Министерство науки и высшего образования РФ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Уральский государственный педагогический университет»
Институт математики, физики, информатики и технологий
Кафедра информационно-коммуникационных технологий в образовании

ИНФОРМАЦИОННАЯ СИСТЕМА ДЛЯ РАБОТЫ С ЭЛЕКТРОННЫМИ КАРТАМИ GOOGLE PAY

Выпускная квалификационная работа
бакалавра по направлению подготовки
09.03.02 – Информационные системы и технологии

Исполнитель: студент группы ИСИТ-1501
Института математики, физики, информатики
и технологий
Прокопченко В.Т.

Руководитель: к.п.н., доцент кафедры ИКТО
Кудрявцев А.В.

Работа допущена к защите
«____»_____ 2019 г.
Зав. кафедрой _____

Екатеринбург – 2019

Реферат

Прокопченко В.Т. ИНФОРМАЦИОННАЯ СИСТЕМА ДЛЯ РАБОТЫ С ЭЛЕКТРОННЫМИ КАРТАМИ GOOGLE PAY, выпускная квалификационная работа: 61 стр., рис. 26, табл. 1, библ. 42 назв.

Ключевые слова: ИНФОРМАЦИОННАЯ СИСТЕМА, ПРОЕКТИРОВАНИЕ ИФНОРМАЦИОННОЙ СИСТЕМЫ, РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ ДЛЯ УПРАВЛЕНИЯ ЭЛЕКТРОННЫМИ КАРТАМИ, ЭЛЕКТРОННЫЕ КАРТЫ, GOOGLE PAY.

Предмет разработки – информационная система для управления электронными картами Google Pay.

Цель работы – разработать информационную систему для управления электронными картами Google Pay, обеспечивающую возможность создания и изменения шаблонов электронных карт, выдачу электронных карт конечному клиенту.

В работе описаны результаты проектирования и программной реализации системы для управления электронными картами Google Pay, обеспечивающую возможность создавать и изменять шаблоны электронных карт Google Pay, выдавать электронные карты Google Pay и изменять их, хранить информацию о выданных картах и их пользователях.

Система состоит из четырех компонентов:

- Панель управления электронными картами,
- Форма выдачи электронных карт,
- Серверное приложение, работающее с клиентами,
- Серверное приложение, работающее с серверами Google.

Все компоненты системы выполнены в текстовом редакторе Visual Studio Code, используются языки PHP и JavaScript. Выполнено с помощью фреймворков

Laravel и Vue.js. Система нигде не внедрена, но может быть использована после регистрации в системе.

Оглавление

Введение

Проблема выдачи пластиковых карт лояльности бизнесом заключается в том, что это долгий и сложный процесс оформления анкеты на бумаге, с последующей отправкой анкеты в офис конкретного бизнеса, с последующим заполнением вручную менеджером электронный формы на создание клиента в базе данных и привязке к нему пластиковой карты. На любом этапе приведённого процесса могут возникнуть огромное количество критичных и мелких ошибок, связанных с утерей анкеты или пластиковой карты, связанных с некорректно заполненными данными менеджером в офисе, и ещё огромное количество проблем.

Электронные карты по сравнению с пластиковыми имеют ряд преимуществ:

1. Просты в установке и управлении.
2. Не имеют возможности быть утерянными.
3. Информация на картах является динамической, она может изменяться исходя из требований заказчика.
4. Легко интегрируются с любой CRM системой с поддержкой API интерфейса.

Создание своей собственной информационной системы управления электронными картами может обойтись крупными затратами для бизнеса.

Поэтому было принято решение создать свою собственную универсальную информационную систему для работы с электронными картами, которая сможет стать бюджетным аналогом существующих систем управления электронными картами, при этом включая все функции систем конкурентов.

Предмет разработки: информационная система на основе языков web-программирования JavaScript, PHP.

Цель разработки: разработать информационную систему для управления электронными картами Google Pay, обеспечивающую возможность создания и

изменения шаблонов электронных карт, выдачу электронных карт конечному клиенту.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Произвести сбор и обработку информации по существующим системам управления электронных карт для выявления обязательных функций системы.
2. В соответствии с техническим заданием провести разработку системы микросервисов для управления электронными картами Google Pay.
3. Подготовить техническую и сопроводительную документацию.

Глава 1. Теоретический анализ востребованности и необходимости разработки системы управления электронными картами

1.1 Теоретические основы создания информационных систем

Рассмотрим классификацию информационных систем по способу организации информационной системы и подробно рассмотрим каждую из них. Как правило, выделяют следующие классы[6]:

- системы на основе архитектуры файл-сервер;
- системы на основе архитектуры клиент-сервер;
- системы на основе микросервисов;

Исторически первыми появились информационные системы с использованием архитектуры файл-сервер (Рис. 1). Эта архитектура только извлекает данные из файла (файлов) базы данных и передает их клиенту для дальнейшей обработки. В процессе работы из базы данных клиенту передаются большие объемы информации. Значительный сетевой трафик иногда особенно сильно сказывается при одновременной работе даже уже нескольких клиентов. В файл-серверной архитектуре всегда передаются избыточные данные. Неважно, сколько записей из базы данных нужны клиенту — файлы базы данных передаются в самом общем случае целиком.

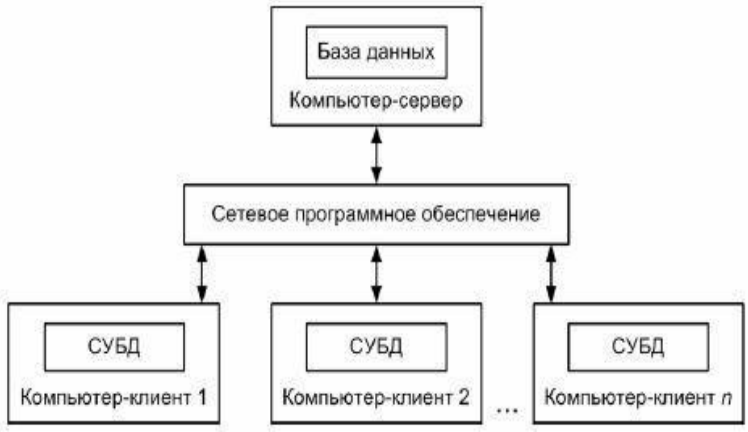

Рис. 1. Структура информационной системы с файл-сервером

Применение архитектуры «файл-сервер» привлекает своей простотой, удобством использования и доступностью. Она представляет интерес для малых рабочих групп, а нередко до сих пор используется и в информационных системах масштаба небольшого предприятия. Информационные системы с клиент-серверной архитектурой позволяют избежать проблем файл-серверных приложений. При такой архитектуре сервер базы данных, расположенный на компьютере-сервере, обеспечивает выполнение основного объема обработки данных. Клиентское приложение формирует запросы к серверу базы данных, как правило, в виде инструкций языка SQL[31]. Сервер извлекает из базы запрошенные данные и передает на компьютер клиента. Главное достоинство такого подхода — значительно меньший объем передаваемых данных.

Большинство конфигураций информационных систем типа «клиент-сервер» использует двухуровневую модель, в которой клиент обращается к серверу (Рис. 2).

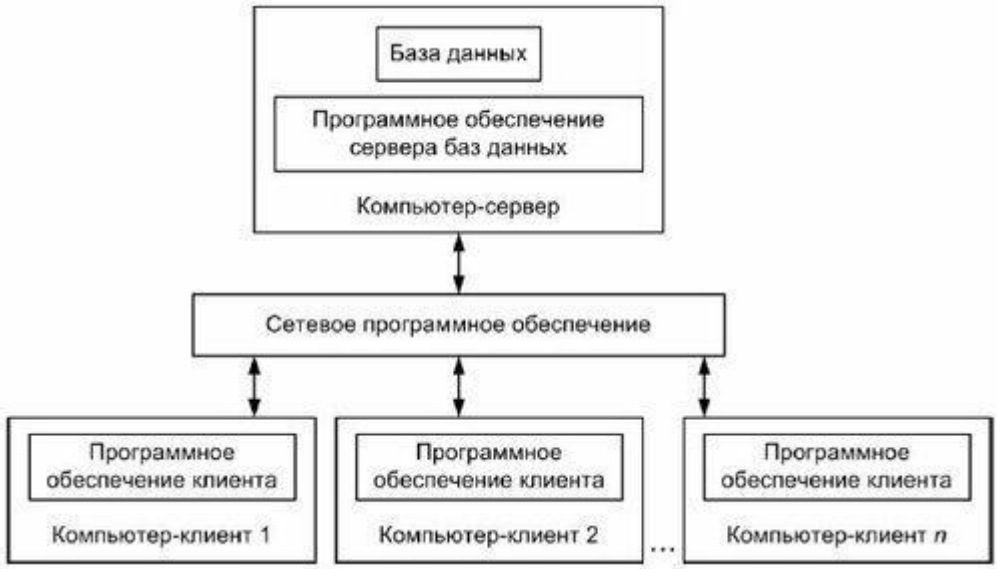

Рис. 2. Структура информационной системы с сервером базы данных

Обеспечение безопасности данных — очень важная функция для успешной работы информационной системы. Если у базы данных слабая система безопасности, любой достаточно подготовленный пользователь может нанести серьезный ущерб работе предприятия. Следует отметить, что защита данных в

файл-серверной информационной системе изначально не может быть обеспечена на должном уровне. Безопасность же современных серверов баз данных, организованная в нескольких направлениях: с помощью самой операционной системы; с использованием схем, имен входов, ролей, шифрования базы данных и т. д.;[38] путем ограничения доступа пользователей через представления, заслуживает похвалы. В настоящее время архитектура «клиент-сервер» широко признана и находит применение для организации работы приложений, как для рабочих групп, так и для информационных систем масштаба предприятия.

Микросервисная архитектура — вариант архитектуры информационной системы, ориентированный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей — микросервисов, получивший распространение в середине 2010-х годов в связи с развитием практик гибкой разработки и DevOps. Если в традиционных вариантах сервис-ориентированной архитектуры модули могут быть сами по себе достаточно сложными программными системами, а взаимодействие между ними зачастую полагается на стандартизованные тяжеловесные протоколы (такие, как SOAP, XML-RPC), в микросервисной архитектуре системы выстраиваются из компонентов, выполняющих относительно элементарные функции, и взаимодействующие с использованием экономичных сетевых коммуникационных протоколов (с использованием JSON, Protocol Buffers, Thrift). За счёт повышения гранулярности модулей архитектура нацелена на уменьшение степени зацепления и увеличение связности, что позволяет проще добавлять и изменять функции в системе в любое время.

Микросервисы — это небольшие, автономные, совместно работающие сервисы. Разберем это определение по частям и рассмотрим, что определяет отличительные черты микросервисов. Микросервисы — небольшие и нацеленные на то, чтобы хорошо справляться только с одной работой. При

создании кода, дополнительных свойств программы разрастается и база программного кода. Со временем из-за слишком большого объема этой базы возникают затруднения при поиске тех мест, куда нужно вносить изменения. Несмотря на стремление к созданию понятных модульных монолитных баз кода, довольно часто эти произвольные, находящиеся в процессе становления границы нарушаются. Код, относящийся к схожим функциям, попадает в разные места, что усложняет устранение дефектов или реализацию функций. Внутри монолитных систем разработчики стремятся бороться с этой тенденцией, пытаясь сделать свой код более связанным, зачастую путем создания абстракций или модулей. Придание связности означает стремление сгруппировать родственный код. Эта концепция приобретает особую важность при размышлении о микросервисах. Она усиливается определением, данным Робертом С. Мартином (Robert C. Martin)[37] принципу единственной обязанности — Single Responsibility Principle, которое гласит: «Собирайте вместе все, что изменяется по одной и той же причине, и разделяйте все, что изменяется по разным причинам». Точно такой же подход в сфере микросервисов используется в отношении независимых сервисов. Границы сервисов формируются на основе бизнес-границ, что позволяет со всей очевидностью определить местонахождение кода для заданной области выполняемых функций. Удерживая сервис в четко обозначенных границах, разработчики не позволяют себе мириться с его чрезмерным разрастанием, со всеми вытекающими из этого трудностями. Нужно также считаться с тем фактом, что разработчики могут быть втянуты в ряд зависимостей, которые сами по себе содержат множество строк кода. Кроме того, некоторые составляющие вашей области деятельности могут быть сложными по определению и требовать немалого объема кода. Джон Ивс (Jon Eaves) из Австралии на сайте RealEstate.com.au охарактеризовал микросервисы как некий код, который может быть переписан за две недели, что имело вполне

определенный смысл в качестве основного правила конкретно в его контексте. Когда решается вопрос о достаточности уменьшения объема кода, программисты предпочитают размышлять в следующем ключе: чем меньше сервис, тем больше проявляются все преимущества и недостатки микросервисной архитектуры. Чем меньше делается сервис, тем больше становятся его преимущества в смысле взаимозависимости. Но верно и то, что возникают некоторые осложнения из-за наличия все большего количества движущихся частей. Микросервисы должны быть автономные.[29] Обмен данными между самими сервисами ведется через сетевые вызовы, чтобы упрочить обособленность сервисов и избежать рисков, сопряженных с тесными связями. Сервисам необходимо иметь возможность изменяться независимо друг от друга и развертываться, не требуя никаких изменений от потребителей. При создании системы микросервисов нужно подумать о том, что именно сервисы будут показывать и что им можно будет скрывать. Если объем совместно используемого будет слишком велик, потребляемые сервисы станут завязываться на внутренние представления. Это снизит автономность, поскольку при внесении изменений потребует дополнительного согласования с потребителями. Микросервисы выставляют напоказ программный интерфейс приложения — Application Programming Interface (API), и другие микросервисы связываются через эти API. Микросервисы обладают множеством разнообразных преимуществ. Многие из них могут быть присущи любой распределенной системе. Но микросервисы нацелены на достижение вершин этих преимуществ, что обусловливается в первую очередь тем, насколько глубоко ими принимаются концепции, положенные в основу распределенных систем и сервис-ориентированной архитектуры. Микросервисы могут использовать внутри различные технологии. Это позволит выбрать для каждого задания наиболее подходящий инструментарий, не выискивая какой-либо стандартный подход на все случаи жизни, зачастую заканчивающийся выбором

наименьшего общего знаменателя. Если какой-то части системы требуется более высокая производительность, можно принять решение по использованию иного набора технологий, более подходящего для достижения требуемого уровня производительности.

Микросервисы могут иметь различные способы хранения данных для разных частей нашей системы. Например, пользовательский обмен сообщениями в социальной сети можно хранить в графоориентированной базе данных, отражая тем самым присущую социальному графу высокую степень взаимосвязанности, а записи в блоге, создаваемые пользователями, можно хранить в документно-ориентированном хранилище данных, давая тем самым повод для использования разнородной архитектуры, похожей на ту, что показана на (Рис. 1). Микросервисы также позволяют быстрее внедрять технологии и разбираться в том, чем именно нововведения могут помочь в решении конкретных задач. Одним из величайших барьеров на пути принятия новой технологии является риск, связанный с ее использованием. Если при работе с монолитным приложением разработчики могут использовать новые язык программирования, базу данных или структуру, то влияние распространится на существенную часть системы. Когда система состоит из нескольких сервисов, появляются несколько новых мест, где можно проверить работу. Разработчик может выбрать такой сервис, с которым риск будет наименьшим, и воспользоваться предлагаемой им технологией, зная, что может ограничить любое потенциально отрицательное воздействие. Возможность более быстрого внедрения новых технологий рассматривается многими организациями как существенное преимущество. Микросервисы позволяют упростить использование разнообразных технологий. Разумеется, использование нескольких технологий не обходится без определенных издержек.[8] Некоторые организации предпочитают накладывать на выбор языков ограничения. В Netflix и Twitter, к примеру, в качестве платформы используется преимущественно Java Virtual

Machine (JVM), поскольку они очень ценят надежность и производительность этой системы. В этих организациях также разрабатываются библиотеки и инструментальные средства для JVM, существенно упрощающие работу в масштабе платформы, но сильно усложняющие ее для сервисов и клиентов, не работающих на Java-платформе. Но для всех заданий ни в Twitter, ни в Netflix не используется лишь один набор технологий. Еще одним аргументом в противовес опасениям по поводу смешения различных технологий является размер. Микросервисы должны быть устойчивыми. Ключевым понятием в технике устойчивости является перегородка. При отказе одного компонента системы, не вызывающем череду связанных с ним отказов, проблему можно изолировать, сохранив работоспособность всей остальной системы. Именно такими перегородками для вас станут границы сервисов. В монолитном сервисе при его отказе прекращается вся работа. Работая с монолитной системой, можно уменьшить вероятность отказа, запустив систему сразу на нескольких машинах, но, работая с микросервисами, разработчики могут создавать такие системы, которые способны справиться с тотальными отказами сервисов и снизить соответствующим образом уровень их функциональных возможностей. Чтобы убедиться в том, что системы, составленные из микросервисов, могут воспользоваться улучшенной устойчивостью должным образом, нужно разобраться с новыми источниками отказов, с которыми должны справляться распределенные системы. Отказать могут как сети, так и машины, и такие отказы неизбежны. Нужно знать, как с этим справиться и как это может (или не может) повлиять на конечного пользователя программного средства.

Микросервисы отличаются своей масштабируемостью. В больших монолитных сервисах расширять приходится все сразу. Одна небольшая часть всей системы может иметь ограниченную производительность, но, если в силу этого тормозится работа всего огромного монолитного приложения, расширять нужно все как единое целое. При работе с небольшими сервисами можно

расширить только те из них, которые в этом нуждаются, что позволяет запускать другие части системы на небольшом, менее мощном оборудовании.

1.2 Выбор метода разработки

Разрабатываемая автоматизированная информационная система должна включать в себя четыре связанных между собой элемента:

1. Панель управления электронными картами.
2. Форма выдачи электронных карт
3. Серверное приложение, способное по средствам API выдавать изменять и удалять данные о тратах бизнеса внутри системы, о клиентах бизнеса.
4. Серверное приложение, способное создавать и изменять шаблоны электронных карт и персональные электронные карт внутри системы Google Pay.

Исходя из этих требований для разработки системы была выбрана микросервисная архитектура ИС, потому что целесообразно разделить все четыре компонента системы на микросервисы, для дальнейшего масштабирования и поддержки системы электронных карт.

Сравнение Dekstop и Web приложения[6]:

	Desktop приложение	**Web приложение**
Доступ к сети Internet	Не требуется.	Необходим.
Установка/обновление	Должно быть развёрнуто или установлено.	Единовременная настройка. Одна установка для всех пользователей. Благодаря централизованности моментально обновление.
Интерфейс взаимодействия	Стандартные интерфейсы, стандартное взаимодействие	Разнообразный интерфейс взаимодействия.
Совместимость с устройствами	Зависимость от платформы. Исключение — кроссплатформенные приложения.	Платформо-независимое.
Анимация, графика	Быстрая, быстрый отклик	Относительно медленный

		отклик, связанный с передачей данных по сети.
Шрифты	Присутствуют только те шрифты, которые установлены у пользователя	Любые шрифты — есть возможность подгрузки необходимого шрифта через Internet
Разработка	Под каждую платформу есть свои инструменты, зачастую под каждую платформу приходиться писать свою версию.	Всё выполняется на сервере. Кроссплаформенно.

Исходя из приведённой таблицы было принято решение разработать систему в рамках Web приложения, потому что система должна постоянно развиваться, быть максимально доступной, за счёт кроссплатформенности Web приложений, быть легко распространяемой за счёт отсутствия этапа установки системы на локальный компьютер и иметь постоянный доступ к сети Интернет.

При выборе разработки Web приложения возникает потребность выбора языков программирования из очень большого списка доступных языков. Поскольку Web приложение имеет серверную часть и клиентскую требуется выбрать для каждой части отдельный язык. Доступные на сегодняшний день серверные языки программирования:

1. PHP
2. Perl
3. Python
4. Ruby
5. ASP.NET
6. Java
7. Groovy

Первый по своей популярности язык PHP это скриптовый язык общего назначения, поддерживается подавляющим большинством хостинг-провайдеров. По статистике сайта w3techs.com на 2019 год в Web проектах,

среди серверных языков PHP удерживает первую позицию с огромным отрывом (78,9%) по популярности среди разработчиков. Такую популярность PHP получил за счёт наличия большого набора встроенных средств и дополнительных модулей. Приведём некоторые из них:

- автоматическое извлечение POST и GET-параметров;
- взаимодействие с большим количеством различных систем управления базами данных;
- автоматизированная отправка HTTP-заголовков;
- работа с HTTP-авторизацией;
- работа с cookies и сессиями;
- работа с локальными и удалёнными файлами, сокетами;
- обработка файлов, загружаемых на сервер;
- работа с XForms.

Выбор серверного языка пал на PHP, исходя из приведённых выше аргументов.

Языки клиентской части Web приложения[42]:

1. JavaScript
2. Flash
3. Silverlight

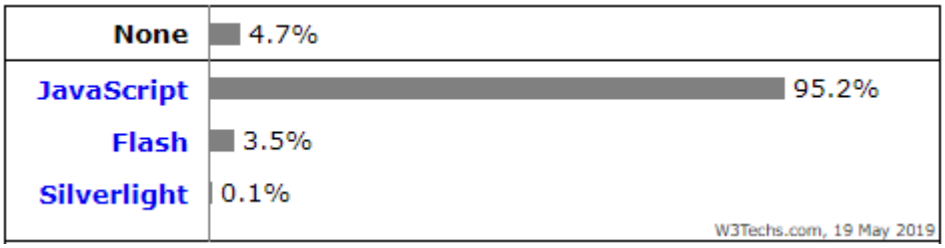

Рис. 3. Популярность языков по W3Techs.com

JavaScript - это полноценный динамический язык программирования, который применяется к HTML документу, и может обеспечить динамическую интерактивность на веб-сайтах. JavaScript невероятно универсален. Обладая большим опытом, разработчик имеет возможность создавать игры,

анимированную 2D и 3D графику, полномасштабные приложения с базами данных и другое. JavaScript сам по себе довольно компактный, но очень гибкий. Разработчиками написано большое количество инструментов поверх основного языка JavaScript, которые разблокируют огромное количество дополнительных функций с очень небольшим усилием. К ним относятся:

- Программные интерфейсы приложения (API), встроенные в браузеры, обеспечивающие различные функциональные возможности, такие как динамическое создание HTML и установку CSS стилей, захват и манипуляция видеопотоком, работа с веб-камерой пользователя или генерация 3D графики и аудио сэмплов.
- Сторонние API позволяют разработчикам внедрять функциональность в свои сайты от других разработчиков, таких как Twitter или Facebook.
- Также возможно применить к HTML сторонние фреймворки и библиотеки, что позволяет ускорить создание сайтов и приложений.

JavaScript так же как и PHP среди серверных языков является неоспоримым лидером среди языков фронтенда (95, 1%)(Рис. 3) по своей популярности. JavaScript является встраиваемым в HTML код веб-страницы, имеет огромное количество фреймворков и превосходную документацию на любом распространенном разговорном языке.

ActionScript — объектно-ориентированный язык программирования, один из диалектов ECMAScript, который добавляет интерактивность, обработку данных и многое другое в содержимое Flash-приложений. ActionScript исполняется виртуальной машиной, которая является составной частью Flash Player.

Исходя из приведённых выше аргументов, выбор языка программирования клиентской части системы пал на JavaScript.

Информационная система управления электронных карт обещает быть технически сложным и нагруженным проектом, поэтому было принято

решение использовать фреймворки для достижения высокой скорости разработки системы.

Фреймворки это программные продукты, включающие в себя некоторые базовые модули системы. Во фреймворке заложен условные каркас, основа системы, который требуется развивать разработчику, не нарушая при этом правил, которые диктует фреймворк в своей официальной документации.

Главное преимущество использования фреймворков при создании web приложения это то, что web приложения зачастую используют одну стандартизированную структуру организации системы классов, а фреймворк уже содержит эту структуру, и не нужно реализовывать её в каждом новом проекте.

Главный минус фреймворков заключаются в том, что они бывают слишком универсальны, фреймворки рассчитаны захватить как можно больше проектов, каждый фреймворк хочет попасть в как можно большее количество систем, поэтому разработчики фреймворков внедряют огромное количество классов, которые зачастую не нужны конечной системе. Часто встречается следующая ситуация, к системе, разработанной на фреймворке, приходит запрос, говоря человеческким языком: “Дай клиента с id = 28”, тогда в системе кроме нужных классов для работы с базой данных так же подключаются большое количество не нужных классов.

Фреймворки постоянно обновляются, в них добавляются новые функции, улучшаются старые, вероятность ошибиться во время разработки проекта у разработчика, использующего фреймворк, значительно уменьшается, следовательно, не приходится тратить огромное количество времени на поиск и решение ошибок в каркасе проекта.

Выбранный язык для серверной стороны системы является PHP, есть несколько фреймворков для упрощения создания проекта на PHP:

1. Laravel

2. Slim
3. Yii

Laravel предназначен для разработки веб приложений как личных так и корпоративных, он является самым популярным среди бекенд разработчиков, по версии многих сайтов опросников[11]. Возможности Laravel:

- Пакеты — позволяют создавать и подключать модули в формате Composer к системе на Laravel.
- Eloquent ORM — реализация шаблона проектирования ActiveRecord на PHP. Позволяет строго определить отношения между объектами базы данных.
- Логика приложения — часть разрабатываемого приложения, объявленная либо при помощи контроллеров, либо маршрутов (функций-замыканий).
- Обратная маршрутизация связывает между собой генерируемые приложением ссылки и маршруты, позволяя изменять последние с автоматическим обновлением связанных ссылок. При создании ссылок с помощью именованных маршрутов Laravel автоматически генерирует конечные URL.
- REST-контроллеры — дополнительный слой для разделения логики обработки GET- и POST-запросов HTTP.
- Автозагрузка классов — механизм автоматической загрузки классов PHP без необходимости подключать файлы их определений в include.
- Миграции — система управления версиями для баз данных. Позволяет связывать изменения в коде приложения с изменениями, которые требуется внести в структуру БД, что упрощает развёртывание и обновление приложения.
- Страничный вывод — упрощает генерацию страниц, заменяя различные способы решения этой задачи единым механизмом, встроенным в Laravel.

Slim — это микроструктура PHP, которая помогает быстро писать простые, но мощные веб-приложения и API. По сути, Slim - диспетчер, который

получает HTTP-запрос, вызывает соответствующую процедуру обратного вызова и возвращает HTTP-ответ. Плюсы Slim:

- Ничего лишнего
- Низкий порог входа
- Доступная документация

Yii — фреймворк похожий на Laravel, имеет схожие возможности, основное различие заключается в том, что в Yii слишком склеены серверные части и клиентские, поэтому использование этого фреймворка в настоящее время, когда стало модно даже переносить клиентскую часть приложения и серверную на разные серверы, под сомнением[36].

Среди этих фреймворков для реализации серверной части системы управления электронных карт был выбран фреймоврк Laravel, потому что в нём есть всё необходимое, для корректной работы системы. Системе потребуются ключевые функции, которые являются встроенными в Laravel и их реализация не составит труда:

- Авторизация
- Постраничный вывод
- Работа с базой данных

Компоненты системы панель управления электронными картами и форма выдачи являются JavaScript проектами, это лицевая сторона информационной системы, поэтому чтобы избежать огромной массы ошибок на стадии разработки проектов было решено использовать фреймворк.

Для JavaScript есть огромное количество разнообразных фреймворков, мы рассмотрим только самые популярные и имеющие отношение к фронтенд разработке:

- React
- Vue
- Angular

React — это эффективная и гибкая декларативная библиотека JavaScript для сборки интерфейса клиентской части системы от команды Facebook. React упрощает сборку объектно-центричных приложений. Основным преимуществом React является долговечность построенных на нём интерфейсах, благодаря высокой обратной совместимости.

Vue.js — это прогрессивный фреймворк для создания пользовательских интерфейсов[9]. В отличие от фреймворков-монолитов, Vue создан пригодным для постепенного внедрения. Его ядро в первую очередь решает задачи уровня представления, что упрощает интеграцию с другими библиотеками и существующими проектами. С другой стороны, Vue полностью подходит и для создания сложных одностраничных приложений, если использовать его совместно с современными инструментами и дополнительными библиотеками.

Angular — JavaScript-фреймворк с открытым исходным кодом. Предназначен для разработки одностраничных приложений. Его цель — расширение браузерных приложений на основе MVC-шаблона, а также упрощение тестирования и разработки. Фреймворк работает с HTML, содержащим дополнительные пользовательские атрибуты, которые описываются директивами, и связывает ввод или вывод области страницы с моделью, представляющей собой обычные переменные JavaScript. Значения этих переменных задаются вручную или извлекаются из статических или динамических JSON-данных.

1.3 Формализованное описание технического задания

1. Общие сведения

1.1.Название организации заказчика

Федеральное государственное бюджетное образовательное учреждение высшего образования «Уральский государственный педагогический университет» Институт математики, физики, информатики и технологий Кафедра информационно-коммуникационных технологий в образовании.

1.2.Название продукта разработки

Информационная система для работы с картами Google Pay.

1.3.Назначение продукта

Организация настройки шаблона электронной карты и организация выдачи электронной карты конечным клиентам.

1.4.Категория пользователей

Менеджеры по работе с клиентами, работающие в конкретном бизнесе.

1.5.Плановые сроки выполнения

Начало работы: 01.09.2018

Конец работы: 10.05.2019

2. Характеристика области применения

2.1.Процессы и структуры, в которых предполагается использование информационной системы.

Организации, с потребностью внедрить карты лояльности.

2.2.Характеристика персонала

- администратор (должен владеть PHP, HTML, JS, Vue.js работать с серверами и базами данных при необходимости)
- менеджер
- пользователь

3. Требование к продукту разработки

3.1. Требования к продукту в целом

Информационная система должна хранить данные о клиентах предприятия, шаблоны электронных карт, данные о транзакциях, проведённых с клиентами, а именно начисление или списание бонусных баллов. Так же нужно сформировывать отчеты о тратах предприятия внутри системы, чтобы дать возможность менеджерам предприятия вести отчеты о расходах на содержание электронных карт. База данных должна осуществлять добавление, изменение, поиск, сортировку и удаление данных, а также просматривать эти данные.

Данные о клиентах предприятия должны включать следующее: ФИО, баланс в режиме реального времени, транзакции осуществленные конечным клиентом, и все возможные данные, которые хранятся в CRM системе предприятия.

Данные о тратах предприятия должны содержать: дату осуществления транзакции, вид транзакции.

Информационная система должна включать функции создания и изменения шаблона электронной карты, выдачу персональных карт. Настройки системы должны быть реализованы с интуитивно понятным интерфейсом, чтобы любой пользователь мог без знания PHP, JS, VueJS, SQL и т.п. выполнить настройку шаблона электронной карты.

3.2. Аппаратные требования

Персональный компьютер с выходом в интернет, оперативная память не менее 2Гб.

3.3. Программное обеспечение

3.3.1. Программное обеспечение для разработки

Браузер, серверная база, кроссплатформенная система, текстовый редактор.

3.3.2. Программное обеспечение для эксплуатации

Браузер, кроссплатформенная система.

3.4. Форматы входных и выходных данных. Источники данных, порядок их ввода в систему

В систему поступает текстовый запрос, введенный пользователем с клавиатур, после чего данный запрос обрабатывается и, в случае успешного выполнения запроса происходит взаимодействие с базой данных, при нажатии на необходимые кнопки запись добавляется в базу в необходимую таблицу.

3.5. Порядок взаимодействия с другими системами

В систему могут поступать HTTP POST запросы об изменении баланса конечного клиента. Один из заголовков запроса обязательно должен содержать авторизацию и ключ авторизации. По этому ключу система распознает к какому именно шаблону электронных карт нужно произвести изменения.

3.6. Меры защиты информации

Пароли шифруются средствами Laravel, двустороннее шифрование, алгоритм AES-256, так, что даже разработчик не может узнать пароль конкретного пользователя, после его регистрации.

4. Требования к пользовательскому интерфейсу

4.1. Общая характеристика пользовательского интерфейса

Форма выдачи электронной карты должна включать в себя минимум полей заполнения, чтобы конечный клиент, для которого будет выпущена электронная карта, не утомился заполнять их, для получения персональной карты.

На странице авторизации в системе есть три пункта:

- Вход
- Регистрация
- Восстановление пароля

При нажатии на любой из этих пунктов появляются соответствующие формы с текстовыми полями. На форме входа необходимо ввести e-mail и пароль, на текстовом поле пароль есть иконка отображения введенного пароля, для возможности посмотреть пароль. После нажатия кнопки войти производится попытка авторизации и пользователь попадает на соответствующую доступу страницу. На форме регистрации необходимо ввести имя, фамилию, e-mail, телефон, пароль и поле подтверждения пароля. Также на форме находится кнопка регистрации, после нажатия на неё производится попытка регистрации в системе, в удачном случае пользователь будет зарегистрирован и сразу же авторизован. В форме восстановлении пароля необходимо ввести e-mail, введенный ранее при регистрации, после нажатия на кнопку «Отправить ссылку на сброс пароля», проверяется есть ли пользователь с введенной электронной почтой в системе, если есть то на указанный e-mail отправляется письмо содержащее ссылку для сброса пароля.

Все страницы приложения не входящих в список страниц авторизации имеют боковое меню слева, в котором находятся необходимые пользователю кнопки управления, в верхнем правом углу находится выпадающий список с выбором доступных функций, выбрать кабинет, создать новый кабинет. Также в правом верхнем углу находится кнопка выхода из приложения.

После входа в систему пользователь может попасть на две возможных страницы, страница выбора кабинета, страница регистрации кабинета, или на страницу управления кабинетом, логика перехода после авторизации находится на сервере и не имеет отношения к интерфейсу.

Страница создания нового кабинета содержит поле имя домена ИНН и промокод, кнопку «Регистрация», после нажатия на эту кнопку

производится попытка регистрации кабинета, в случае успеха пользователь попадает на страницу управления определенным кабинетом.

На странице выбора кабинета пользователю предоставляется таблица с именами кабинета, к которым у пользователя есть доступ, и с соответствующим балансом кабинета. После нажатия на имя кабинета пользователь попадает на страницу управления определенным кабинетом. Также на странице реализован поиск по имени кабинета, требуется пользователям, имеющим более 20 кабинетов, для удобства пользования системой.

На главной странице управления кабинетом находится два информационных виджета, количество выданных карт и баланс, содержащих ссылки на соответствующие страницы. В боковом меню на этом этапе имеется три кнопки, осуществляющие переходы на соответствующие страницы.

Вкладка «Менеджеры» содержит в себе информацию о пользователях входящих в выбранный ранее кабинет. На этой странице есть возможность добавить пользователей, им будет выслано письмо с ссылкой регистрации, и отредактировать конкретного пользователя, нажав на значок карандаша в правом столбце таблицы. После нажатия пользователь переходит на страницу редактирования менеджера, там он может отредактировать имя или фамилию менеджера, и номер телефона, также есть возможность изменить типы получаемых менеджером уведомлений. И таблица приглашённые, в которой содержится информация о приглашенных в выбранный кабинет, но не зарегистрированных в системе, пользователях.

Вкладка «Клиенты» содержит таблицу с данными о владельцах выданных электронных карт. В таблице реализован поиск по e-mail, имени фамилии, или по номеру телефона клиента, также реализована сортировка по дате регистрации, имени, телефону. После нажатия на имя определенного клиента пользователь попадает на страницу редактирования клиента, на этой

странице есть возможность отредактировать ФИО клиента, день рождения, e-mail, телефон, баланс, бонус или скидку, также на этой странице есть QR-код содержащий ссылку на скачивание выданной карты клиента, её номер. Все изменения на этой странице вступят в силу после нажатия на кнопку «Сохранить».

Страница «Карты» является ключевой страницей приложения, через эту страницу осуществляется создание и редактирование шаблона электронной карты. Страница делится на две части, на одной из них показано как примерно будет выглядеть электронная карта, на второй расположены вкладки доступных настроек:

Логотип

На этой вкладке пользователь может настроить картинку, логотип, которая будет располагаться на карте. Для этой цели пользователю предоставлено текстовое поле, в которое он должен вставить URL картинки. Также присутствуют рекомендации для картинки.

2) Изображение

Настройка изображения включает в себя текстовое поле для вставки URL картинки, которая будет размещаться на карте, это основное изображение карты.

3) Цвета

Можно настроить цвет фона карты и цвет шрифта. Для этих целей внедрена палитра цветов, на ней можно выбрать нужный цвет и оттенок, после выбора цвет сразу же применяется на превью карты.

4) Тексты

Настройка текстовых полей включает в себя четыре текстовых поля для ввода информации, которая должна отобразиться на карте. При вводе текст реактивно меняется на превью карты.

4.2.Размещение информации на экране. Макет дизайна экрана

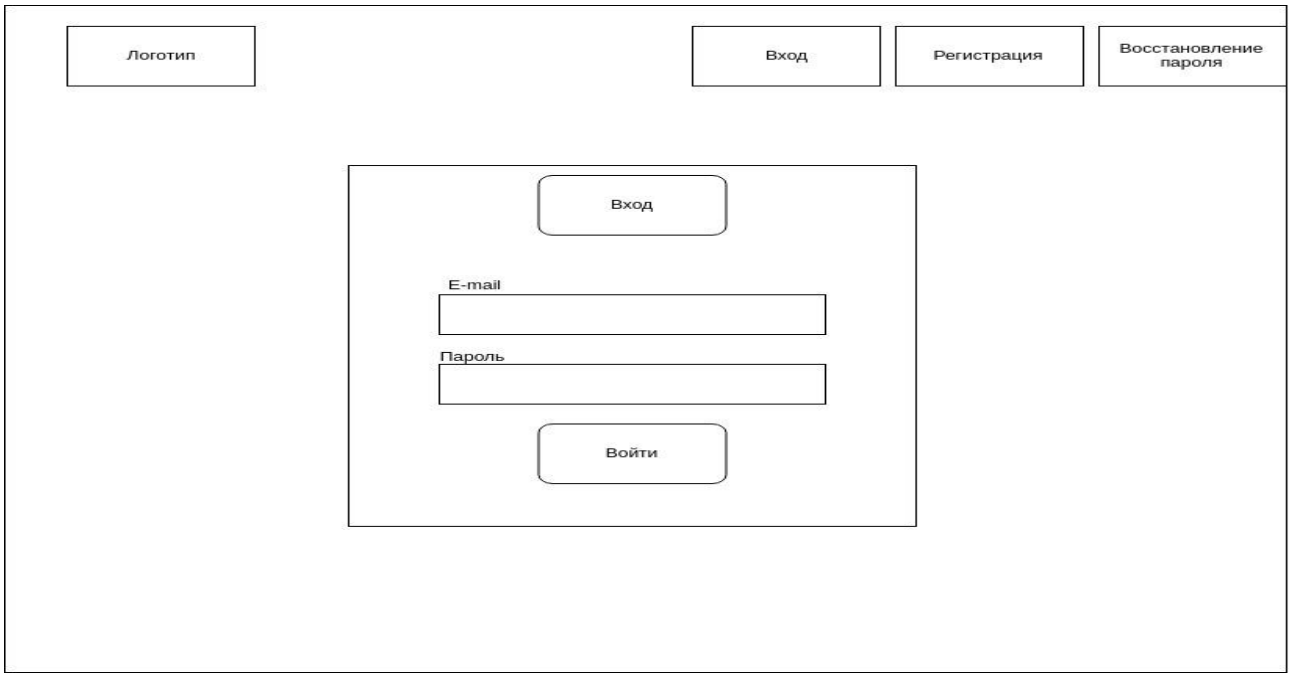

Рис. 4 Макет страницы аутентификации

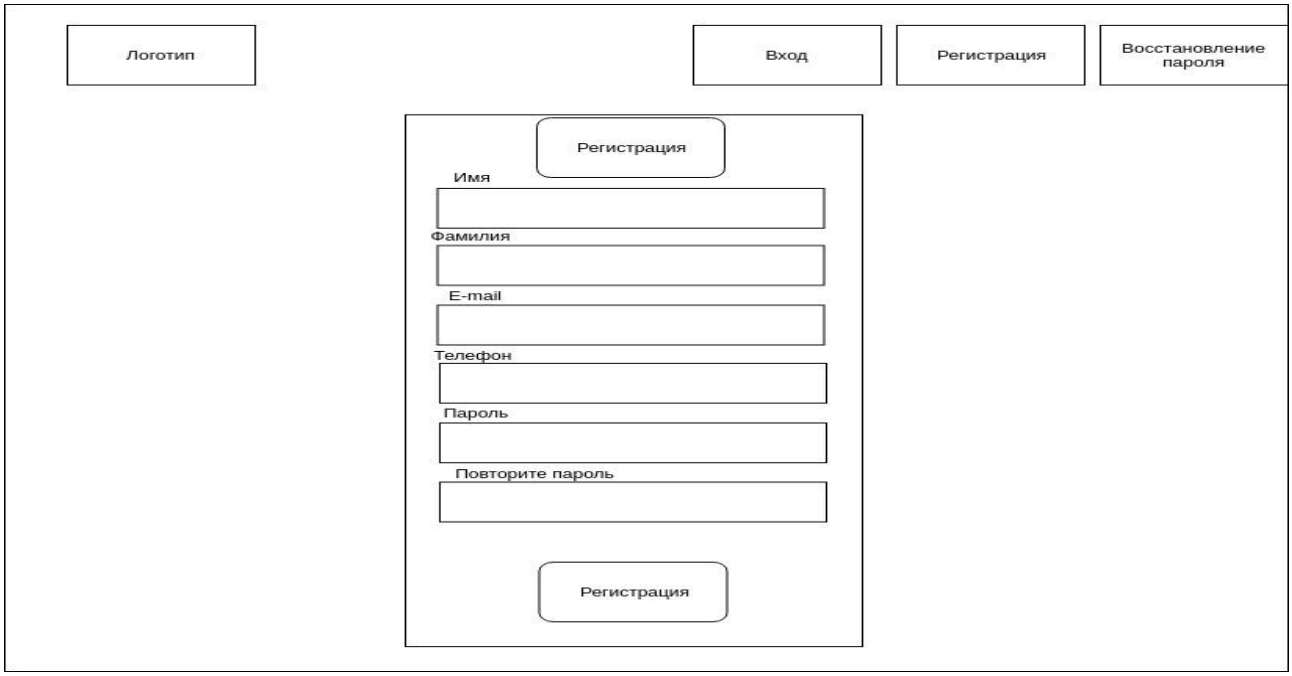

Рис. 5 Макет страницы с регистрацией

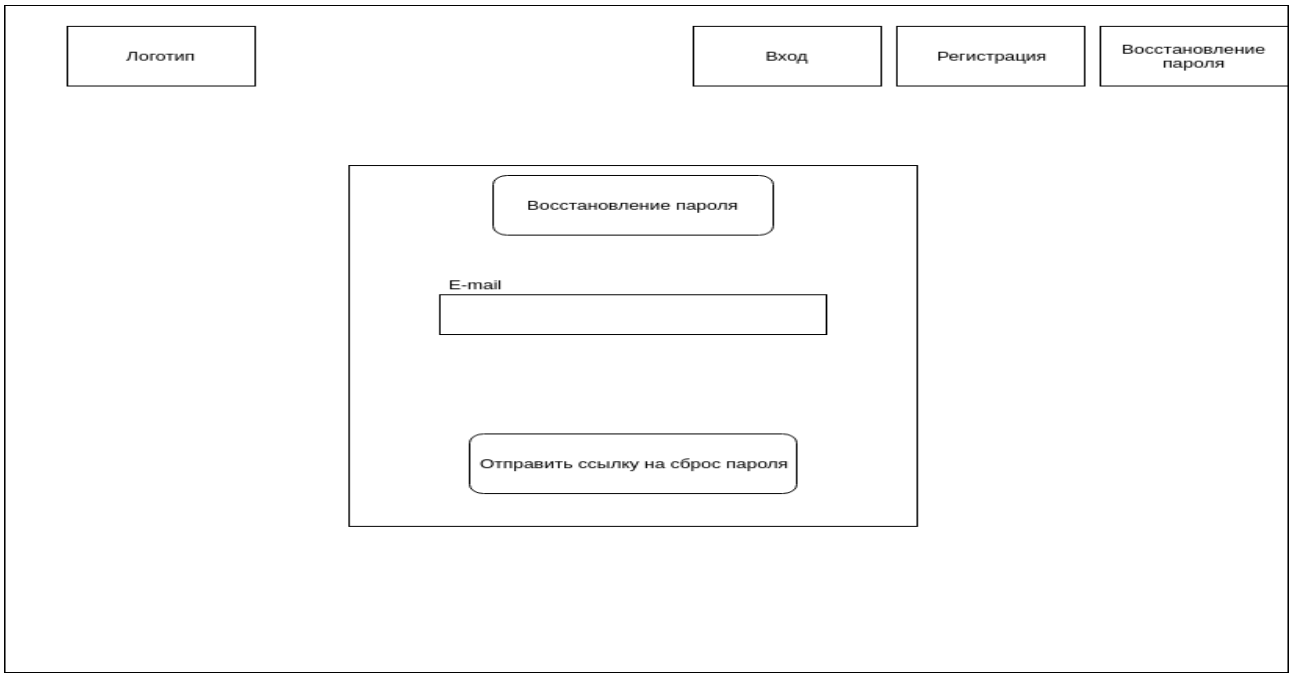

Рис. 6 Макет страницы с восстановлением пароля

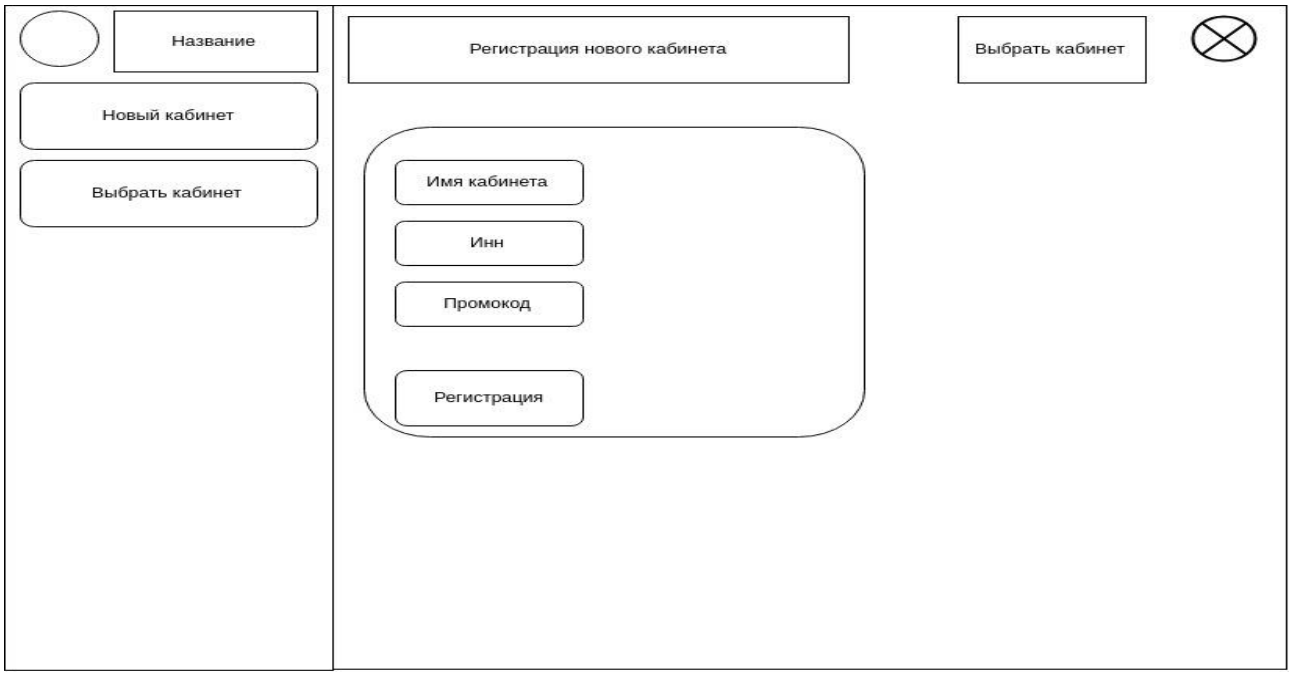

Рис. 7 Макет страницы с регистрацией нового кабинета

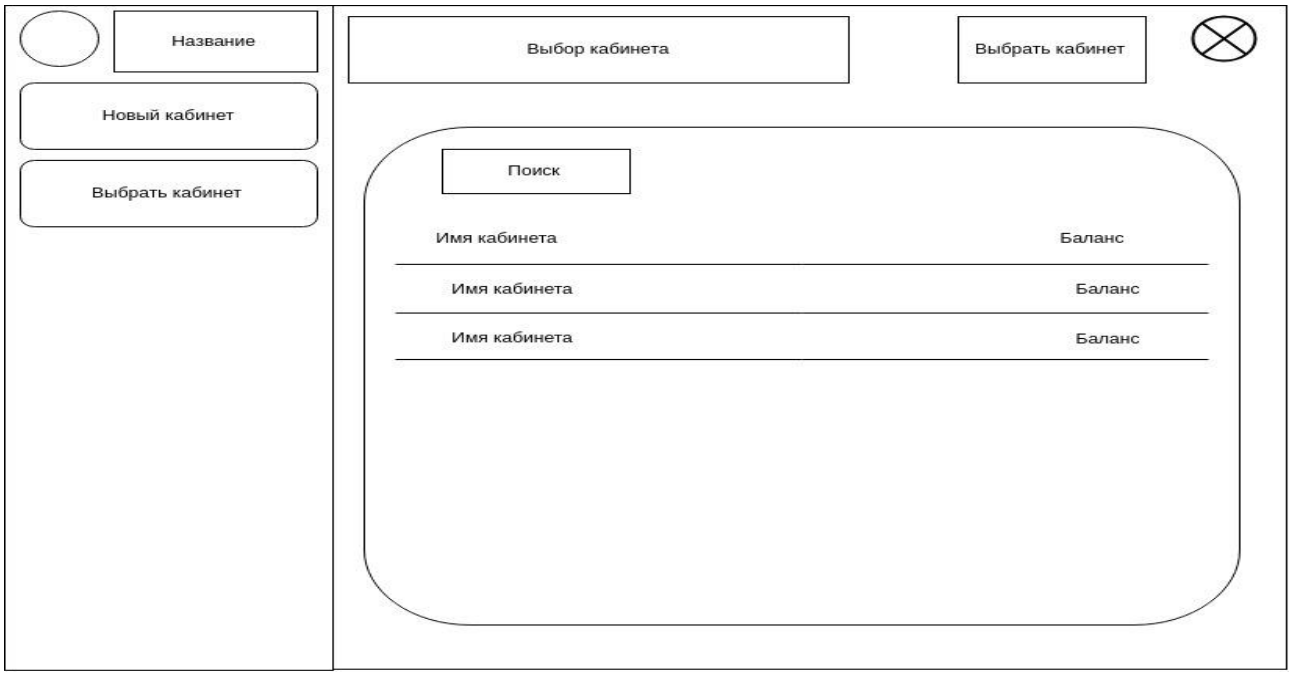

Рис. 8 Макет страницы с доступными кабинетами

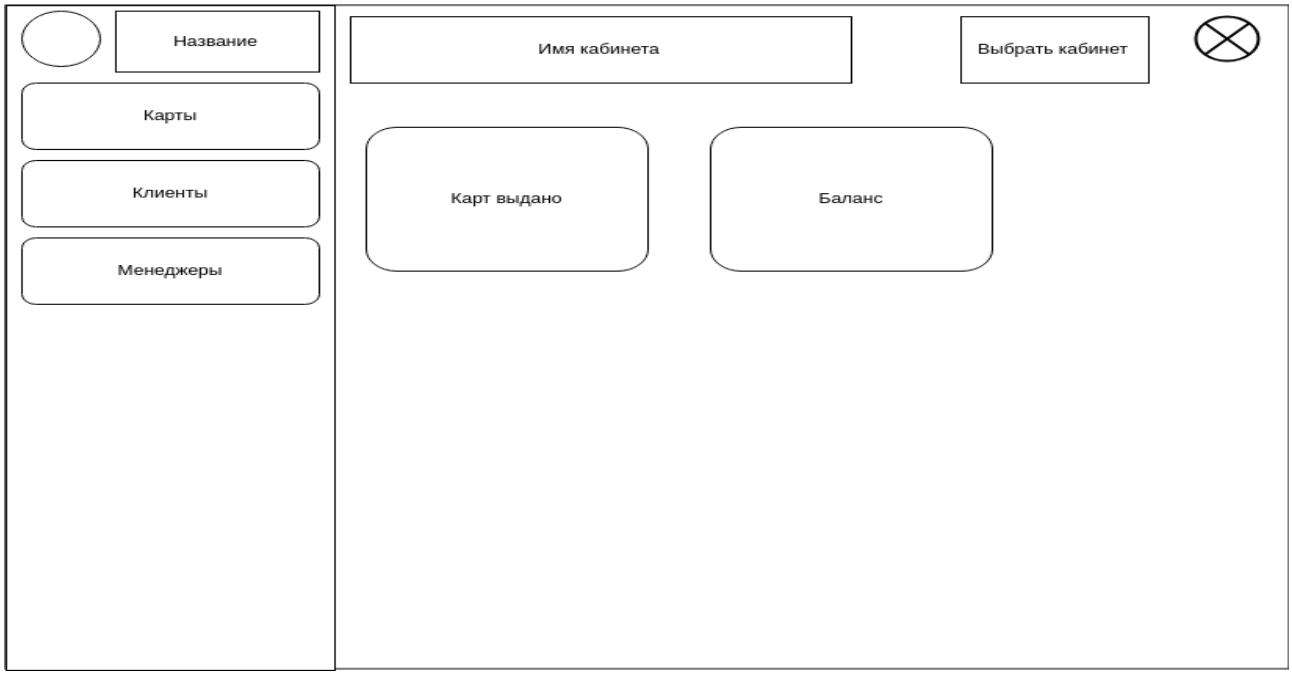

Рис. 9 Макет страницы с главной страницей выбранного кабинета

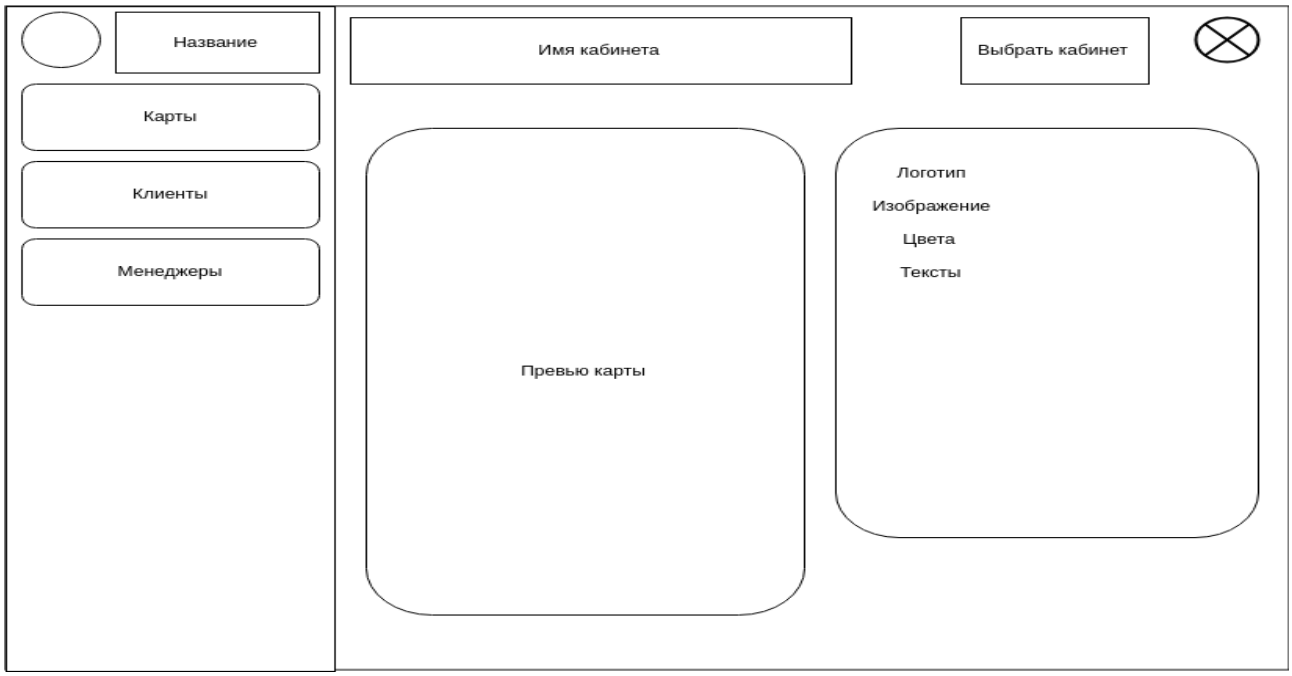

Рис. 10 Макет страницы с редактированием шаблона карты

Рис. 11 Макет страницы с таблицей владельцев карт

5. Требования к документированию

5.1. Перечень сопроводительной документации

Техническое задание – главный документ, сопровождающий генератор БРС и отражающий все характеристики и требования к разрабатываемой системе.

5.2. Требования к содержанию отдельных документов

Техническое задание исходный документ для проектируемого объекта, содержащий технические требования, предъявляемые к объекту и исходные данные для его разработки; в документе указывается информация о назначении объекта, области его применения и сроках выполнения.

6. Порядок сдачи-приема базы данных

   В соответствии со сроками выполнения ВКР.

Глава 2. Разработка программного комплекса информационной системы

1.1 Компоненты информационной системы

Компоненты созданной информационной системы являются независимыми микросервисами, и могут быть использованы как вместе так и отдельно. Так же к системе легко подключить новые микросервисы, если это потребуется, или технически возможно использовать микросервисы в других информационных системах.

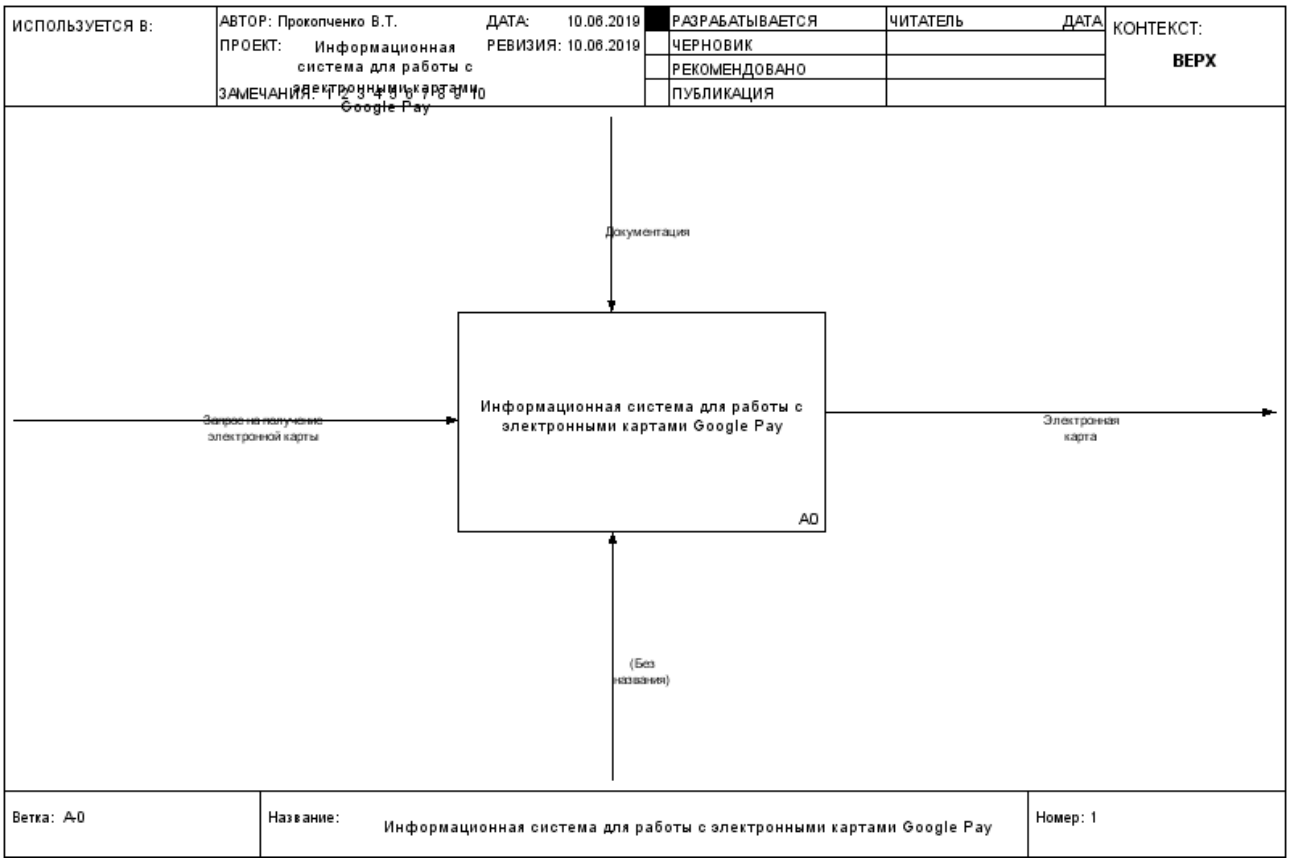

Рис. 12 Модельное представление работы системы. Уровень A0.

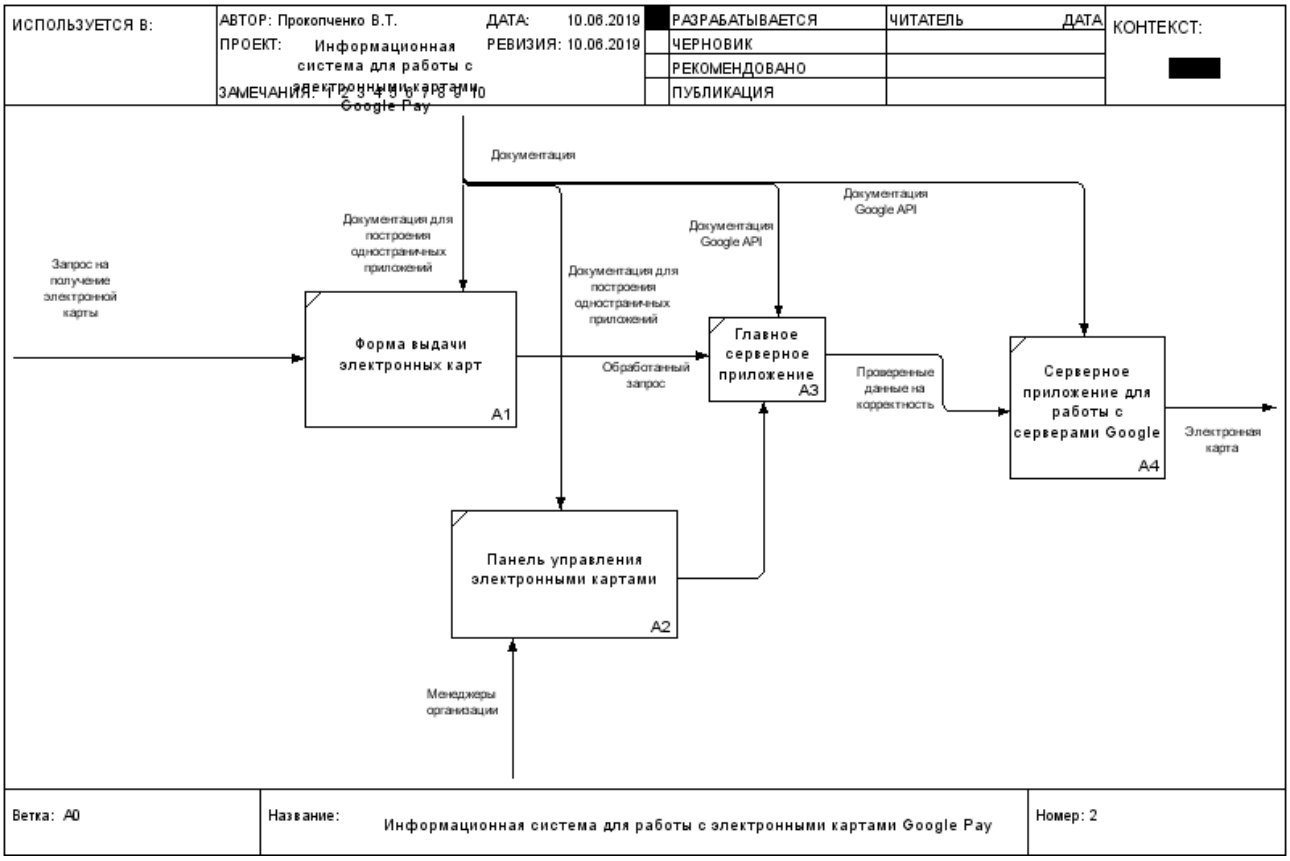

Рис. 13 Модельное представление работы системы. Уровень A1.

2.1.1 Интерфейс

Пользовательский и административный интерфейсы были выполнены как веб-сайты, с помощью JavaScript фреймворка Vue.js. Так же была установлена библиотека material-design для упрощения работы с CSS стилями.

Интерфейс состоит из двух составляющих:

- Форма выдачи электронных карт
- Административная панель управления электронными картами

Форма выдачи электронных карт это SPA(single page application)[23] web-приложение, которое может быть установлено на сервер ресторана, и иметь адрес соответствующий организации, выдающей карты. Оно отправляет http-запрос на адрес главного сервера информационной системы, и в зависимости от ответа настраиваются текстовые поля, внешний вид формы, поля для ввода гостем, конечным клиентом, и данные для отправки http-запроса на второй

микросервис системы, который работает с электронными картами Google. Приложение хранит информацию о входившем на сайт пользователе в файлах браузера, для того, чтобы с одного устройства, случайным образом, не были выданы несколько карт.

Порядок получение электронной карты через форму выдачи:

1. Подтверждение номера телефона.

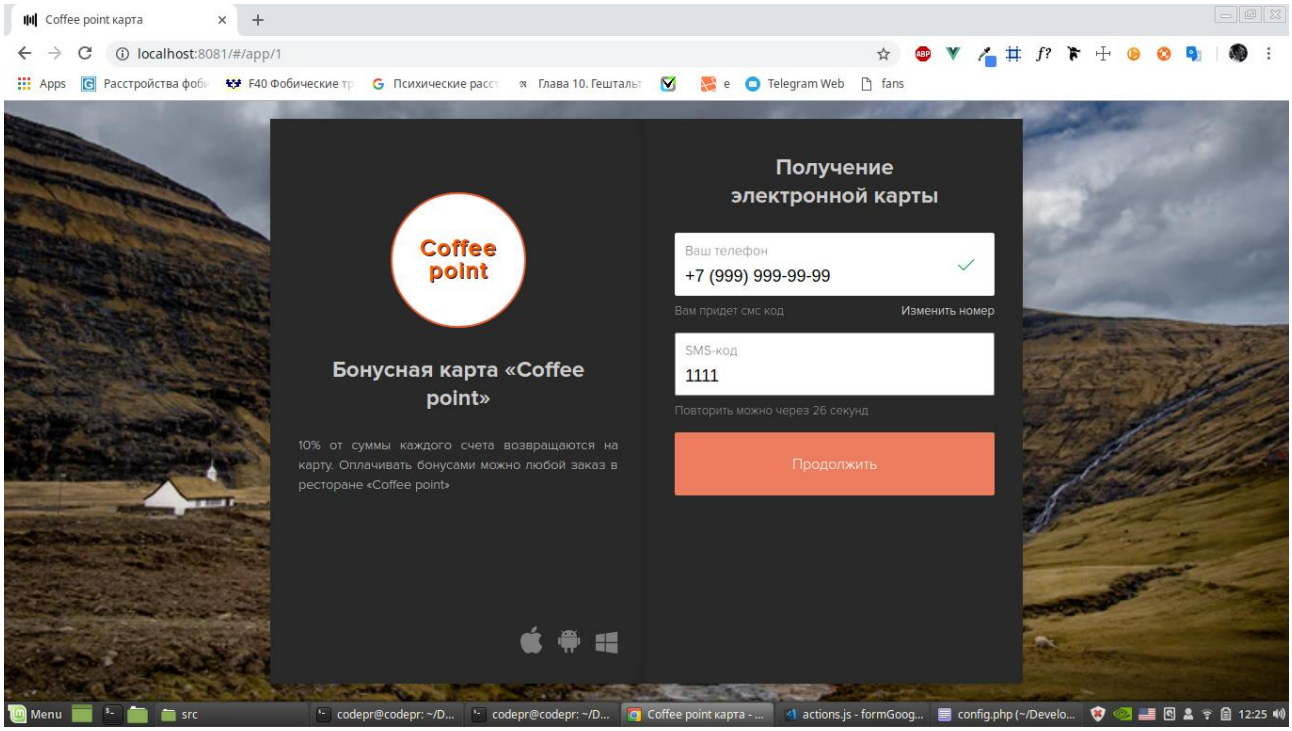

Рис. 14 Форма получение электронной карты. Подтверждение номера телефона.

2. Ввод персональных данных.

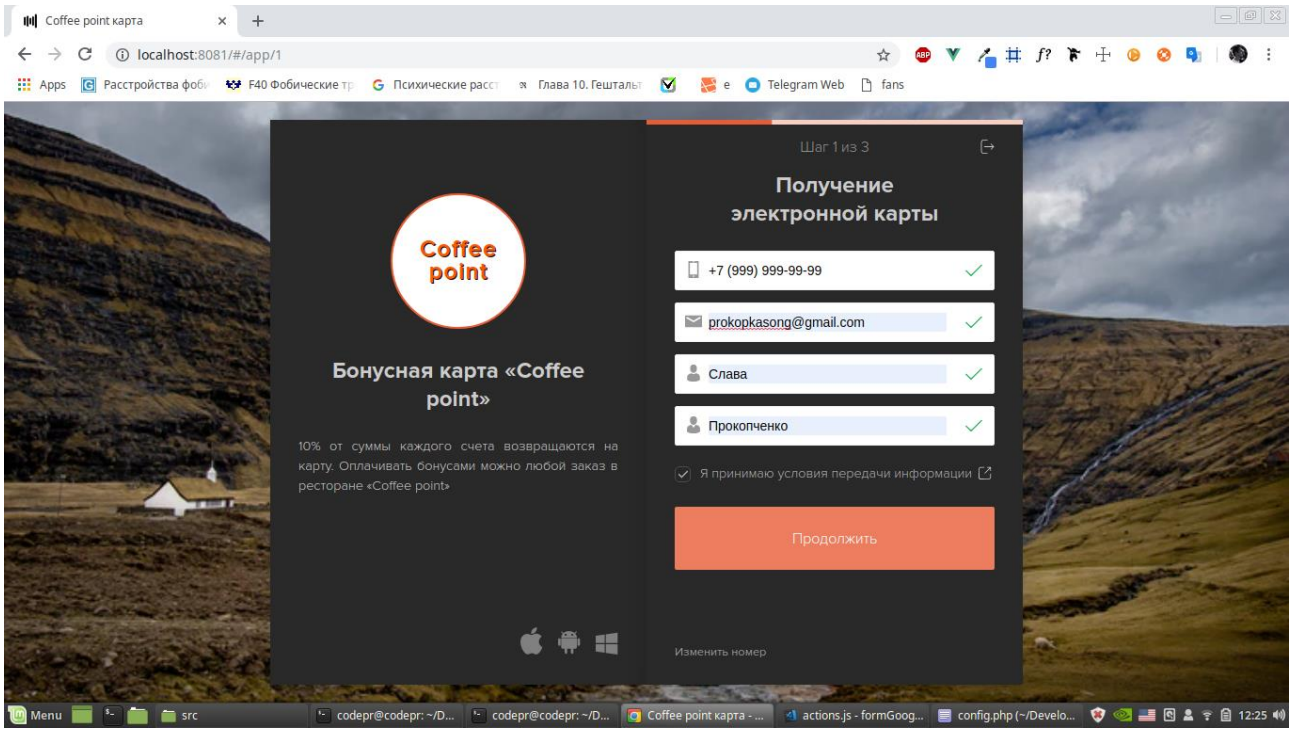

Рис. 15 Форма получение электронной карты. Ввод персональных данных.

3. Процесс загрузки.

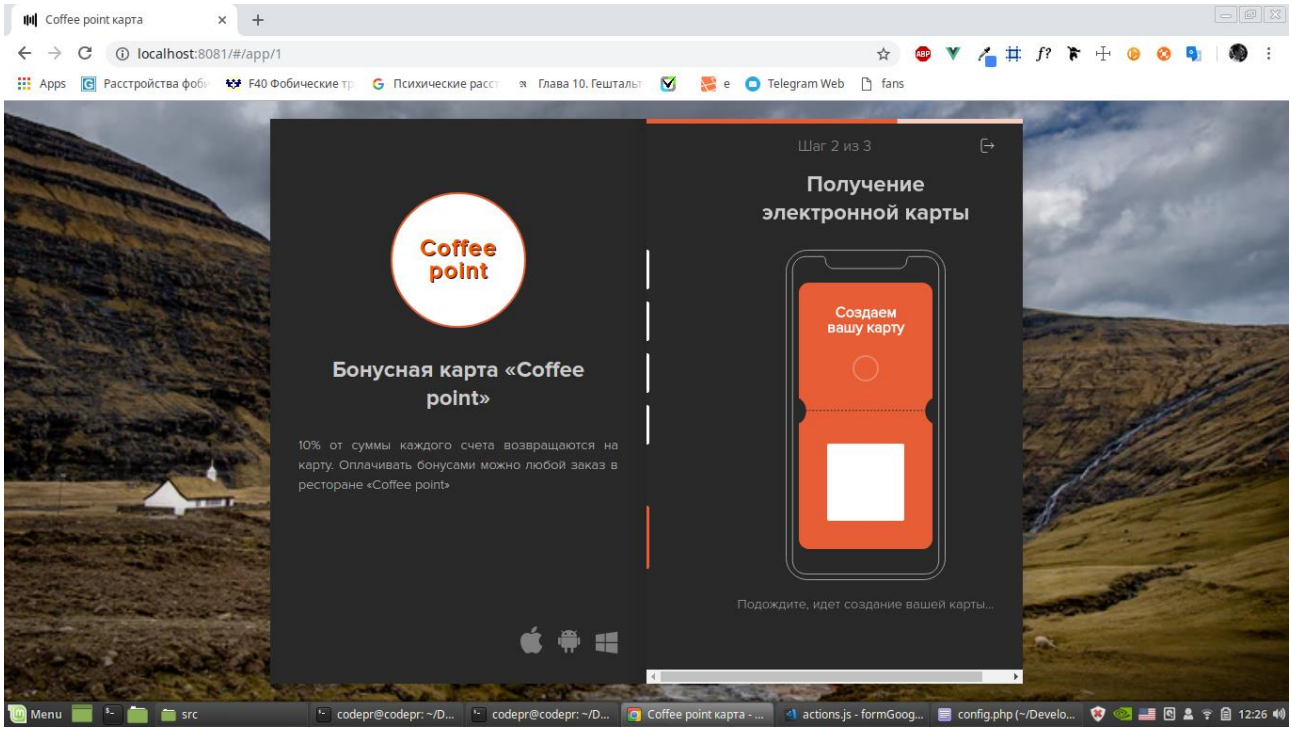

Рис. 16 Форма получение электронной карты. Процесс загрузки.

4. Получение электронной карты.

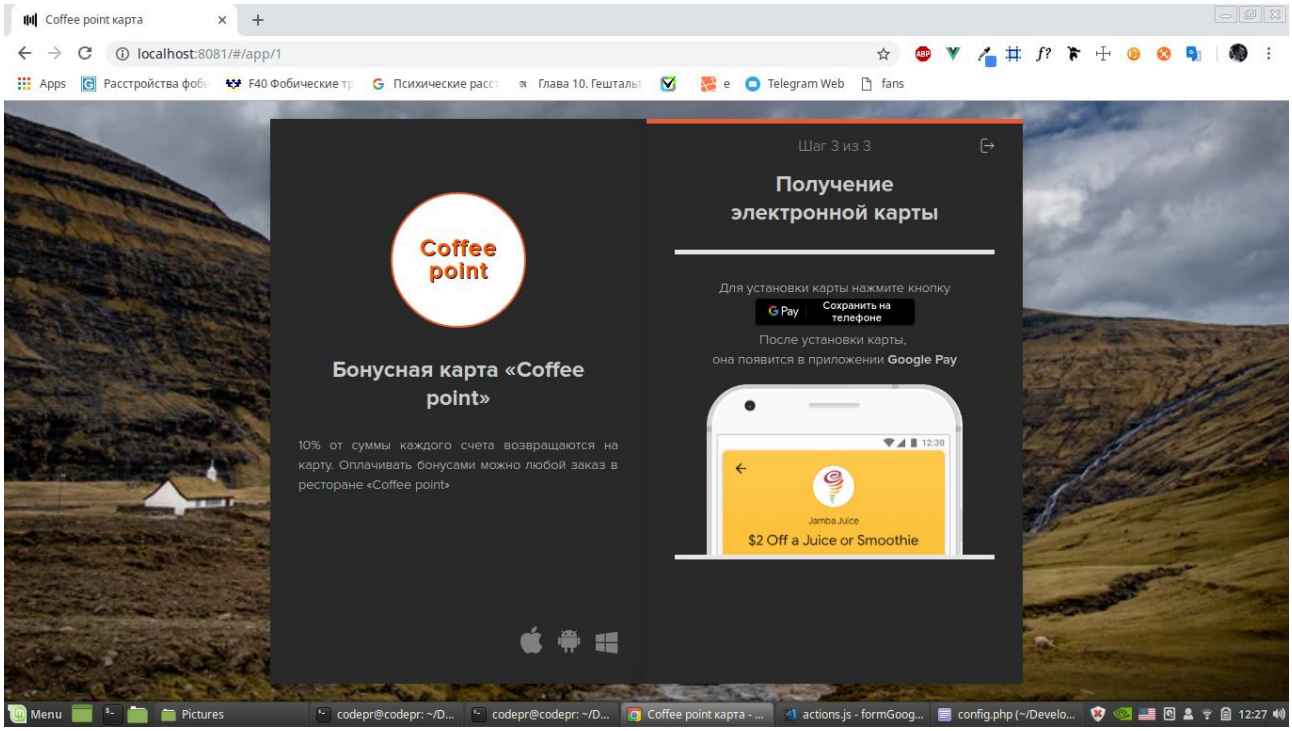

Рис. 17 Форма получение электронной карты. Получение электронной карты.

5. Подтверждение сохранения карты в Google Pay.

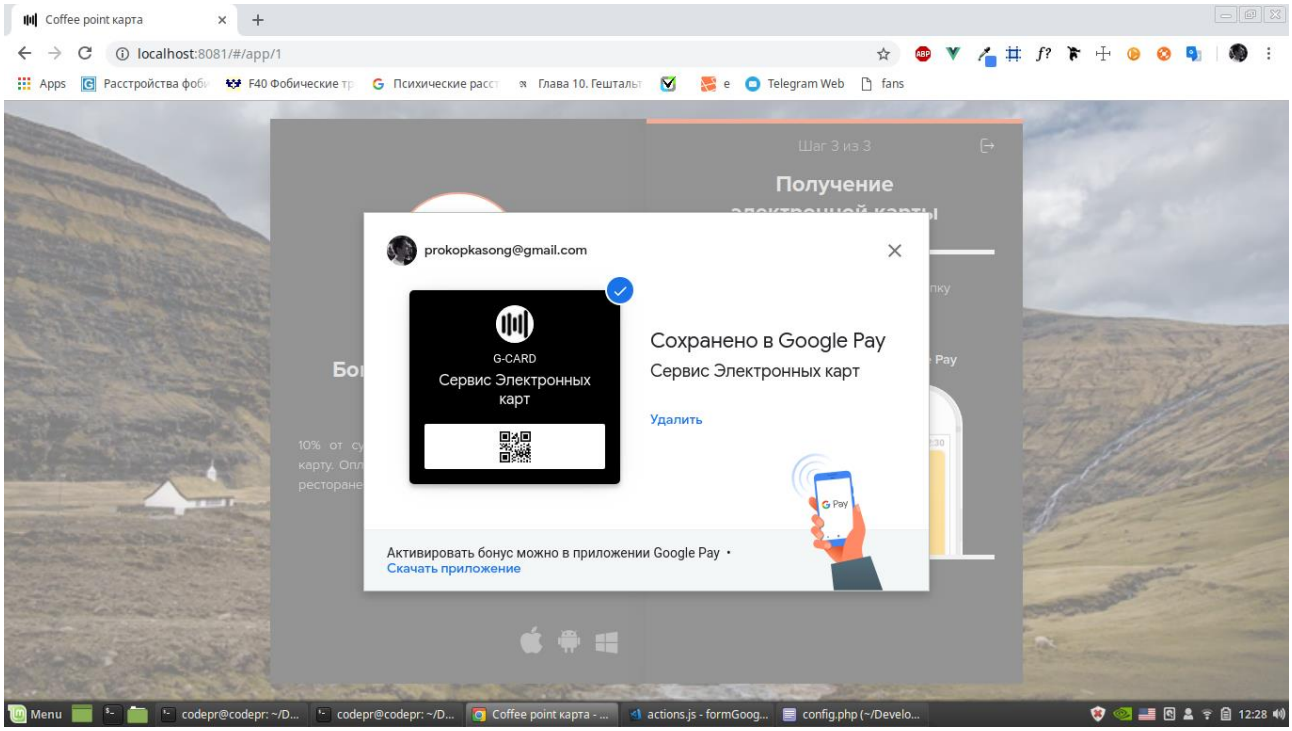

Рис. 18 Форма получение электронной карты. Подтверждение сохранения карты.

Административная панель настройки шаблонов электронных карт выполнена как одностраничное web-приложение, для управления электронными картами Google Pay. Приложение отправляет http-запросы на микросервис работающий с организациями, в приложении можно настроить свой шаблон электронных карт, получить ссылку на выдачу карт, и изменять карты действующих постоянных клиентов, обновлять их баланс и отправлять маркетинговые сообщения. В приложении реализована система авторизации пользователя в системе, для регистрации пользователь должен ввести свою электронную почту и пароль. Реализована возможность восстановления пароля через электронную почту. Реализован конструктор карты, с помощью которого пользователь сможет настроить свой собственный шаблон электронной карты, настроить цвет карты, тексты, присутствующие на карте, установить логотип и главное изображение.

2.1.2 Серверная составляющая

Серверная часть информационной системы состоит из следующих элементов:

- Внутренний API
- Внешний API

Внутренний API интерфейс, решающий задачи хранения и изменения информации об организациях, зарегистрированных в системе. Приложение реализовано на фреймворке Laravel 5.4. Использование Laravel предполагает следованию паттерна веб-разработки MVC (model view controller). Созданы следующие модели:

- User
- Application
- ImportServer
- Card
- Settings

Модель User работает с таблицей базы данных v1_users. Модель нужна для работы с данными о пользователях приложения, с помощью её методов можно найти пользователя по заданным параметрам, в основном используется для авторизации и определения прав доступа пользователя к конкретным шаблонам электронных карт. Свойства модели:

- Name
- Surname
- Phone
- Email
- Password
- Remember_token

Модель Application работает с таблицей базы данных v1_applications. Модель нужна для работы с данными организаций, так же осуществляет доступ к шаблонам электронных карт. Свойства модели:

- Name
- Roles
- Form_id
- disabled
- cabinet_id
- users_ids

Модель ImportServer работает с таблицей базы данных v1_importServers. Модель нужна для добавления, изменения и поиска информации о внедренных внешних сервисах для разных приложений в системе.

Свойства модели:

- type
- inquiry
- server
- disabled
- org_name
- login
- password
- application_id

Модель Card работает с таблицей базы данных v1_cards. Нужна для работы с данными электронных карт, использую её методы можно осуществить поиск, извлечение, обновление и изменение конкретного шаблона электронной карты. Свойства модели:

- main_image
- second_image
- bonus_text
- discount_text

- program_name
- organization_name
- text_color
- background_color
- application_id

Модель Settings работает с таблицей базы данных v1_settings. Модель требуется для реализации различных настроек для каждой компании, зарегистрированной в системе. Свойства модели:

- type
- application_id
- disabled
- inquiry
- server
- api_token

Модели позволяют осуществлять выборку, в том числе сложные SQL запросы, по соответствующим таблицам с помощью методов фреймворка Laravel, и так же другие операции с данными из таблиц базы данных.

Следующий по важности элемент приложения это контроллеры. Контроллеры работают с моделями приложения, они должны понимать из какой модели нужно взять данные, и какие из этих данных необходимо вернуть на заданный запрос. Созданы следующие контроллеры:

- UserController
- ApplicationController
- SettingsController
- CardController
- BillingController

UserController работает с моделями User, Application. Через него проходит авторизация пользователей в системе, регистрация пользователей, процесс восстановления пароля, так же определят права доступа конкретного пользователя к определенной организации и шаблоны электронных карт.

ApplicationController работает с моделями Application, Settings, Card. Определяет, к какой организации относится определенный шаблон электронных карт, осуществляет создание организаций внутри системы, определяет отношение настроек и организаций.

SettingsController работает с моделями Settings, Application. Через данный контроллер создаются и редактируются настройки системы для определённых организаций, определяет какие настройки применимы к каким организациям.

CardController работает с моделями Card, Application. Через контроллер проходят операции выборки, изменения, обновления и удаление шаблонов электронных карт, так же работает с клиентами организаций, выдает заданное количество клиентов организации, определяет какую информацию связанную с картами клиентов необходимо вернуть на входящий http-запрос.

BillingController работает с моделяи User, Application. Определят баланс организации внутри системы, проводит транзакции с организациями, определяет каким пользователям из каких организаций необходимо отправить напоминание об оплате сервиса, отключает шаблоны электронных карт при отрицательном балансе организации.

Laravel включает в себя авторизацию посредствам ввода e-mail адреса, который служит логином, и пароля пользователя. Шифратор Laravel использует OpenSSL для шифрования по алгоритмам AES-256 и AES-128. Все шифрованные значения подписаны кодом аутентификации сообщения (MAC) для предотвращения любых изменений в зашифрованной строке.

Внутренний API интерфейс хранит информацию о внешнем виде формы для выдачи электронных карт для каждой зарегистрированной организации.

Интерфейс отвечает на входящие http-запросы, если в заголовках запроса есть секретный токен доступа.

HTTP-маршрутизация внутреннего API включает в себя следующие маршруты:

(POST)http://localhost:8081/api/v1/user/login

На данный адрес интерфейс принимает запросы метода POST, в теле запроса должна быть строка в формате JSON, в которой должны находиться в соответствующих полях логин(email) и пароль. На запрос внутренний API возвращает токен доступа, его необходимо прикреплять к заголовку Authorization в последующих запросах к интерфейсу.

(POST)http://localhost:8081/api/v1/user/register

Маршрут нужен для регистрации пользователя в системе, в теле запроса нужно прикрепить строку JSON с соответствующими полями необходимыми для регистрации:

- Электронная почта
- Номер телефона
- Имя
- Фамилия
- Пароль
- Подтверждение пароля

Если присутствуют все поля, интерфейс проверяет, есть ли пользователь с данным адресом электронной почты, если такого пользователя нет, то регистрация будет успешно завершена, чтобы получить токен доступа, необходимо отправить запрос на маршрут (POST)http://localhost:8081/api/v1/user/login, прикрепляя к запросу JSON с введёнными при регистрации реквизитами для входа.

(POST)http://localhost:8081/api/v1/application/register

Маршрут требуется для регистрации нового шаблона электронной карты. Для регистрации шаблона необходимо отправить http-запрос с телом, включающем в себя имя регистрируемого шаблона, так же в заголовках отправляемого запроса необходимо прикрепить токен доступа, полученный при авторизации. Интерфейс проверяет, есть ли в базе шаблон с отправленным именем, если такого шаблона в базе данных не имеется, то регистрация шаблона успешно заканчивается, и внутренний API добавляет запись с именем шаблона в соответствующую таблицу базы данных.

Чтобы получить список зарегистрированных пользователем шаблонов, необходимо отправить http-запрос на следующий маршрут внутреннего интерфейса:

(GET)http://localhost:8081/api/v1/applications

В заголовках запроса обязательно должен быть токен доступа, полученный при авторизации в системе. Интерфейс определяет по токену доступа пользователя, ищет в базе данных доступные данному пользователю шаблоны, формирует из них список и возвращает строкой в формате JSON.

Если внутренний интерфейс работает с пользователем и его данными, то внешний API интерфейс работает исключительно с внутренним API интерфейсом, на запросы, отправленные из вне системы, интерфейс отвечать не будет.

Маршрутизация внешнего интерфейса включает в себя следующие маршруты:

(POST)http://localhost:8082/api/v1/createTemplate

Создаёт шаблон электронных карт в базе данных Google, в теле этого запроса необходимо указать имя шаблона, и данные связанные с дизайном шаблона. Интерфейс проверяет отправленные внутренним API данные, в случае корректности принимаемых данных, интерфейс формирует данные для отправки их на сервера Google, если Google возвращает ответ об удачно

созданном шаблоне внешний API интерфейс создаёт запись об удачно созданном шаблоне электронных карт и отвечает внутреннему интерфейсу об удачно созданном шаблоне, и возвращает имя созданного шаблона, внутренний интерфейс в свою очередь отвечает пользовательскому интерфейсу об удачной операции, пользователь получает уведомление об удачно созданном шаблоне на панели управления электронными картами.

Для изменения созданного шаблона существует маршрут с ниже приведенным адресом:

(POST)http://localhost:8082/api/v1/updateTemplate

В теле запроса необходимо указать какой именно шаблон необходимо изменить, и поля с соответствующими данными для изменения. Интерфейс сформирует эти данные и отправит их на соответствующий маршрут API Google, если последний утвердит успешность операции, то в таблице с шаблонами в базе данных внешнего API будет произведено изменение с заданной записью, после чего последует ответ внутреннему программному интерфейсу об успешной операции изменения шаблона, последний в свою очередь уведомит об этом пользователя сообщением в административную панель управления электронными картами.

Для создания и получения ссылки на скачивание персональной электронной карты Google Pay требуется отправить http-запрос на следующий маршрут:

(POST)http://localhost:8082/api/v1/createCard

В теле запроса необходимо подставить данные о будущем держателе электронной карты, и указать от какого шаблона необходимо создать карту. Если все обязательные поля будут присутствовать в запросе, то внешний интерфейс обработает эти данные и сформирует в требуемый компанией Google формат, и отправит их на сервера Google, если при этом он получит положительный ответ о создании персональной электронной карты и получит

ссылку на скачивание этой карты, то он сформирует эти данные и вернет их в формате JSON. Дальше действует пользовательский интерфейс выдачи электронных карт, принимает данные от сервера и строит ссылку на скачивание карты. Далее пользователю будет предложено авторизоваться под аккаунтом Google, если такого нет, то будет предложена регистрация в системе Google, в успешном исходе прохождения этих этапов пользователь будет уведомлен, что в его приложении Google Pay создана электронная карта, на ней он может просмотреть баланс и увидеть последние новости организации, внедрившей электронные карты Google Pay.

1.2 Инструмент “Google Pay API for Passes”

В разработке информационной системы используется программный интерфейс приложения “Google Pay API for Passes”, он представляет из себя набор методов для управления электронными картами постоянного клиента. С его помощью можно взаимодействовать с пользователями Android устройств, отправляя информацию в режиме реального времени, сообщения и уведомления на основе местоположения владельца карты.

Для внедрения этого механизма в информационную систему необходимо проанализировать возможности предоставляемого компанией “Google” API интерфейса.

Интерфейс позволяет создавать, изменять и получать информацию о шаблоне электронной карты или о персональной карты клиента средствами http-запросов к специальному серверу Google. Удобство использования данного интерфейса состоит в том, что не нужно задумываться, как установить электронную карту в Android устройство клиента, Google позаботились об этом сами, нужно лишь отправить на их сервер информацию о шаблоне карты и данные клиента, для отображения этих данных на его карте, и Google вернёт ссылку на установку электронной карты, перейдя по этой ссылке кленту будет предложено установить уже готовую электронную карту в его Android устройство, причём, не важно на каком устройстве клиент заполнит форму, на всех устройствах клиента, где установлено приложение Google Pay клиент получит возможность пользоваться выданной ему электронной картой, данные о карте так же будут изменяться на всех Android устройствах клиента в реальном времени. Синтаксис запросов и тип возвращаемых данных к серверам Google строго определены на стороне самого интерфейса [https://developers.google.com/pay/passes/reference/v1/?hl=ru].

Чтобы создать шаблон электронной карты нужно ознакомиться со стандартом, по которому будет строиться электронная карта.

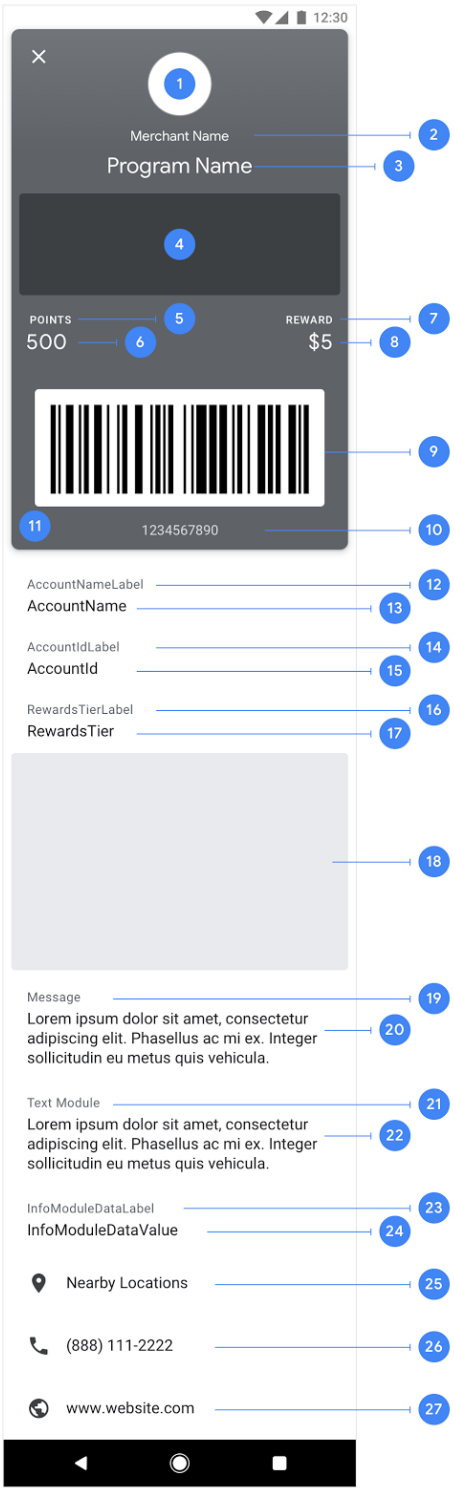

Рис. 19 Стандарт электронной карты Google Pay постоянного клиента

Подробнее обо всех пунктах, представленных на (Рис. 19):

6. Главная картинка на электронной карте, это место используется для размещения логтипа компании, в которой внедрены электронные карты Google Pay.
7. Поле для названия компании, в которой внедрены электронные карты Google Pay.
8. Поле для названия программы лояльности, в которой находится клиент организации.
9. Главная картинка на карте.
10. Название бонусов, которые выдаёт компания, в которой внедрены электронные карты
11. Динамическое поле, в котором отображается количество накопленных клиентом бонусов.
12. Название поля, под которым будет отображаться размер скидки, или размер накопления бонусов
13. Динамическое поле, которое отображается процент накопления бонусов или скидки.
14. Штрих-код или QR-код, для удобства считывания кассовыми аппаратами организации, в которой внедрены электронные карты.
15. Номер карты клиента.
16. Цвет фона карты в формате RGB.
17. С 12 по 24 пункты соответстуют заголовкам и текстовым полям, размещаемых на карте. Так же в эти поля можно вставить картинки, например основные блюда ресторана.
25. Ссылка на местонахождение компании, ссылка открывается в приложении Google Maps, или в любом другом приложении, поддерживающим ссылки местонахождения.
26. Номер телефона для связи с компанией, при нажатии с Android устройства, производится звонок на телефонный номер указанный на карте.

27. Поле в котором можно разместить веб-сайт компании.

После ознакомления со стандартными полями электронной карты Google Pay необходимо создать дизайн электронной карты, и отправить http-запрос методом POST на следующий адрес сервера Google:

https://www.googleapis.com/walletobjects/v1/loyaltyClass

В теле запроса необходимо отправить строку в формате JSON следующего вида:

```
{
    "id": STRING,
    "issuerName": STRING,
    "programLogo": URL,
    "programName": STRING,
    "reviewStatus": STRING
}
```

Подробнее обо всех составляющих JSON строки:

Id – уникальный идентификатор шаблона карты, он будет использоваться в дальнейшем как ключ определяющий к какому именно шаблону электронной карты требуется применить выбранный метод. Google требуют, чтобы id состоял в себе секретный токен доступа, который можно получить при регистрации аккаунта разработчика Google. Id должен выглядеть следующим образом:

secretToken.identifier

Где sercetToken – секретный токен доступа, который можно получить при регистрации аккаунта разработчика Google.

IssuerName – название шаблона электронной карты, обычно используют имя бренда, например “Duo Coffee”, отображается в заголовке электронной карты.

ProgramLogo – Ссылка на изображение, которое требуется поместить на место логотипа карты.

ProgramName – Имя программы лояльности.

ReviewStatus – Статус шаблона электронной карты, возможные значения:

- "approved"
- "draft"
- "rejected"
- "underReview"

Для изменения созданного шаблона требуется отправить http-запрос методом PUT на следующий адрес:

https://www.googleapis.com/walletobjects/v1/loyaltyClass/resourceId

Где resourceId – id ранее созданного шаблона.

В теле запроса необходимо отправить строку в формате JSON следующего вида:

```
{
    "id": STRING,
    "issuerName": STRING,
    "programLogo": URL,
    "programName": STRING,
    "reviewStatus": STRING,
    …
}
```

В указанной JSON строке показаны только обязательные поля, все поля которые можно изменять можно посмотреть здесь [https://developers.google.com/pay/passes/reference/v1/loyaltyclass/update?hl=ru].

Для выпуска персональной карты клиента, после создания шаблона необходимо создать персональную электронную карту. Эта операция, так же

как и создания шаблона карты производится средствами Google Pay API for Passes.

Для создания персональной электронной карты постоянного клиента требуется отправить http-запрос методом POST на следующий адрес сервера:

https://www.googleapis.com/walletobjects/v1/loyaltyObject

В теле запроса требуются сформировать строку в формате JSON, где нужно указать следующий обязательные параметры:

```
{
    "id": STRING,
    "classId": STRING,
    "state": URL
}
```

Где id – уникальный идентификатор персональной карты, id является ключом, с помощью которого в дальнейшем можно будет обращаться именно к созданной электронной карте.

ClassId – Id шаблона электронной карты.

State – Значение, указывающее в какое состояние необходимо привести электронную карту постоянного покупателя.

Для обновления карты нужно отправить http-запрос на адрес:

https://www.googleapis.com/walletobjects/v1/loyaltyObject/resourceId

Где resourceId – secretToken и id электронной карты, которую требуется изменить.

В теле запроса необходимо передать строку в формате JSON:

```
{
    "id": STRING,
    "classId": STRING,
    "state": URL
}
```

Где id – уникальный идентификатор персональной карты, id является ключом, с помощью которого в дальнейшем можно будет обращаться именно к созданной электронной карте.

ClassId – Id шаблона электронной карты.

State – Значение, указывающее в какое состояние необходимо привести электронную карту постоянного покупателя.

1.3 Руководство пользователя.

Назначение информационной системы состоит в создании и редактировании шаблонов электронных карт Google Pay, и в выдаче персональных электронных карт клиентам организации, внедрившей электронные карты. Такой подход помогает пользователям системы внедрить электронные карты Google Pay, не затрачивая средства на разработку собственной системы.

Условием применения является наличие персонального компьютера со стабильным выходом в сеть Интернет. Пользователь должен обладать навыками работы с Интернет браузером.

Для начала работы требуется зайти на сайт информационной системы для управления электронными картами Google Pay, авторизоваться в системе и настроить шаблон электронной карты. Внедрить в свой сайт организации форму выдачи электронных карт. После чего информационная система готова к работе.

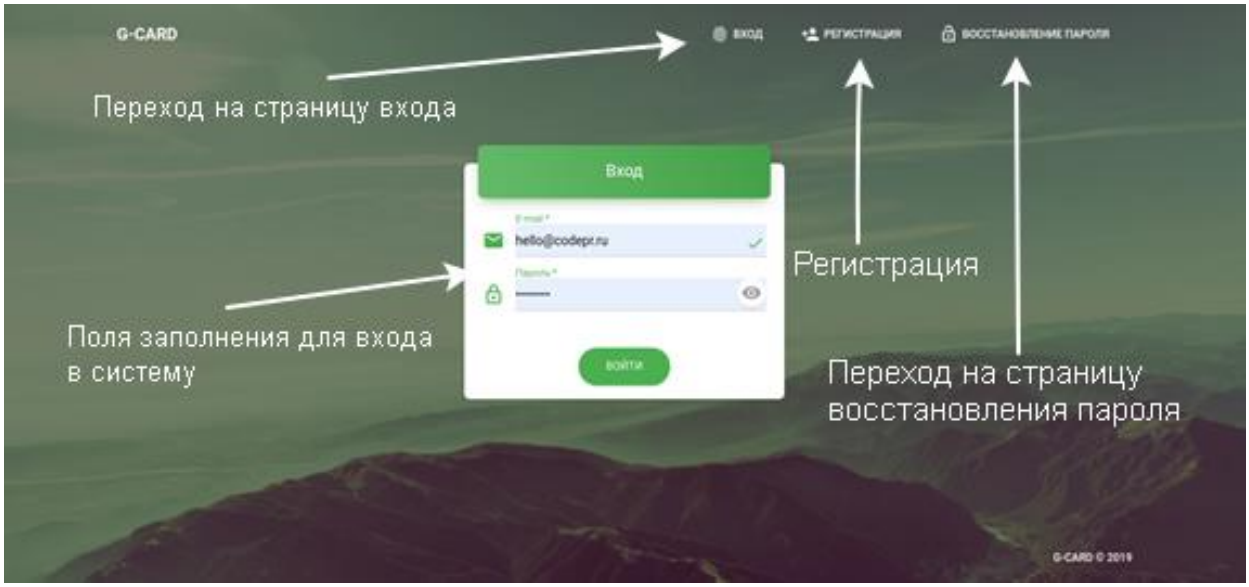

Рис. 20. Скриншот входа в информационную систему

При переходе на сайт информационной системы пользователь попадает на страницу авторизации. На скриншоте выше (см. 20) указаны кнопки на странице авторизации в системе и подписаны соответствующие им функции.

После авторизации в системе пользователь попадает на главную страницу информационной системы, на которой есть информация о балансе организации и количестве выданных крат (см. Рис. 21).

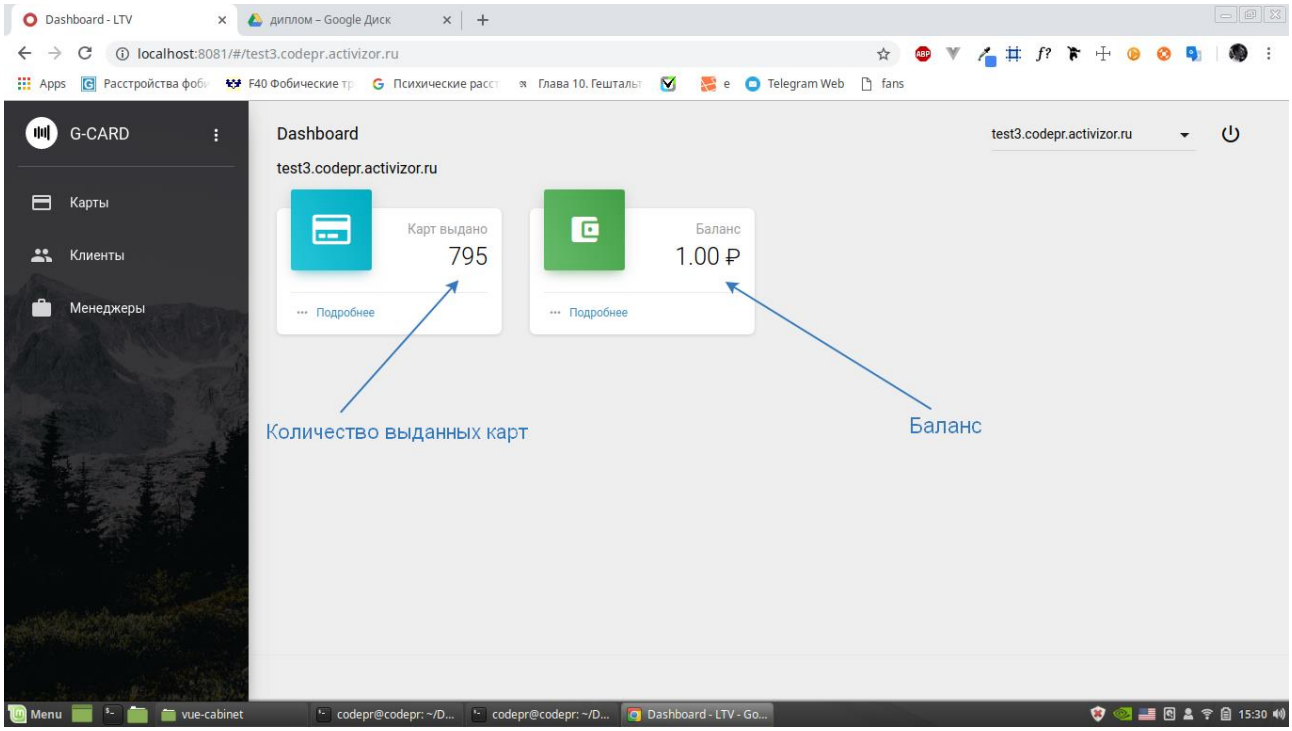

Рис. 21. Скриншот главной страницы

Перейдя на вкладку “Карты” пользователь может настроить шаблон электронной карты, на скриншоте (см. Рис. 22) подписаны пункты меню, с помощью которых можно настроить шаблон электронной карты.

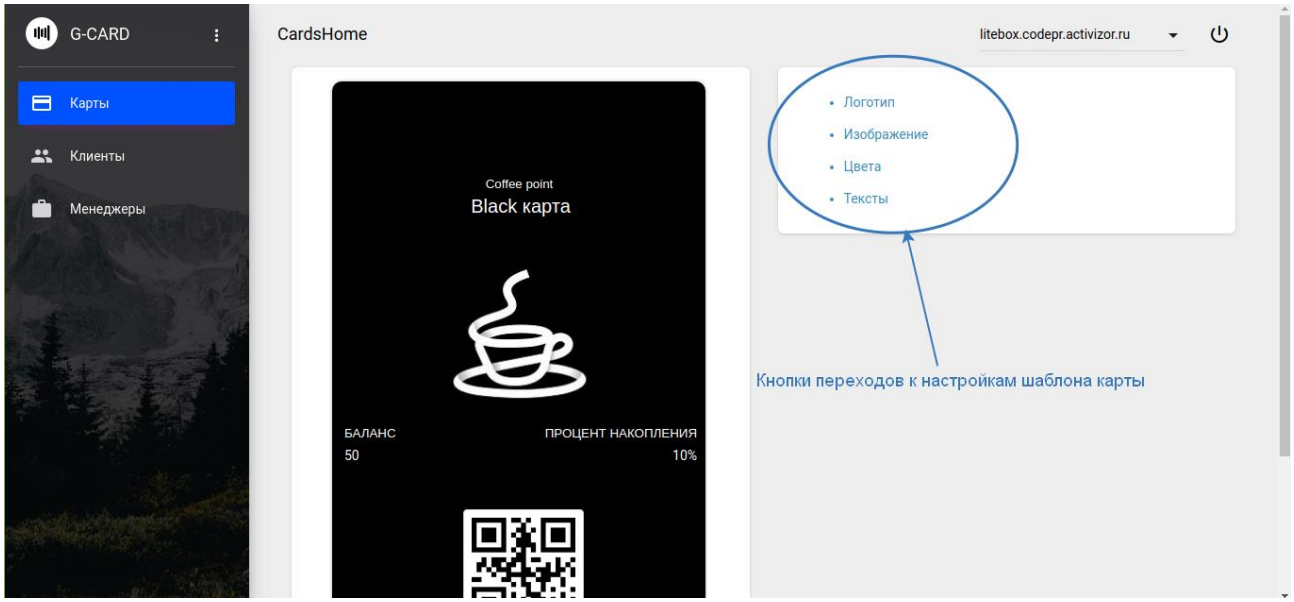

Рис. 22. Скриншот вкладки “Карты”

Перейдя на вкладку “Клиенты” пользователь может просмотреть таблицу выданных карт, в таблице вынесены все основные свойства конкретной карты (см. Рис. 23).

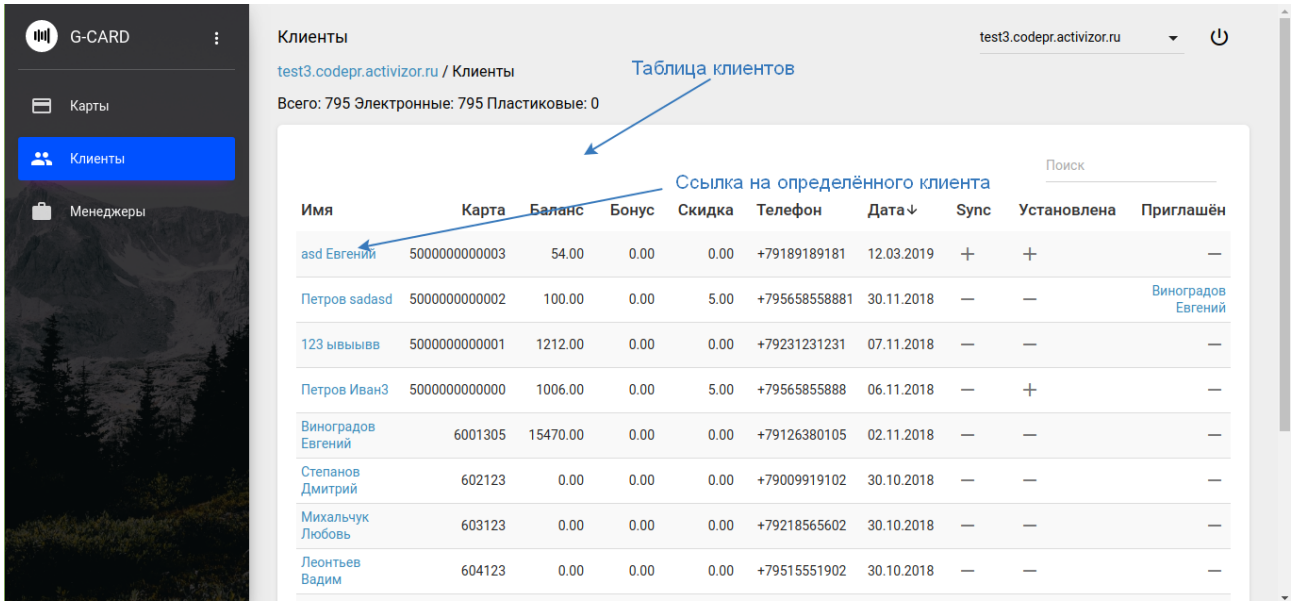

Рис. 23. Скриншот вкладки “Клиенты”

При нажатии на конкретного клиента в таблице, можно просмотреть и изменить информацию о выданной карте, сделанные изменения отобразятся на карте клиента (см. Рис. 24).

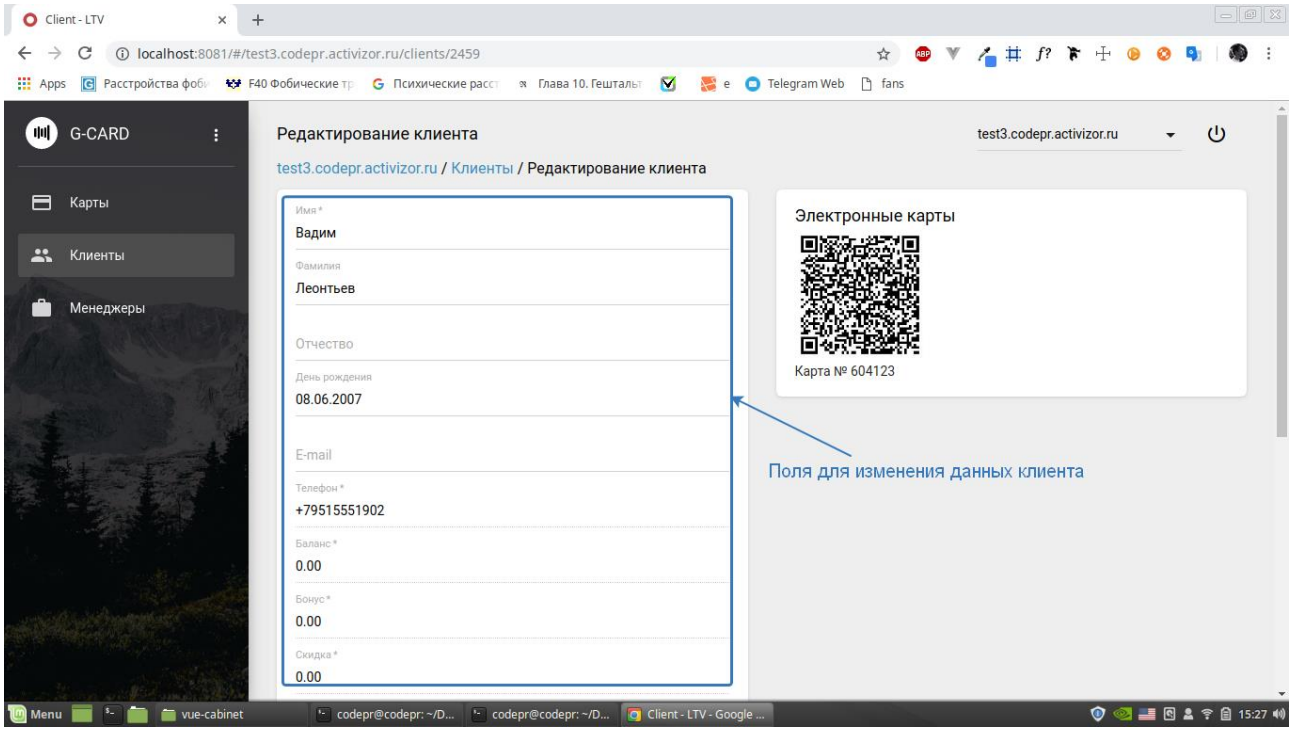

Рис. 24. Скриншот вкладки “Клиент”

Перейдя на вкладку “Менеджеры” пользователь может пригласить другого пользователя в кабинет управления электронной карты, так же приглашённый пользователь получит права на редактирование шаблона электронной карты, просмотр и редактирование выпущенных электронных карт. На странице показана таблица менеджеров, у которых есть доступ к текущей организации, и таблица приглашённых, но еще не зарегистрированных в системе пользователей.

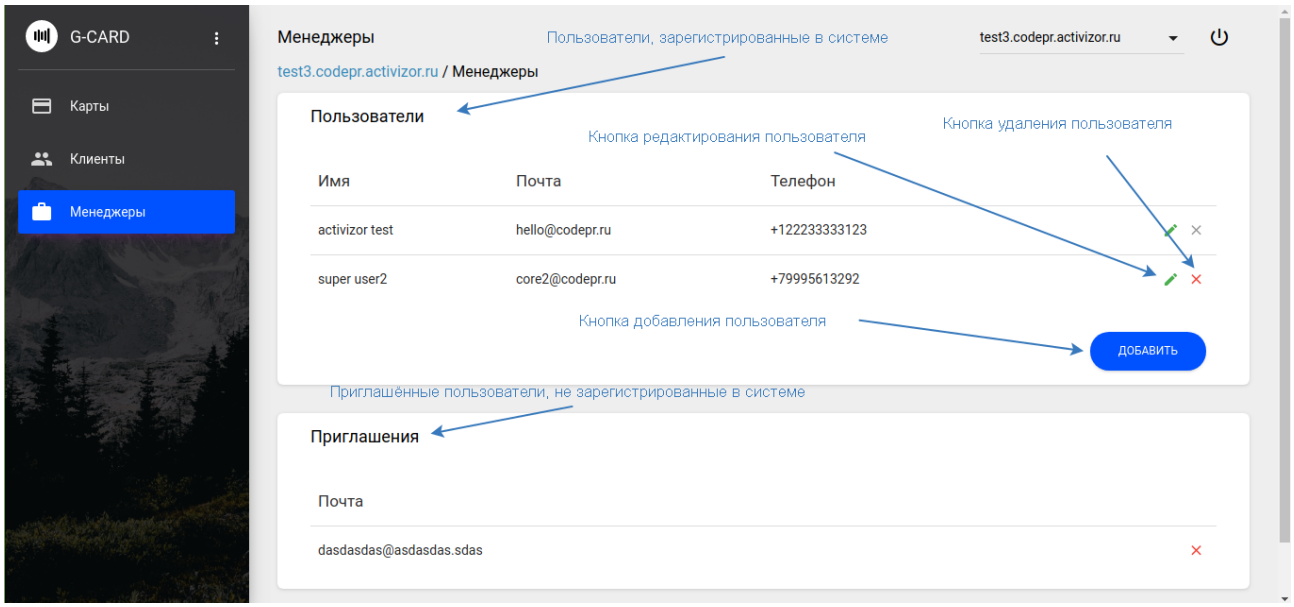

Рис. 25. Скриншот вкладки “Менеджеры”.

На скриншоте подписаны таблицы и кнопка добавления пользователя, так же указана кнопка редактирования и удаления пользователя (см. Рис. 25). Нажав на кнопку редактирования менеджера в таблице менеджеров, будет осуществлен переход на страницу редактирования данного пользователя, где можно изменить имя, фамилию, номер телефона пользователя, так же можно изменить получаемые менеджером уведомления (см. Рис. 26).

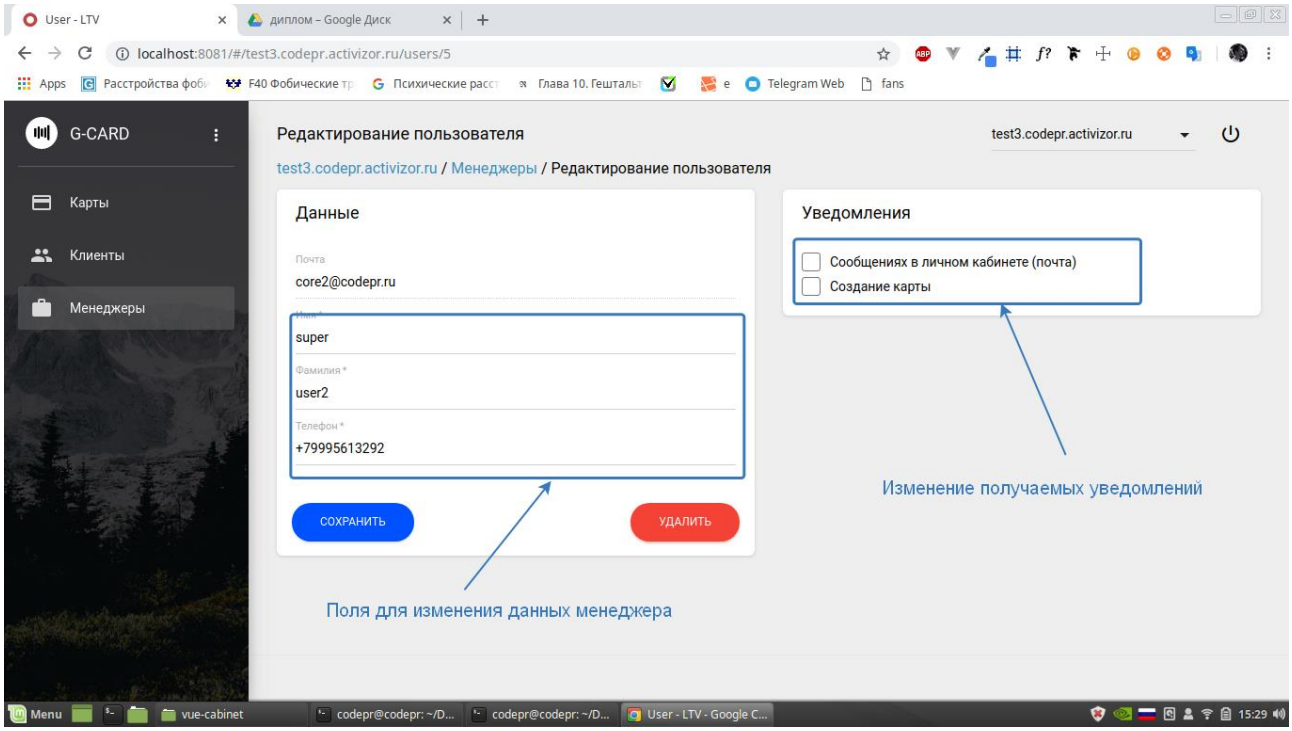

Рис. 26. Скриншот вкладки "Менеджер".

При отказе или сбое в работе сайта необходимо перезагрузить страницу. В случае, если перезагрузка не решила проблему, необходимо закрыть интернет браузер, заново открыть браузер и зайти на сайт информационной системы.

Заключение

В бизнесе есть много задач, которые являются рутинными, если автоматизировать часть из них, бизнес становится более доходным занятием. С помощью информационной системы для работы с электронными картами «Google Pay» любой предприниматель сможет внедрить электронные карты в свой бизнес, и автоматизировать процесс выдачи карт своим клиентам.

Цель разработки**:** разработать информационную систему для управления электронными картами «Google Pay», обеспечивающую возможность создания и изменения шаблонов электронных карт, выдачу электронных карт конечному клиенту.

Для достижения цели были сформулированы и достигнуты следующие задачи:

1. Произвести сбор и обработку информации по существующим системам управления электронных карт для выявления обязательных функций системы.
2. В соответствии с техническим заданием провести разработку информационной системы для управления электронными картами «Google Pay».
3. Подготовить техническую и сопроводительную документацию.

Работа носит законченный характер, цель работы достигнута, задачи выполнены. Продукт полностью соответствует техническому заданию и требованиям заказчика.

Список информационных источников

1. Оформитель библиографических ссылок // SNOSKA.INFO URL: http://snoskainfo.ru/ (дата обращения: 14.02.2019).
2. ГОСТ 34.602-89 Межгосударственный стандарт. «Информационная технология (ИТ). Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы» от июнь 2009 № 661 // М.: Стандартинформ, 2009 с изм. и допол. в ред. от 27.12.2006 г.
3. Парминдер С. К. Микросервисы и контейнеры Docker. - М.: ДМК Пресс, 2019. - 240 с.
4. Ньюмен С. Building Microservices. - СПб.: Питер, 2016. - 304 с.
5. ГОСТ 19.505-79 ЕСПД. Руководство оператора. Требования к содержанию и оформлению. М.: Стандартинформ, 2010. 2 с.
6. Определение и классификация информационных систем // tsput.ru URL: http://tsput.ru/res/informat/sist_seti_fmo/lekcii/lekciy-1.html (дата обращения: 08.09.2019).
7. ГОСТ Р 56645.5-2015 Национальный стандарт Российской Федерации. «Системы дизайн-менеджмента. Термины и определения» от 01.06.2016 № 1577-ст // М.: Стандартинформ, 2016 с изм. и допол. в ред. от 19.10.2015 г.
8. Ergogames [Электронный ресурс]: База данных как сервис / Дата обращения: 29.02.2019. — Режим доступа: Intefaces pices
9. Прогрессивный JavaScript-фреймворк Vue.js // Vue.js URL: https://ru.vuejs.org/index.html (дата обращения: 02.04.2019).
10. Vue.js для сомневающихся. Все, что нужно знать // habr.com URL: https://habr.com/ru/post/329452/ (дата обращения: 02.04.2019).
11. Тот самый PHP-фреймворк для веб-ремесленников Laravel // laravel.ru URL: https://laravel.ru/ (дата обращения: 01.02.2019).
12. Material Design Palette // materialpalette URL: https://www.materialpalette.com/light-blue/indigo (дата обращения: 20.05.2019).

13. Гаврилова Т А.. Хорошевский В.Ф. Базы знаний интеллектуальных систем. - СПб.: Питер, 2001.-384 с.
14. ГОСТ Р 52657-2006 Национальный стандарт Российской Федерации. «Информационно-коммуникационные технологии в образовании. Образовательные интернет-порталы федерального уровня. Рубрикация информационных ресурсов.» от 01.07.2008 № 423-ст // М.: Стандартинформ, 2007 с изм. и допол. в ред. от 27.12.2006 г.
15. ГОСТ Р 52653-2006 Национальный стандарт Российской Федерации. «Информационно-коммуникационные технологии в образовании. Термины и определения.» от 01.07.2008 № 419-ст // М.: Стандартинформ, 2018 с изм. и допол. в ред. от 27.12.2006 г.
16. ГОСТ 2.105-95. Межгосударственный стандарт «Единая система конструкторской документации. Общие требования к текстовым документам» от 08.08.1995 № 426 //Всероссийским научно-исследовательским институтом стандартизации и сертификации в машиностроении (ВНИИНМАШ) Госстандарта России. 1995 г.
17. ГОСТ 19.502-78. Межгосударственный стандарт «Единая Система Программной Документации. Описание применения. Требования к содержанию и оформлению» от 01.01.1980 № 3350 // М.: Стандартинформ, 2010. с изм. и допол. в ред. от январь 2010
18. ГОСТ 19.503-79 Межгосударственный стандарт «Единая Система Программной Документации. Руководство системного программиста. Требования к содержанию и оформлению.» от 01.01.1980 № 74 // М.: Стандартинформ, 2010 с изм. и допол. в ред. от январь 2010
19. ГОСТ 19.504-79 Межгосударственный стандарт. «Единая система программной документации. Руководство программиста. Требования к содержанию и оформлению» от 01.01.1980 № 74 // М.: Стандартинформ, 2010 с изм. и допол. в ред. от январь 2010

20. ГОСТ 19.505-79 Межгосударственный стандарт. «Единая система программной документации. Руководство оператора. Требования к содержанию и оформлению» от 01.01.1980 № 74 // М.: Стандартинформ, 2010 с изм. и допол. в ред. от январь 2010
21. ГОСТ Р 53620-2009 Национальный стандарт Российской Федерации. «Информационно-коммуникационные технологии в образовании. Электронные образовательные ресурсы. Общие положения" от 01.01.2011 № 956-ст // М.: Стандартинформ, 2018. с изм. и допол. в ред. от 15.12.2009 г
22. Бенкен, Е.С. PHP, MySQL, XML. Программирование для Интернета (+ CD-ROM) / Е.С. Бенкен. - М.: БХВ-Петербург, 2011. - 250 с.
23. Веллинг, Л. Разработка веб-приложений с помощью PHP и MySQL / Л. Веллинг. - М.: Диалектика / Вильямс, 2016. - 116 с.
24. Гизберт, Дамашке PHP и MySQL / Дамашке Гизберт. - М.: НТ Пресс, 2011. - 663 с.
25. Дунаев, Вадим HTML, скрипты и стили Уцененный товар (№1) / Вадим Дунаев. - М.: БХВ-Петербург, 2015. - 816 с.
26. Дэн, Рамел Joomla! Для профессионалов / Рамел Дэн. - М.: Диалектика / Вильямс, 2014. - 226 с.
27. Жадаев, Александр PHP для начинающих / Александр Жадаев. - М.: Питер, 2014. - 251 с.
28. Клименко, Роман Веб-мастеринг на 100% / Роман Клименко. - М.: Питер, 2013. - 512 с.
29. Колисниченко, Д.Н. PHP 5/6 и MySQL 6. Разработка Web-приложений (+ CD-ROM) / Д.Н. Колисниченко. - М.: БХВ-Петербург, 2010. - 163 с.
30. Колисниченко, Денис PHP и MySQL. Разработка Web-приложений / Денис Колисниченко. - М.: БХВ-Петербург, 2013. - 560 с.
31. Кристиан, Уэнц PHP и MySQL. Карманный справочник / Уэнц Кристиан. - М.: Диалектика / Вильямс, 2016. - 656 с.

32. Кузнецов, Максим PHP. Народные советы (+ CD-ROM) / Максим Кузнецов , Игорь Симдянов. - М.: БХВ-Петербург, 2007. - 368 с.
33. Кузнецов, Максим Самоучитель PHP 5/6 / Максим Кузнецов , Игорь Симдянов. - М.: БХВ-Петербург, 2009. - 672 с.
34. Локхарт, Джош Современный PHP. Новые возможности и передовой опыт / Джош Локхарт. - М.: ДМК Пресс, 2016. - 304 с.
35. Ляпин, Д.А. PHP — это просто. Начинаем с видеоуроков (+ CD-ROM) / Д.А. Ляпин. - М.: БХВ-Петербург, 2013. - 722 с.
36. Макаров, Александр Yii. Сборник рецептов / Александр Макаров. - М.: ДМК Пресс, 2015. - 372 с.
37. Маклафлин, Б. PHP и MySQL. Исчерпывающее руководство / Б. Маклафлин. - М.: Питер, 2013. - 512 с.
38. Никсон, Робин Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript и CSS / Робин Никсон. - М.: Питер, 2013. - 151 с.
39. Прохоренок, Николай HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера (+ CD-ROM) / Николай Прохоренок. - М.: БХВ-Петербург, 2010. - 912 с.
40. Харрис, Энди PHP/MySQL для начинающих / Энди Харрис. - М.: КУДИЦ-Образ, 2005. - 384 с.
41. Ховард, Майкл Как написать безопасный код на C++, Java, Perl, PHP, ASP.NET / Майкл Ховард , Дэвид Лебланк , Джон Виега. - М.: ДМК Пресс, 2014. - 288 с.
42. W3Techs - World Wide Web Technology Surveys // W3Techs URL: https://w3techs.com/ (дата обращения: 20.05.2019).