com/2777531/6975/v/950/depositphotos_69756347-
stock-illustration-laughing-cartoon-guy.jpg', 'ADMIN', true, true, true,
true, '$2a$10$0Tj3sv/9GrOASKSri0QKB.TGkglBh1p3iITrDmr9/11gL1nZ5WViW',
'admin'),
 ('e03a1618-f42f-4579-b006-e45f02b201ba',
'https://i.pinimg.com/originals/89/90/48/899048ab0cc455154006fdb9676964b3.jpg
', 'USER', true, true, true, true,
'$2a$10$e7FNTsXe2dOQbyPxU4nXJuLLMG1nT3L3EQlGvpTT9d2Mv2jKdyTh6', 'user'),
 ('e03a1618-f42f-4579-b006-e45f02b202ba',
'https://st2.depositphotos.com/3369547/11458/v/950/depositphotos_114587292-
stock-illustration-man-icon-avatar-person-design.jpg', 'USER', true, true,
true, true, '$2a$10$e7FNTsXe2dOQbyPxU4nXJuLLMG1nT3L3EQlGvpTT9d2Mv2jKdyTh6',
'user1'),
 ('e03a1618-f42f-4579-b006-e45f02b203ba',
'https://png.pngtree.com/element_our/png_detail/20181228/avatar-vector-icon-
png_296057.jpg', 'USER', true, true, true, true,
'$2a$10$e7FNTsXe2dOQbyPxU4nXJuLLMG1nT3L3EQlGvpTT9d2Mv2jKdyTh6', 'user2'),
 ('e03a1618-f42f-4579-b006-e45f02b204ba',
'https://yt3.ggpht.com/ytc/AKedOLRFVBlbKKW4mBFpX4kR29Ibx7ihZpU7wPwR12D5ew=s90
0-c-k-c0x00ffffff-no-rj', 'USER', true, true, true, true,
'$2a$10$e7FNTsXe2dOQbyPxU4nXJuLLMG1nT3L3EQlGvpTT9d2Mv2jKdyTh6', 'user3'),
 ('e03a1618-f42f-4579-b006-e45f02b205ba',
'https://st3.depositphotos.com/1077687/13865/v/450/depositphotos_138659994-
stock-illustration-isolated-avatar-man-design.jpg', 'USER', true, true, true,
true, '$2a$10$e7FNTsXe2dOQbyPxU4nXJuLLMG1nT3L3EQlGvpTT9d2Mv2jKdyTh6',
'user4'),
 ('e03a1618-f42f-4579-b006-e45f02b206ba', 'https://shop.solo-
it.ru/upload/iblock/bcd/logo_market.png', 'USER', true, true, true, true,
'$2a$10$e7FNTsXe2dOQbyPxU4nXJuLLMG1nT3L3EQlGvpTT9d2Mv2jKdyTh6', 'user5');

insert into dishes (id, icon_url, dish_type, description, name, price) values
('e01a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/22e35d3f-ea69-409c-b085-
0c4b910efdfe.jpg', 'PLATTERS', 'description', 'Blue Ribbon Special', 200.0),
('e02a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/96953250-e6e2-4b05-ad8a-
23aab90b5b82.jpg', 'PLATTERS', 'description', 'Sushi', 36.0),
('e03a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/4e05fece-6789-4dbb-a527-
1dce79c55b15.jpg', 'PLATTERS', 'description', 'Sushi Deluxe', 44.0),
('e04a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/73cac2b6-e099-4f09-b19f-
17dbf2b585ee.jpg', 'PLATTERS', 'description', 'Sashimi', 41.0),
('e05a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/cee4bc06-ab48-48df-86f2-
9da201a3482d.jpg', 'PLATTERS', 'description', 'Sashimi Deluxe', 52.0),
('e06a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/f1e4ac0d-22ed-4ea1-a853-
```

```
ffa88c792cb8.jpg', 'PLATTERS', 'description', 'Sushi & Sashimi Combination',
56.0),

('e07a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/45bf5a51-0676-4a04-8a14-
460796db416d.jpg', 'ROLLS', 'description', 'Blue Ribbon', 32.0),
('e08a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/2de274f2-b146-4c3a-90b6-
34c205656361.jpg', 'ROLLS', 'description', 'California With Kanikama', 11.0),
('e09a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/fbf09705-ee2b-488f-9185-
1f0a3eb440c3.jpg', 'ROLLS', 'description', 'California With Blue Crab',
16.0),
('e10a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/08187594-f3d3-49e2-a47a-
67c231fae330.jpg', 'ROLLS', 'description', 'Negi Toro', 18.0),
('e11a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/785ee42b-f68e-43cc-85e2-
f029980c6433.jpg', 'ROLLS', 'description', 'Spicy Tuna', 12.5),
('e12a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/nzjeiphw/61cb251c-23e8-4a16-9443-
30e8bbe59ce5.jpg', 'ROLLS', 'description', 'Spicy Tuna & Tempura Flakes',
14.5),

('e13a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/ca662c28-4dcd-4f25-a448-
48219b47d612.jpg', 'SUSHI', 'description', 'Yaki Zake', 8.0),
('e14a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/653c3b26-0e39-4d47-b9e1-
9bc17ad143db.jpg', 'SUSHI', 'description', 'Shima Aji', 8.3),
('e15a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/92d7b30b-a15f-499f-8a68-
098cdb65afae.jpg', 'SUSHI', 'description', 'Kyushu Aji', 8.3),
('e16a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/55c35c36-480c-4e29-9be6-
f0dc962f22ed.jpg', 'SUSHI', 'description', 'Renkodai', 8.3),
('e17a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/29fc9606-282b-4bec-9424-
739cee0d1be9.jpg', 'SUSHI', 'description', 'Kinmedai', 9.5),
('e18a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/beb8620c-d105-4c5c-99fd-
083dc082a33d.jpg', 'SUSHI', 'description', 'Hokkaido Uni', 15.0),

('e19a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/8432c23a-535b-4882-80e6-
5a51cbc808e0.jpg', 'SOUP', 'description', 'Oyako Don', 32.0),
('e20a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-
cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-
down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/60aaab53-e092-4b85-b8d3-
5759f25d0a0d.jpg', 'SOUP', 'description', 'Tekka Don', 26.0),
```

```
('e21a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/286d5d5a-5cc8-49d8-82bd-4a91ced933cc.jpg', 'SOUP', 'description', 'Unadon', 34.0),
('e22a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/cb72ac3d-4a53-4f07-a925-dc9fe427abf1.jpg', 'SOUP', 'description', 'Miso Soup', 7.5),
('e23a1618-f42f-4579-b006-e54f02b207ba', 'https://popmenucloud.com/cdn-cgi/image/width%3D1920%2Cheight%3D1920%2Cfit%3Dscale-down%2Cformat%3Dauto%2Cquality%3D60/sdzgybkv/463f9426-8cea-4d0f-a8a8-62254135dec8.jpg', 'SOUP', 'description', 'Spicy Seafood Soup', 19.0);

insert into bags (count, user_id, dish_id) values
('2', 'e03a1618-f42f-4579-b006-e45f02b201ba','e16a1618-f42f-4579-b006-e54f02b207ba'),
('1', 'e03a1618-f42f-4579-b006-e45f02b201ba','e01a1618-f42f-4579-b006-e54f02b207ba'),
('3', 'e03a1618-f42f-4579-b006-e45f02b201ba','e18a1618-f42f-4579-b006-e54f02b207ba');

insert into reviews (id, user_id, review, review_date) values
('341fea43-9cc4-4240-9da7-57d5cc12d2a0', 'e03a1618-f42f-4579-b006-e45f02b201ba','Fabulous. Came here for my boyfriends birthday last week and i haven’t stopped thinking about it since. The chef is so talented and full of personality. Wonderfully executed Omakase with a perfect sake pairing. The fish was super fresh and the uni was amazing! We enjoyed it so much and can’t wait to come back!', '2016-06-22 19:10:25-07'),
('342fea43-9cc4-4240-9da7-57d5cc12d2a0', 'e03a1618-f42f-4579-b006-e45f02b202ba','As visitors from out of town, we actually did not have this restaurant on our list. We were looking for somewhere to eat after Dante and got lucky with a reservation. Pure luck and what a truly amazing experience. The omakase was probably the best sushi I have ever had. Our chef was a truly nice and awesome chef. I couldnt recommend this place more. The only downside is how expensive it is, but the food is so worth it. Check it out!', '2017-06-22 19:10:25-07'),
('343fea43-9cc4-4240-9da7-57d5cc12d2a0', 'e03a1618-f42f-4579-b006-e45f02b203ba','BEST SUSHI I HAVE EVER HAD! Ok let me start off by saying that it truly is fresh quality sushi. My boyfriend took me here recently for my birthday and it was a really fun experience. We sat at the lounge counter and it was a cool seeing how everything is prepared right in front of you. The service was also really good! I recommend this sushi place to anyone who wants fresh quality sushi and an overall fun michelin star restaurant experience!', '2018-06-22 19:10:25-07'),
('344fea43-9cc4-4240-9da7-57d5cc12d2a0', 'e03a1618-f42f-4579-b006-e45f02b204ba','This place was amazing. The best sushi I have ever had by far. The omakase was a work of art and made me so happy. The chef was knowledgeable, friendly, and explained every piece of fish. He was so attentive and passionate about his craft, he made our experience so memorable.The customer service was beyond excellent. We asked for drink recommendations and each drink we had paired perfectly with our sushi. Would recommend that you eat here at least once if you’re in NYC. I will definitely go back next time I’m in NYC.', '2019-06-22 19:10:25-07'),
('345fea43-9cc4-4240-9da7-57d5cc12d2a0', 'e03a1618-f42f-4579-b006-e45f02b205ba','Traditional style sushi that focuses on bringing out the best of ingredients. Sushi was very delicious and was full at the end. Service was perfect and they were very attentive to the details. Pricey but coming once in a while for special occasion will be worth.', '2020-06-22 19:10:25-07'),
('346fea43-9cc4-4240-9da7-57d5cc12d2a0', 'e03a1618-f42f-4579-b006-e45f02b206ba','Went there on a Sunday evening for dinner and was seated in the lounge area with other 4 people (ironically all couples celebrating their anniversary) the course was set for 21 pieces. Reservation is highly recommended as it is very popular restaurant', '2021-06-22 19:10:25-07');
```