Картина, която съдържа символ, лого, Шрифт, Графика

Описанието е генерирано автоматично

Технически университет - Варна

Факултет по изчислителна техника и автоматика

    Катедра Софтуер и Интернет Технологии

Дипломна Работа

Разработил:

Ивайло Пламенов Руменов  Ф.№19621627

Съдържание

[I. Увод 2](#_Toc136690741)

[II. Изложение 3](#_Toc136690742)

[1. Обзор на използваните програмни средства и технологии. 3](#_Toc136690743)

[2. Оценка на пазара и конкуренцията 4](#_Toc136690744)

[3. Оценка на конкуренцията 6](#_Toc136690745)

[4. Модел на Ансов приложен за система за контрол на клиентите. 8](#_Toc136690746)

[5. Планиране на тактика за разработка на програмния продукт 12](#_Toc136690747)

[III. Проектиране 13](#_Toc136690748)

[1. Разработка на база от данни. 13](#_Toc136690749)

[2. Реализация на интерфейса за програмираното на приложение 22](#_Toc136690750)

[A. Обзор на нежните технологии и алгоритми 22](#_Toc136690751)

[B. Проектиране на програмния код 28](#_Toc136690752)

[C. Тестване на системата 39](#_Toc136690753)

[3. Реализация на потребителски интерфейс 42](#_Toc136690754)

[IV. Заключение 52](#_Toc136690755)

[V. Използвана литература 54](#_Toc136690756)

[VI. Приложения 55](#_Toc136690757)

# Увод

В предстоящата дипломна работа ще се разгледа в детайли всички особености при създаването на система за контрол на потока от клиенти. Започвайки от гледната точна на използваната технология, която ни е заложена за реализация на идеята. Последван от оценяване нужните функционалност спрямо конкуренцията на пазара и оценка на нужните критерии, която ще се приложи към проекта според пазарните критерии което служи за гаранция че продукта ще е конкурентно способен спрямо целевия пазар. После преминавайки към оценка на конкуренцията и потенциала на продукта да се развие на нашия и чужд пазар спрямо метода на Ансов[[1]](#footnote-1) за оценка на пазара. След тези нужни за доказателство за устойчивост на проекта се преминава към планирането на тактиката за разработка. Тоест главния модел която ще следва за реализацията на програмния продукт. Планът е ключов да се зададе предварително за да може да се следи времетраенето на разработката, както и за пресмятане на нужните ресурси.

Проектирането е следващата глава която се разглежда в детайли. Тя служи за да поясни методите използвани за разработка на множеството части на проекта. Като се започне от базата от данни, тъй като е пресметнато според модела на работа да се разработи първи той. След което се премине към разработката на реализация на интерфейса за програмираното на приложение, или така наречения сървърен интерфейс или модел на бизнес логиката. В него ще бъде разгледа първо обзора на технологии и алгоритми нужни за реализация на проекта и премине към обосновка на методите използвани в програмния код. Реализация на потребителския интерфейс е последната под-глава от главата на проектиране и служи за анализ и оценка на потребителите и съответно изграждане на интерфейс покрай техните потребности.

# Изложение

## Обзор на използваните програмни средства и технологии.

В тази под глава се разгледа в детайл подбраните за задачата програмни средства и технологии нужни за реализацията на проекта. Започвайки от базата от данни, която както името гласи седи като основа на проекта и нейната разработка ще е ключова за цялостния характер на системата. Затова е избрано да използва релационна база от данни. Поради самия характер на релационните бази от данни, които поддържа множество вписвания на данни с възможност за мащабиране в бъдещето. Конкретно е избрано да се използвам Postgresql[[2]](#footnote-2), защото е свободна и отворена релационна база данни с възможности; като много потребителски достъп, гъвкави опции при разширяване и е транзакционно базирана тоест поддържа ACID[[3]](#footnote-3) (Atomicity, Consistency, Isolation, Durability).

Следващия ключов елемент за проекта е самия програмен език на който ще се извършва цялостната функционалност на приложението. За целта ще се нуждае език който да може да борави едновременно с база от данни и да изпраща нуждата информация към потребителя. Затова за изпълнението на задачата е избрано Java[[4]](#footnote-4) версия седемнадесет с Spring[[5]](#footnote-5) версия 3, като фреймуърк който ще спомага за по-устойчиво имплементиране на java езика. Сървъра подбран за реализацията и разгъване на задачата е Apache[[6]](#footnote-6) Tomcat[[7]](#footnote-7). Инструмента за изграждане, който са подбрани за проекта, е Maven[[8]](#footnote-8) заради обвързване на всичките зависимости към програмния продукт. Зависимостите към проекта включват Spring security, за устойчива имплантация на защита на клиенти и потребители, Log4j за всички нужни логвания по проекта и за допълнителна информация при сриване или разкриване на бъгове[[9]](#footnote-9), Hibernate за динамичното закачане на java класове с базата от данни и не на последно място стои Junit Jupiter версия 5, тя служи за тестване на индивидуалните части от код, както и за интеграции с Mockito. Mockito служи за моделиране на отделни нужни части за тестване. Средата за програмиране на проекта е подбрана IntelliJ, избрана е поради нейната лесна и гъвкаво подреждане на проект със всичките му градивни елементи.

Като метод за визуализация трябва да отговаря на стратегията на проекта, която гласи че е нужен уеб базирана визуализация. За целта е избрана Vanila JavaScript[[10]](#footnote-10) съчетана с html[[11]](#footnote-11) страници. Сървъра подбран за реализация и разгъване на визуализацията е Vite версия 4.3.5, който е широко прието поради нейната стабилна екосистема за разработване на уеб приложения.

Съвкупността от тези елементи позволява гъвкавост на начина на подреждане и администриране на системата контрол на клиентите. Както и устойчивост на цялостния проект. Тъй като със силно типизиран език като java когато се открие уязвимост в част от системата или даден пакет и или библиотека то може да се отстрани с лекота без да се налага цялостна преработката. Това е поради самата методология на обектното ориентирано програмиране. С правилна енкапсулация и имплементиране на интерфейси можем да се осигури и че програмния продукт ще издържи на външния натиск от страна на потребители, клиенти и злонамерени личности.

## Оценка на пазара и конкуренцията

В тази под тема се разглежда пазарното проучване, което е предприето преди реализацията на програмния продукт.

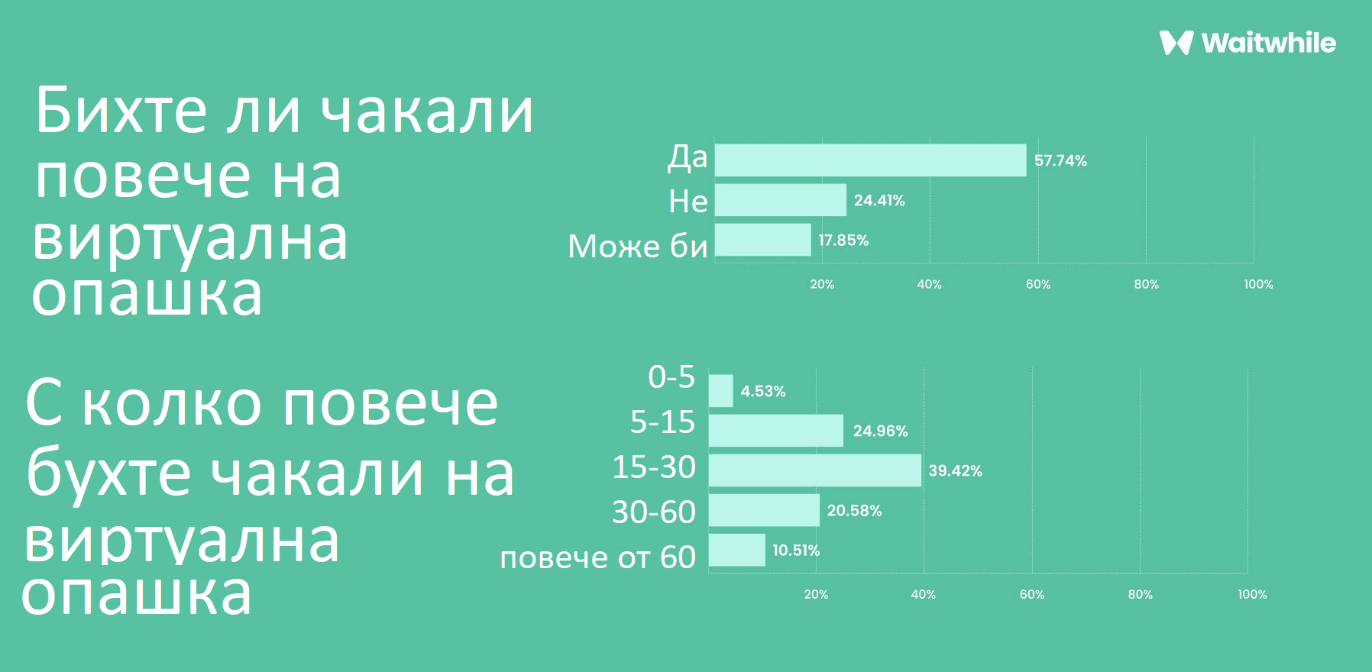
При разглеждането на конкуренцията първо трябва да се погледне над статистика, която стой зад системи за контрол на клиентите и те дали са рентабилни и или полезни. Източниците гласят че в широк характер никой където и да е не харесва да седи на опашка. Според годишната статистика от „*Wait while*“[[12]](#footnote-12) само в Съединените Американски щати клиенти чакат близо дан 37 билиона часа чакайки на опашка всяка година. Както и според същата статистика най-често се прекарва време чакайки на опашки за базови потребности като храна и лекарства (Фиг. 1).

Демографска която е целева за проекта се намира не толкова на най-концентрираните места, където има опашки, а там където една виртуална опашка би ускорила работния процес на заведенията, които имат малко персонал но голям поток от клиенти. Според същата годишна анкета за чакане на опашки хората, който най-често чакат на опашки фигурират около 70% предпочитат да чакат на виртуална опашка. Това е такава опашка, на която имаш билет под някаква форма. Дали тази форма ще бъде физически билет или електронен, на който е отредено реда. Както и хората, които чакат на виртуална опашка, биха чакали средно с по петнадесет минути повече от колкото ако бяха на физична опашка.

Картина, която съдържа текст, екранна снимка, дизайн

Описанието е генерирано автоматично

Фиг. 1. Статистика от анкетирани къде най-често чакат на опашка.



Фиг 2: Статисти за предимствата на виртуална опашка.

Когато се вземат тези данни в предвид можем да оцени проекта за разработка на гъвкава система за контрол на потока от клиенти, като нужен и предпочитан от мнозинството.

## Оценка на конкуренцията

Оценката на конкуренцията представлява разпознаването на услугите и предимствата помежду различните фирми. Анализа на фирмите конкуренти включва: вида на предлаганата услуга, под каква форма се заплаща и съответната цена за поддръжка, целевият им пазар, функционалността на продукта им, набора от клиенти. За правилно оценяване на конкурентоспособността на проекта е изведено анализ на няколко фирми конкуренти, по които можем да оценим програмния продукта. Всяка фирма е разгледана в детайли, след което е представена в таблица ( Фиг. 3.) за ясна визуализация.

Започвайки от „*Esii*“, която е от лидерите на инсталации на софтуера им върху различни билето подаващи машини. Билето подаващата машина представлява физична машина, която разпечатва хартиени билети върху което е записано номера на билета и за коя услуга или гише чака клиента. Фирмата разполага и с теглене на виртуални билети за заведението. И конкретно мобилно чрез уеб сайт за теглене на билети. Фирмата разделили и ограничили целевите демографски на: търговци на дребно, медицинския сектор, публичен сектор и финансов. Пазарният им дял се намира в членките от европейския съюз, Канада и Кот д'Ивоар.

Преминавайки към „Q-net“, които са предлагат: визуализация на опашката, организация на опашката с снимки и ключови думи, подържат редовни клиенти с карата или ( Near-field communication) NFC[[13]](#footnote-13), обратна връзка с купувача на системата, статистика и модифициране на вече вградената логика спрямо изискванията на клиента. Главната им демографска е заложена в финансовия сектор и конкретно с банкови клонове.

Следващата фирма „Q-better“ се откроила на пазара като доверчива фирма за работа с опашки. Системата им поддържа: едновременно менажиране на билето подаващи устройства и на виртуални билети, статистика, уеб запазване на часове както и редактиране на вече създадената логика на опашката за преференцията на клиентите. Фирмата разполага с широка демография, като започнем с търговски магазини, телекомуникационни компании, медицински центрове, публичния сектор, образователни институции и банковите клонове. Клиентите са разпространени в петдесет различни държави на четири континента включвайки: Азия, Африка, Южна Америка и Европа.

Фирмата „Qudini“ е една от най-развитите, когато стане дума за дълбочина на софтуера и неговата функционалност. Софтуера поддържа: множествена връзка с различните клонове на дадена фирма, облачно базирана сигурност, пролежим софтуер на множество операционни системи, готов за употреба с предефинирани шаблони за употреба на опашката, различна имплементация на опашката спрямо клона, бизнес статистика. Фирмата разполага с потвърдили се марки на пазара като техни клиенти. Фирмата е широко скроена но целевия и пазар се намира в Съединение Американски щати. Клиентите им включват но не са ограничени до: Mc Martens, Bass pro shop, Nike, Iqos Sky, Next west, Tesco, East West bank и много други. Оценява се като сериозен конкурент твърдо стъпил на пазара.

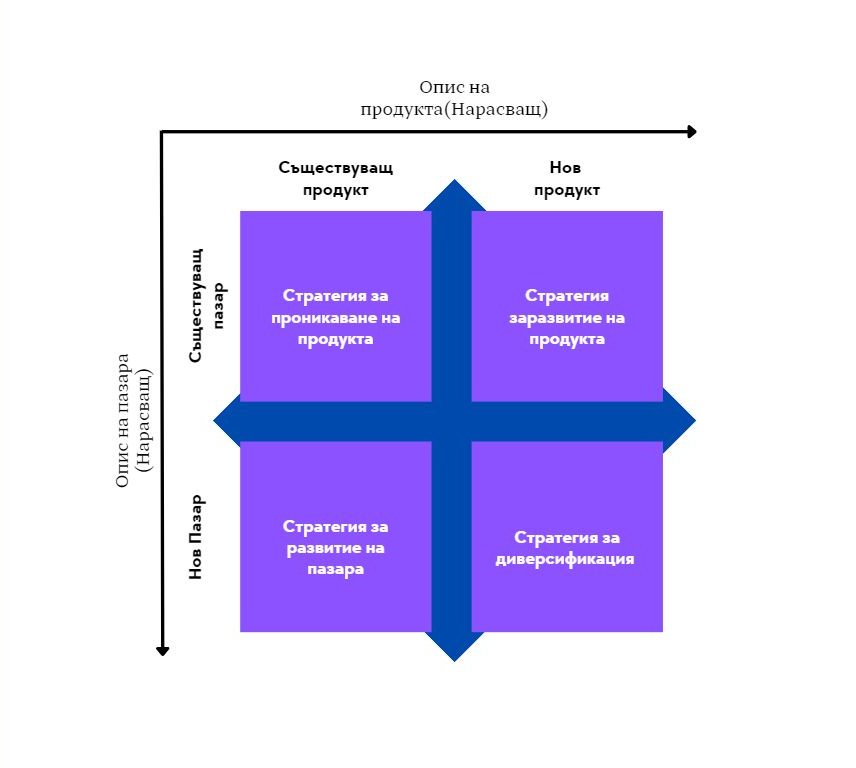
Последна фирма „Kiosk“ е съсредоточен главно в билето подаващите машини, което е сериозен подход към това да си зачислиш демография и да се предържаш към нея. С лимитираната информация предоставена от фирмата може да се оцени че главните клиенти на фирмата са банкови клонове, медицински центрове и търговски магазини. Софтуера предлага визуализация на опашката, като и визитация върху самото билето подаващо устройство.

Картина, която съдържа текст, екранна снимка, Шрифт, линия

Описанието е генерирано автоматично

Фиг. 3. Таблица на конкуренцията, която изобразява различните функции които фирмите конкуренти предлагат.

## Модел на Ансов приложен за система за контрол на клиентите.

Когато се разполага с информация конкретна за фирмите конкуренти, може да се изгради как даден продукт ще изглежда на пазара и той до колко ще бъде конкурентно способен. При такова проучване може да се приложи модела на Ансов употребявайки неговата матрица. Матрицата на Ансов представлява стратегията на фирмата спрямо в кой пазар се намира (Фиг. 4).

Фиг. 4. Матрица на Ансов.

1. **Стратегия за проникване на продукта**

* Приход от закупува: 120$ на месец за всяка инстанция или отдел на фирмата.
* Група на продукта: Системи за масово обслужване.
* Отклонения: Не са нужни апарати за маркиране или вписване на клиенти с билети.
* Новост: Богат избор на видовете обслужване в една система. Тоест клиента да притежава гъвкавост да си нагласи системата по неговите потребности.
* Процент 20-80: Ако приемем че 80% от приходите идват от 20% от клиентите това може лесно да се приложи за системата. Ако заведение, като увеселителен парк или верига кина реши да имплементира масово интегриране на нашата система можем да се прогнозира печалбата. Зa пример може да се вземе кино Арена, която е месното за България пазар верига кина. В цялост имат само седем кина, които по устроените тарифи ще означава по 20$ на месец за всяко кино. Водейки до заключението от 10 080$ годишен приход от една фирма.
* Икономия от мащаба: В системата с фиксирана цена и план за поддръжка не е възможна икономия от мащаба.

1. **Стратегия за развитие на пазара**

* Как може да се потвърди на пазара: Чрез рекламация на множество езици и преминавайки през социалната бариера, можем да предложим продукта с неговата пълна функционалност.
* Темпото на новия пазар: Когато се стъпва на нов пазар продукта има за задача да навлезем с неговия темп. Тоест ако е утвърдено в един пазар, като на пример англоговорещия свят и реши се да премине към хиндоговорещия пазар то ще има изградена инерция да настъпи към по-малък но се така конкурентен пазар.
* Брой участници: Броя участници за дадения продукт е ключов, защото при установили се вградени продукти трудно се сменя. Поради естеството на система, което гласи че ще се ползва безвъзвратно и продължително от клиента. Самата идея за смяна ще възникне или от неефективността на сегашната или от нарастването на потока от клиенти.
* Входни бариери: Софтуера сам по себе си няма много бариери за преминаване. Софтуера се закупува и се употребява. Тоест на клиента може да закупи системата за един месец и да реши дали тя ще е подходяща.
* Уникалното: Поради гъвкавостта на продукта е достатъчна уникално за да се отличи на нов пазар.

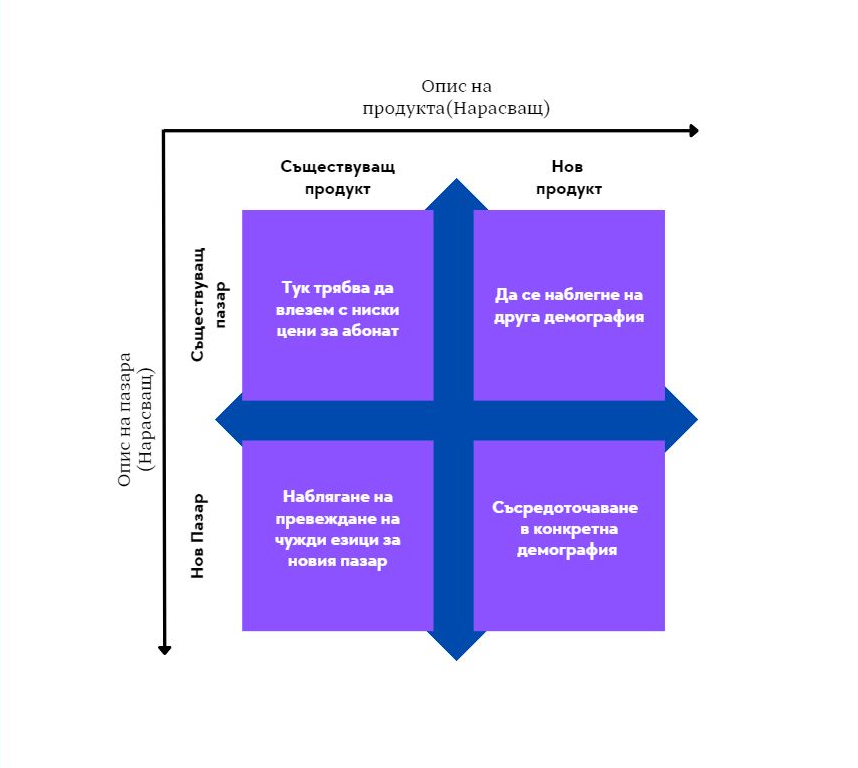
1. **Стратегия за развитие на продукта**

Стратегията за развитие на продукта можем да разгледаме случай където се решава да се надградим над избора на предефинирани шаблонно за допълнително улесняване на конфигурацията на системата. Тоест да се вкарат конкретни демографии към рекламата на продукта. Като на пример: медицински център, банково звено и други.

* Теми на пазара: Типа може да не от стандартните институции, които срещаме в ежедневието. Употребата от болница за справяне с високо заразните отдели. Може да бъде имплементирано в отделение за визити или за приемане на пациент.
* Отразяване на конкуренцията: При отразяването на конкуренцията трябва да се вземе в впрочем процента заетост на конкуренцията в дела, който се желае да се подобри продукта. Не може да се конкурира със строго специализирани програмни продукти, като на пример фирмата „Kiosk“ която строго се е концентрирала върху билето подаващите машини.
* Ниво на границите за пробиване: Нова граница за пробиване може да се разгледа централизиран опит за проникване на продукта или да се подходи пасивно. Тоест продукта да влезе като заместител на по-скъпа и по-специализант продукт.
* Иновативност: Разглежда дали иновативността на системата е достатъчно отделена от конкуренцията на сегашния пазар. Продукта с неговата гъвкавост и приложения се отличава от конкуренцията.
* Раздялата: С разделността на пазара може да се разгледа дали системата ще попадне в такава демография където в момента има спад на пазара или цялостна липса на продукт. Пример: Закона за адекватно обслужване на клиенти в територията на република България. Тук държава задава за всяка от нейните институции да има система за обработване на потока от клиенти.
* Ниво на посоката: Разглежда се ниво на посоката като стандартите които ще са нужни за да е конкурентно способна системата. Пример: Подържа ли се ред на опашките. Дали се поддържа приоритетно ( пререждане) на опашката. Дали е възможно прехвърлянето на часове от една дата на друга и други.

1. **Стратегия за диверсификация**

Разглеждането на диверсификацията на системата се разглежда от страна на нов пазар и нов или подобрен продукт.

* Темпо на растене: Темпото на растене се разглежда от страна на конкурентността и от страна на продукта. Тоест водейки до един среден отчет дали решението за диверсификация е възможно или влиза на сферата на пасивите.
* Наличие на собствена компенсация: Под наличие на собствена компетенция се разглежда адекватността за диверсификацията. Тоест може дадения отрасъл да не подходящ за приложение на системата. Пример: Използване на системата за потвърждаване на автомобили на платени магистрали. Това е пример за липса на умения за потвърждение на регистрационни номера и употребата на камери за целта.
* Ниво на компетенцията: Нивото на компанията е мярката на отклонение спрямо сходните продукти на пазара. Пример: Липсват ни стандарти за незрящи клиенти на системата.
* Възможности на риска: Възможности на риска е калкулацията за възможността на продукта да пробие на пазара. Пример: Ако не се успее да се предоставят наличните машини за даден нов обект можем да се платят неустойки.

Фиг. 5. На фигурата е изобразена реализацията на матрицата за нашата конкретна ситуация.

## Планиране на тактика за разработка на програмния продукт

След цялостна оценка и обзор на продукта като идея. Трябва да се реализира по адекватен, устойчив и бърз начин. За целта се проследяват следните точки:

* Реализация на база от данни според нужните алгоритми за контрол на клиентите.

Базата от данни е ключова за гъвкавостта на система като цяло. А тя ще е организирана по такъв начин че да може да достави достатъчно функционалност на програмния код, който да се използва динамично и с висока скорост.

* Реализация на интерфейса за програмираното на приложение.

Това са самите контроли, които приложението ще притежава. Тоест че ще се организират контролите спрямо нуждите, които клиента ще има когато си настройва системата както ще му е подходящо. Тук влиза и цялостната логика на системата, нейните алгоритми непряко свързани към крайната употреба.

* Реализация потребителски интерфейс. При имплементирането на интерфейса се подхожда по начин на симулиране на характеристиките, които клиента би искал да има при използването на системата.

Когато се следва подобен план, при който към плана където всички модели са ти изолирани един от друг се гарантира устойчивост и лекота при добавяне и тестване на нови функционалности. Модела най-добре описвайки тази тактика е Model-view-controller къде съществува разделение помежду бизнес логиката и интерфейса на крайния потребител ( Фиг. 6.).

Картина, която съдържа текст, диаграма, скица, Шрифт

Описанието е генерирано автоматичноФиг. 6. Визуализация на метода на употребяваната стратегия за изработка на проекта.

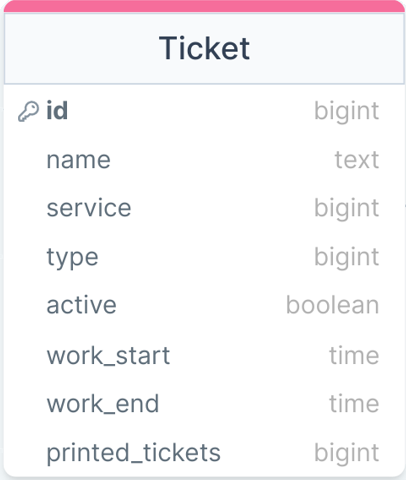
# Проектиране

Проектa се състоеше от три главни части: проектиране на база от данни, проектиране на интерфейса за програмираното приложение и проектиране на потребителския интерфейс.

## Разработка на база от данни.

Разработка на базата от данни започва с вмъкване първо на таблиците, които ще са от първа необходимост. Първите таблици се състоят от тези, която ще седят като ядрото на цялата система. И то именно е таблицата за билетите(Фиг. 7, Фиг. 12 таблица „Ticket“) в тази таблица са всички необходими редова за реализация на билет служещ за обслужване на клиенти. Редовете от които е съчинена включват:

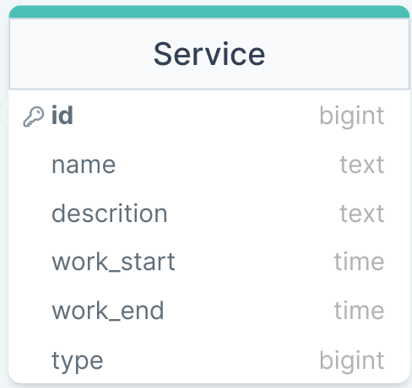
* Име ( „name“) на билета, който служи за неговото наименование;
* Услуга ( „service“) за която работи съответния билет и е реализирана под формата на релация от вида много към много;
* Тип на билета ( „type“) които съществува поради различните възможности на имплементация на един билет. Типа на връзката много към едно;
* Активности ( „active“) репрезентира дали система ще може да разпечатва дадения билет. Типа на реда от таблицата е булева.
* Започване на работа на билета ( „work\_start“) е ред от таблицата, който репрезентира началото чрез час и минути на започвате в формат „час: минути“;
* Край на работа на билета ( „work\_end“) е ред от таблица, който репрезентиран чрез час и минути на край на работата на билета в формат „час: минути“;
* Персонални билет ( „prited\_tickets“) служи за записване на всички персонални билети, който биват изтеглени от клиентите на системата.



Фиг. 7. Таблица от базата от данни отнасяща се за билетите.

След това се взема и в предвид че даден билет ще е свързан с дадена услуга ( Фиг. 8, Фиг. 12 таблица „Service“). Тегленето се извършва точно по услуга така билета който ще бъде издаден се определя точно от тази услуга. По този начин осигуряваме гъвкавост на различен набор от варианти на системата. Когато тегленето се осъществява по услуги може да се настройват допълнително артикули да бъдат добавени към билета, или да се осъществи даден ред на преминава през гишета. Нейните редове включват:

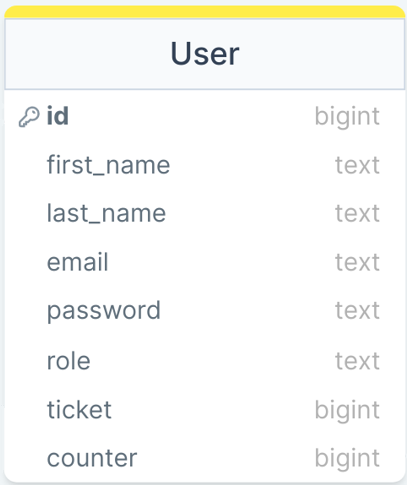
* Име ( „name“) на услугата, който служи за неговото наименование;
* Описание ( „description“) служи ча опис на дадената услуга. Вида на реда от таблицата е текст.
* Започване на работа на услугата ( „work\_start“) е ред от таблицата, който репрезентира началото чрез час и минути на започвате в формат „час: минути“. Тук добавя още едно ниво на гъвкавост когато има работно време и на билета както и на услугата.
* Край на работа на услугата ( „work\_end“) е ред от таблица, който репрезентиран чрез час и минути на край на работата на билета в формат „час: минути“.
* Тип на услугата („type“) е ред от таблицата, който служи за разглеждането на различните типове. Тези типове са възможни и да добавят допълнително усложнение към настройката на системата.



Фиг. 8. Визуализация на таблицата отговаряща за услугите на системата в базата от данни.

Следващата ключова за разработката таблица е тази на потребителя (Фиг. 9, Фиг. 12. таблица „User“). Поради неговото естество тази таблица трябва да служи за различен набор от регистрации, като се започне от администраторски и се премине през мениджърски и работнически и се спре на регистрации на клиенти на системата. Както когато се запазват чувствителни данни то бива да се кодирани но това вече е работа на контролера[[14]](#footnote-14). Редовете на тази таблица включват:

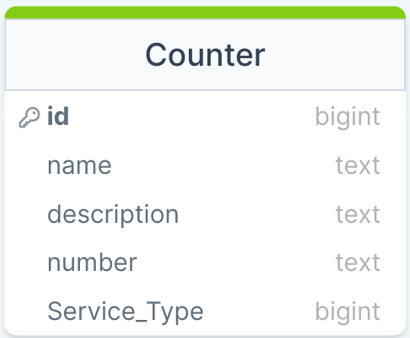
* Първото име на потребителя ( „first\_name“) от типа текст;
* Фамилното име на потребителя ( „last\_name“) отново от типа текст;
* Емайл адреса ( „email“)на потребителя, който служи и за логване на потребителя. Реда е от тип текст;
* Паролата на потребителя ( „password“), коя пристига криптирана и се декриптира от контролера. Реда е от тип текст;
* Ролята ( „role“) на потребителя, която служи за различните функционалности на потребителя, както и се използва от филтъра за ограничаване на достъпа. Реда е от тип текст и когато се приеме от контролера ролята бива реализирана като енум[[15]](#footnote-15);
* Билетите които е обложил фигурират като релация от тип много към много за билета ( „ticket“). Този ред служи за набор от различни статистики, от статистика за продуктивност до изкуствения интелект който решава дали някой ще остане на опашката.
* Гишета ( „counter“) този ред съществува за определяне на потребителя на кои гишета може да се вписва, това е полезно поради различните квалификации и специалности на работниците.



Фиг. 9. Визуализация на таблицата за потребителите на системата в базата от данни.

Гишето, което представлява физично или абстрактно такова, също така трябва да съществува в системата( Фиг. 10, Фиг. 12. таблица „Counter“). Самата концепция за гише както го познаваме, някое място където се викат да бъдеш обслужен. Идеята зад него е че спрямо нагласата на администратора той може да позволява достъп на работници и на услугите. Тоест едно гише може да служи и като стандартно гише както най-често се познава или като нестандартно от рода на район на сервитьор или чекиращо устройство, или за така наречените системи за контрол на обходите, устройство на кондуктор или на охрана на вход. Редове на таблица за гише включват:

* Първото име на гишето ( „name“) от типа текст;
* Описание ( „description “) който служи за кратък опис на функционалности на гишето и за неговото разположение ако това е от значение;
* Номера напишете ( „number“), който служи за допълнително нагласяне на системата. Защото може да има няколко гишета с името „каса“ и с различен номер;
* Сервизния тип ( „service\_type“) е различия, и служи за настройване на кои услуги ще може да обслужва гишето. Това е избрано за да се настройва по тип на услугата отколкото по самата услуга от гледна точка на бързата настройка. Защото може да има стотина услуги по един тип на услугите и така администратора ще му се улесни настройването на системата.



Фиг. 10. Визуализация на таблицата за гише на системата в базата от данни.

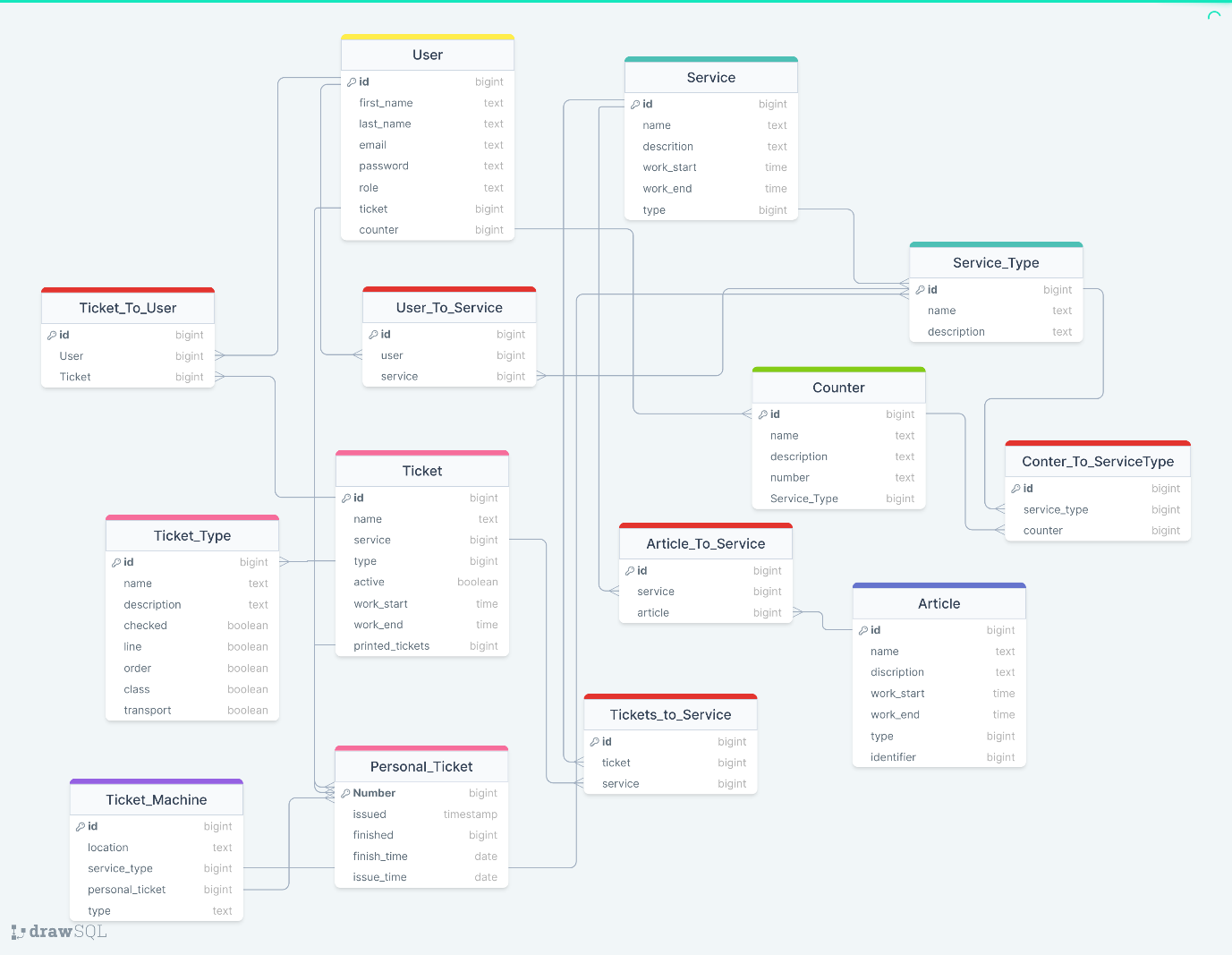
Относно главните елементи, който ще са полезни за системата остана таблицата за артиколи. Която служи за добавяне на малки не обосновани предмети или услуги към даден билет( Фиг. 11, Фиг. 12. таблица „Article“). Като за пример може да са билет за ресторант всяко ястие и напитка да е отделен артикул към финалния билет за масата. Нейните редове включват:

* Първо е име на артикула ( „name“) от типа текст;
* Описание ( „description “), който служи за кратък опис на артикула;
* Започване на работа на артикула ( „work\_start“) е ред от таблицата, който може да служи за ограничаваме от началния часа на работи кухнята на ресторант или дига подобна абстракция. Формата е „час: минути“;
* Край на работа на билета ( „work\_end“) е ред от таблица, който може да служи за ограничаване на работния час на кухня на ресторант като пример. Формат на данните е „час: минути“;
* Реда за тип на артикула ( „type“) служи главно за сортиране и подреждане на различните типове. Реда в таблицата е индексиран за по-бързо търсене.
* Идентификационния ( „identifier“) ред е различен от реда на идентификационния на един запис в таблицата. Този ред служи само и единствено когато даден артикул има собствен номер за идентификация, често срещан пример бива бар код на продукт или номер за инвентаризация.



Фиг. 11. Визуализация на таблицата за артиколи на системата в базата от данни.

Относно базата от данни като цяло са реализирани множество много към много връзки за да бъде успешно реализирана гъвкавостта на приложението ( Фиг. 12). Както и част от успешна реализация на дадена база от данни са стимулационните данни или представянето на примерни ситуации където може да попадне базата. За тази цел е реализирана таблица с примерни клиенти, който биха имплементирали системата по крайно различни начини ( Фиг. 13).



Фиг. 12. База от данни и нейните релации.

|  | Кино | Ресторант | Община | Влак | увеселителен парк |
| --- | --- | --- | --- | --- | --- |
| Service | | | Service | | | |
| name | Зала №3 | Хранене | Акт за Раждане | Варна-София | Въртелешка |
| description | Зала за прожекций | Сервиране на храна  до масата | Свидетелство за  новородените | ... | ... |
| work\_start | 15:30 h | 11:30 h | 8:00 h | 8:10 h | 8:20 h |
| work\_end | 23:30 h | 23:30 h | 17:00 h | 8:45 h | 22:10 h |
| type | Прожекция | Сервиране | Актове | Бърз влак | Увеселително влакче |
| Service\_Type | | | Service\_Type | | | |
| name | Прожекция | Сервиране | Актове | Бърз влак | Увеселително влакче |
| description | Прожектиране на филм | Да се вземе поръчат от  кухнята и да се занесе на клиента | Всякаккъв документ  който се връчва спрямо дадена ситуаци | Блак който е бърз | Тип развлечение  за малки и големи |
| user | Иван, Йордан... | Иванка, Петър... | Йордан, Иван... | Кодукторите и  расиерите | Иван, Йордан... |
| Articke | | | Articke | | | |
| name | Coca-Cola | Боб по селски | Свидетелство от  болницата | Място | Снимка |
| description | Газирана напитка | ... | Документ нужен  за извършване на услугата | Място на влака | Снимка на клиента  докато се забавлява |
| work\_start | 8:30 h | 10:30 h | 0:00 h | na | na |
| work\_end | 22:30 h | 12:40 h | 0:00 h | na | na |
| type | Наливна Напитка | Яхния | Свидетелство | Резервация | Сувенир |
| indentifier | barcode | barcode | idcode | number-of-seat | barcode |
| Counter | | | Counter | | | |
| name | Вход А | Маса В | Гише А | Гише А | Гише А |
| description | От дясна страна на залата | Външните маси | ... | ... | ... |
| number | 1 | 3 | 4 | 8 | 1 |
| service\_type | Зала 3 | Хранене, Напитки... | Актове | Всички | Бъртелешки, VIP |
| Ticket | | | Ticket | | | |
| name | Аватар | Маса В3 | Акт за Раждане | София-Варна | Дневен-VIP |
| description | Филм за извънземни | Билет за сервизната  на масата | Получаване акт за  новородено | ... | Дава възможната да се качваш на всички без да чакаш опашката |
| work\_start | 14:30 h | 10:30 h | 8:10 h | 8:55 h | 9:10 h |
| work\_end | 23:30 h | 17:30 h | 16:50 h | 15:00 h | 22:00 h |
| active | ✔️ | ✔️ | ✔️ | ✔️ | ✔️ |
| personal | 1500 | 3400 | 5000 | 100000 | 12000 |
| type | Entrance | Table order | Queue | Transport | VIP |
| Ticket\_Type | | | Ticket\_Type | | | |
| name | Entrance | Table order | Queue | Transport | VIP |
| description | За билети които  трябва да се проверят преди влизане | Билет, който ще има множество артиколи които ще се сервират на клиент | Тип билет за който ще се чака на опашка за да може да се достъпи гише за обслужване | Билети който служат за транспорти нужди | Билета дава  възможности да се возиш на всички влакчета цял ден без да чакаш на опашка |
| checked | ✔️ | ✖️ | ✖️ | ✔️ | ✔️ |
| base/line | ✖️ | ✔️ | ✔️ | ✖️ | ✖️ |
| order | ✖️ | ✔️ | ✖️ | ✖️ | ✖️ |
| class | ✖️ | ✖️ | ✖️ | ✔️ | ✔️ |
| transport | ✖️ | ✖️ | ✖️ | ✔️ | ✖️ |
| Personal\_Ticket | | | Personal\_Ticket | | | |
| number | 1499 | 3399 | 4560 | 930334 | 152 |
| issued | 14:45 h | 11:35 h | 14:45 h | 9:30 h | 23:59 h |
| finished | ✔️ | ✔️ | ✔️ | ✔️ | ✔️ |
| User | | | User | | | |
| first\_name | Иван | Иванка | Ахмед | Пламен | Иван |
| last\_name | Иванов | Иванова | Ахмедов | Илиев | Асен |
| email | [Iivanov@Kino.com](mailto:Iivanov@Kino.com) | [Iivanova@Restorant.com](mailto:Iivanova@Restorant.com) | [Aahmed@varna.com](mailto:Aahmed@varna.com) | [Piliev@bdj.bg](mailto:Piliev@bdj.bg) | [Iasen@park.com](mailto:Iasen@park.com) |
| password | alabala453 | 012345678 | 12345678 | 12345679 | 12345678 |
| role | user | user | user | user | user |
| service\_type | Прожекция | Сервиране | Актове | Кондуктор | Продавачка билети |
| personal\_ticket | 540 | 10000 | 6000 | 6001 | 7005 |
| Tickete Machine | | | Tickete Machine | | | |
| type | Киоск принтер с плащане | Визуализатор | Киоск принтер | Турникет | Киоск принтер с плащане |
| location | На касата на входа | В главната кухня | Пред входа | Пред входа на станцията | Помежду влакче 1,2,3 |
| servicr\_type | Прожекция | Хранене | Административни, и др. | Преди перона | Въртелешка, Влакче, и др. |
| tickets | 940 | 540578 | 100123 | 120264 | 54003 |
| Обеснение  по рплите на машините | Тук машината служи за  принтиране на билетчета спрямо избора на клиента | В този случай  машината служи да визуализира поръчките на готвачите | В този случай машината  служи единствено за разпределение на опашката за | В този случай машината служи за предварително чекиране на входа и не допуска друг. | Тук отново машината се  ползва за закупуване директно от нея. |

Фиг 13. Примерни данни за различни имплементации на системата за обслужване на клиенти.

## Реализация на интерфейса за програмираното на приложение

Реализацията започва с концепция на какво ще е нужно за да се изпълни лесно и бързо управлението и контрола на клиенти. След дълбоко разузнаване по темата е стигнато до резултата че ще бъде нужно да се приложено теорията на опашките[[16]](#footnote-16) за оптимизирано.

### Обзор на нужните технологии и алгоритми

Теорията на опашките, наричана още и теория на опашките за изчакване ( Queueing theory), е математическа дисциплина, която изисква постъпването, обработката и извличането на заявки или обекти от система с опашка. Тя се прилага за моделиране и анализиране на системи, в които се изпълняват някакви процеси със задачи. Където постъпват по-определен начин и се обработват и извличат от системата.

Основните компоненти на теорията на опашките включват следните елементи:

* Постъпление на заявки: Това е процесът на постъпване на заявките в системата с опашка. Постъпването може да бъде случайно или според определено разпределение;
* Обработка: След като заявката постъпи в опашката, тя трябва да бъде обработена. Времето за обработка може да бъде фиксирано или случайно;
* Опашка: Заявките, които не могат да бъдат обработени веднага, се нареждат в опашка, където чакат своя ред за обработка;
* Излизане: След като заявката бъде обработена, тя напуска системата. Времето за излизане може също да бъде фиксирано или случайно.

Теорията на опашките анализира различни параметри на системата, като средното време за престой в опашката, средното време за престой в системата, производителността на системата, вероятностите за определени събития и други. Това позволява на икономисти, инженери и управленски специалисти да проучват и оптимизират работата на системите с опашки и да вземат решения, свързани с ефективността и капацитета им. За задаване на рака на опашката може да се формулира от функцията му спрямо фирмата която употребява системата. В сценария на системата за контрол на опашките баланса, който търсим е потока от клиенти, да не надвишава възможностите на обслужващия персонал. Менажерът чрез методите му представени от системата за контрол на клиентите ще предостави точно тази преценка, където има достатъчно персонал да обслужи клиентите и не се хабят ресурси. Първи пример, който е представен за най-голямата демография, която ще се възползва от системата ( Фиг. 14.). Това са институция предлагащи административни услуги ( общини, здравни каси, агенцията по вписванията, контрол на автомобилния транспорт и др. ). Разглеждайки диаграмата се стига до заключението че действията предприемани от клиента са малко. Тоест той най-често идва да си свърши една работа, която му е належаща и ако системата е претоварена би дошъл в друг ден. Този случай също е подкрепен от множество проучване и статистка разгледан за проекта.

Картина, която съдържа рисунка, скица, дизайн, илюстрация

Описанието е генерирано автоматично

Фиг. 14. Пример за имплементация тип административни нужди.

Втория пример ( Фиг. 15.) е по-нестандартен но отново е приложим към теорията на опашките. На диаграмата може да се види как системата за контрол на клиентите е употребена за контрол на опашката към различни кино салони. Тоест в този сценарий билетчето бива подадено след негово закупуване а услугите, които се предлагат са индивидуалните филми. Ролята на гишето играе охраната на входа, която проверява и чекира билетите в системата като проверени. В най-идеалния случай едно кино използвайки системата не би имала нужда и от проверяващ персонал а само от билето подаващо устройство за закупуване на билет, и се чекира от автоматичен турникет[[17]](#footnote-17) на входа на киното.

Картина, която съдържа скица, рисунка, диаграма, дизайн

Описанието е генерирано автоматично

Фиг. 15. Пример за имплементация тип кино.

Следващия сценарии разгледан за представяне на теорията на опашките в гъвкавата система за контрол на опашките е пример, в който се употребява от заведение за хранене ( Фиг. 16). Тоест в този случай билетите са индивидуалните поръчки на клиентите, а гишето на което отиват поръчките е района ( групата от маси, за който е отговорен сервитьора). Така теорията на опашките отново се реализира. Тук влиза и артикулите споменати в предишната глава. Артикулите ще се използват точно за такъв вид имплементации на опашката.

Картина, която съдържа скица, рисунка, текст, анимирана рисунка

Описанието е генерирано автоматично

Фиг. 16. Пример за имплементация тип обект за хранене.

Друг елемент, който ще е от високо значение за статистическата гледна точна на приложението и употребата му, е колекциониране на адекватната информация за доказателство че опашката реално оптимизирана. Това е ключов момента. Защото ако не се подберат правилните моменти за показване на потребителя ( менажера) на системата той може да остане в неясност за състоянието и употребата на системата. За решението на този проблем е решено чрез модела на Дюпон за оценяване на ефективности чрез показатели.

Моделът на Дюпон (DuPont model) е финансов модел, който се използва за анализ на приходите на компанията и определяне на факторите, които влияят върху рентабилността и. Този модел е създаден от американската компания DuPont Corporation през 1920-те години и е използван от тях за вътрешни анализи и управление на предприятието.

Моделът на Дюпон разделя рентабилността на компанията на три компонента: марж на печалба, оборот на активите и финансово задействане. Той изразява следната формула:

Рентабилност = (Марж на печалба) × (Оборот на активите) × (Финансово задействане)

Където:

* Марж на печалба (Profit Margin) - отразява способността на компанията да генерира печалба спрямо приходите. Тя се изчислява като деление на чистата печалба на компанията ( нетния доход) на приходите от продажби.
* Оборот на активите (Asset Turnover) - показва ефективността на компанията в използването на своите активи. Той се изчислява като деление на приходите от продажби на компанията на активите й.
* Финансово задействане (Financial Leverage) - отразява степента, в която компанията използва заемен капитал вместо собствен капитал. Той се изчислява като деление на средствата на компанията, финансирани с дълг, на собствения й капитал.

Като се анализира всеки от тези компоненти, моделът на Дюпон може да помогне на управленските екипи да идентифицират и фокусират усилията си върху областите, които могат да подобрят рентабилността на компанията.

Третото и последен метод заложен за осъществяване на система за контрол на клиентите е начин за предсказване на това дали човек от опашката ще остане на опашката. Водейки се по извършеното проучване, статистиката гласи че множество от загубите идват при клиенти които не изчакват реда си на опашка. В система където можеш да се погледне в реално време колко е средното време на обслужване и колко души са на опашката. Така клиента си формулира решението дали ще си изчака реда или ще отиде при конкуренцията. Затова е планирано чрез изкуствен интелект да се реализира това предвиждане. Изкуствения интелект ни позволява да вградим алгоритъм, който вместо човешко същество да предвид вижда натоварване и разтоварване спрямо индикатори в нея.

За реализация на предвиждането на товара в опашката е избран алгоритъма „Дърво на решенията“ защото най-добре описва стъпките, през които човешкия мозък взема решение в подобни сценарий. Алгоритъмът на дървото на решенията (Decision Tree) е метод за машинно обучение, които се използва за задачите на класификация и регресия. Той представлява дървовидна структура, където всеки възел в дървото представлява тест върху някои от признаците на данните, а всеки клон от дървото отговаря на възможни резултати от този тест. Листата на дървото съответстват на класове или стойности на целевата променлива в регресията. Основните стъпки в алгоритъма на дървото на решенията включват:

* Избор на признак: В началото на алгоритъма трябва да се избере признак, който да се използва за тест в кореновия възел на дървото. Има различни методи за избор на признак, като например информационни мерки (като ентропията или при ръста на информация) или Gini[[18]](#footnote-18) коефициент.
* Разделяне на данните: Данните се разделят въз основа на стойностите на избрания признак. Всеки отделен клон в дървото отговаря на една стойност за признака.
* Рекурсивно построяване на дървото: За всеки подмножество от данни, получено след разделянето, алгоритъмът се изпълнява рекурсивно, като избира нов признак за следващия възел и продължава да разделя данните.
* Определяне на листата: Процесът на построяване на дървото продължава, докато не се достигне критерий за спиране. Например, ако всички елементи в дадено подмножество са от един и същ клас или ако са използвани всички признаци. В такъв случай се определя листо в дървото, което съдържа класовете или стойностите на целевата променлива.
* Класификация и регресия: След като дървото е построено, то може да се използва за класификация.

Замислено е и да се реализира само чрез математическите алгоритми без вмъкване на библиотеки, поради рисковете които крие вмъкването на библиотеки. Такива рискове вкупчват: спиране на поддръжка, умислено промяна на кода за да се измъкнат чувствителни данни, точност на алгоритмите. Когато е реализиран някой алгоритъм сам по себе си той може да се променя и доусъвършенстване както е подходящо в бъдещето.

Картина, която съдържа текст, диаграма, Паралелен, Шрифт

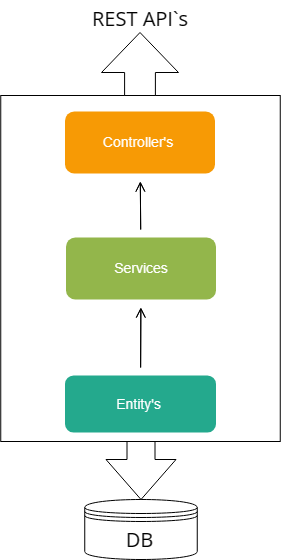
Описанието е генерирано автоматично

Фиг. 17. Клас диаграма на логиката на дървото на решенията.

### Проектиране на програмния код

След като се знае какви ще са алгоритмите заложени за реализация на проекта и се знае как ще изглежда базата се преминава към реализацията на програмния код. Нужна са всички предходни стъпки преди полагането на програмния код, защото се разглеждат крайни случай и множество фактори, които може да се пропуснат ако се започне директно с писането на кода. Така се подхожда и когато клиент представи нужния план. Клиента има много идеи и визия за продукта но не взема в предвид: мащабируемостта, поддръжката, добавянето на фикции и др. Затова се преминава през период от усилено планиране където се мисли и действа с абстракции и се доказват с конкретни случай, така е един от най-силните методи за гарантиране на устойчивост на едно приложение.

Структурата която е използвана е от многослоен тип. Тоест имаме множество слоеве който са слабо свързани и капсулирани за устойчива работа на приложението. Приложението набляга на представителенo на състоянието на трансфера ( така наречената Representational state transfer REST ) методологията, която е приета глобално за репрезентиране на заявки и отговори във уеб екосистемата ( Фиг. 18.). Използвайки REST служа и за лесно откриване на проблеми когато те възникнат, защото когато една заявка не съответства тя връща адекватен код за грешката на проблема. Както и ако приложението премине към друг потребителски интерфейс, то директно може да се свърже към контролите от сървъра без да има нужда от пренаписване или добавяне на други контроли. Като на пример ако приложението реши да премине към мобилен интерфейс на приложението. Новия интерфейс може да се свърже само с контролите на сървъра без да се променя нищо по бизнес логиката на приложението. Тоест REST е предназначено точно за проекти, който се нуждаят от един достъп до ресурси. Където тези ресурси са, бизнес логика, база от данни и комуникационни портали. Самата методология подържа идеята за много слоен подход към разработката на контролерите, който ще се свързват към потребителския интерфейс. Затова е прието да се подходи по три слоен метод за разработка на приложението. Където контролера извършва работата да приема заявки и да връща определени сигнали обратно. Последван от обслужващия слой, който служи за бизнес логика и извикване на базовите обекти. Където базовите обекти служат за съдържане на данни. И последно е връзката към базата от данни и класовете от който се създават различните обекти за бизнес логиката. Когато има многослоен подход може да се покрият всички точки на (Single Responsibility Principle (SRP), Open/Closed Principle ,Liskov’s Substitution Principle (LSP), Interface, Segregation Principle (ISP), Dependency Inversion Principle (DIP) - SOLID) модела.



Фиг. 18. Диаграма на приложената методология при разработване на системата контрол на клиентите

1. **Базов слой на програмния код**

Първите стъпки предприети са към базовите класове с които ще се съхранява информация за различните обекти в програмата за контрол на клиентите. Или така наречените „Entity“ класове. Това са най-простите класове в нашата структура в тях не е заложено да има логика, защото стратегията за разработване гласи че бизнес логиката бива да е на едно нива за да доведе до лесен контрол и промяна при нужда. Не е забранено да има логика на това ниво, тъй като повечето стратегии при ООП ( обектното ориентирано програмиране) са заложели по логически начин. Като например да се състави клас от това ниво с модела на строителя[[19]](#footnote-19) ( Builder prater). Това се прави за да се използват силните страни на ООП. Метода за свързване на тези класове е строго свързан с релационната база от данни, защото при използване на java Spring съчетанието можем да имплементираме инструмента Hibernate. Този инструмент служи за връзване на таблици от базата от данни към класове ( Фиг. 19). Това се прави за да се работи лесно с множеството обекти, който биват създадени по тази методология.

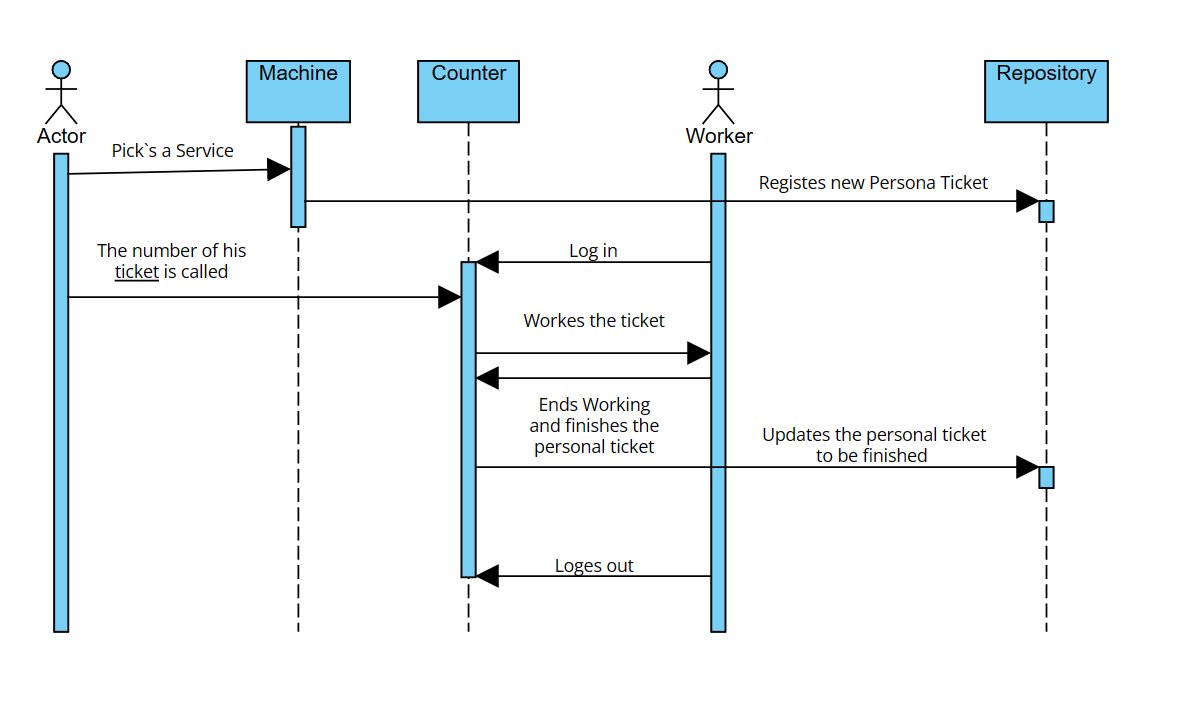
Картина, която съдържа текст, диаграма, План, Паралелен

Описанието е генерирано автоматично

Фиг. 19. Клас диаграма на базовите класове, които отговарят за съдържане на информация.

1. **Бизнес слой на програмния код**

След базовото ниво на обектите служейки за данни следва нивото на бизнес логиката. Тук се реализира реалната логика към опашката и как тя ще работи вътрешно. Главната логика задвижвате цялото приложение ще се имплементира тука включвайки: логиката за опашката ( Фиг. 20), логиката за филтриране на реда ( Фиг. 21), логиката за визуализация на чакащите и логиката за администриране на системата ( Фиг. 22). Където има връзки една помежду друга само там където е наложно и е нужно (Фиг. 23).



Фиг. 20. Диаграма на последователност представляваща логиката зад обслужване на клиент чрез системата за контрол.

Относно логиката по как клиента ще бъде обслужван от системата следдва следната логика. Първо клиента на иституцията, която работи със ситемата, реши за коя услуга ще желае да бъде обслужен то тогава ще му се разпичата съотвентия билет за дадената услуга. Този билет представлява или физичен или виртуален и на него ще бъде разписано: дата и час на издаване, дата и час на годност, избранта услуга, хората чакащи на тази опашка. Избрано е да не му се разписва на клиента колко гишета в момента обслужват тази услуга тъй като би довело до по-висок набор от клиенти напускаяи опашката преди отреденият им ред.

Когато работник постъпи на работа след негово логване, той ще може да започне работа по съотвентия набор от услуги зададени за гишето. Тоест колкото услуги обслужва гишето толкова различни опашки от услуги ще са наредени на него. Така реда на изпълнение на билетите не е по услуга а по ред на изтеглене на билети. Това е допостимо за голям набор от институции но някой имат така наречените приоритетни услуги, който има значение кой за какво е на опашката. Затова когато има приоритен билет той ще предреди вчики чакащи на опашката поради естеството на билета.

Когато служителя е готов за работа с опашката той започва да вика билети. Тук може да възникне проблем при викането на билети, когато двама работници на еднакви гишета с еднаки настроени услуги извикат билет. Затова е изградена синхронизация на викането на билети. След като бива извикан билет от работника за да се яви на гишето той бива записан в базата от данни. Но дадения билет не е отбелязан като приключил. Един билет бива приключен след като се извика следващ. Тъй като служителя може да прицени че не е за съответната опашка или не е заплатил нужни такси. Така клиента със един билет може да се прехвърли на друга опашка. Това позволява на клиента да бива препращан помежди гишета без да има нужда да тегли нови билети и да чака отново на опашки. Когато билет бива прехвърлен от той е с предимство, защото се предполага че си е изчакал времето за чакане и ако отиде отново най-отзад на опашката той няма да си изчка реда и ще напусне или по-зле ще създаде неприятности за работниците.

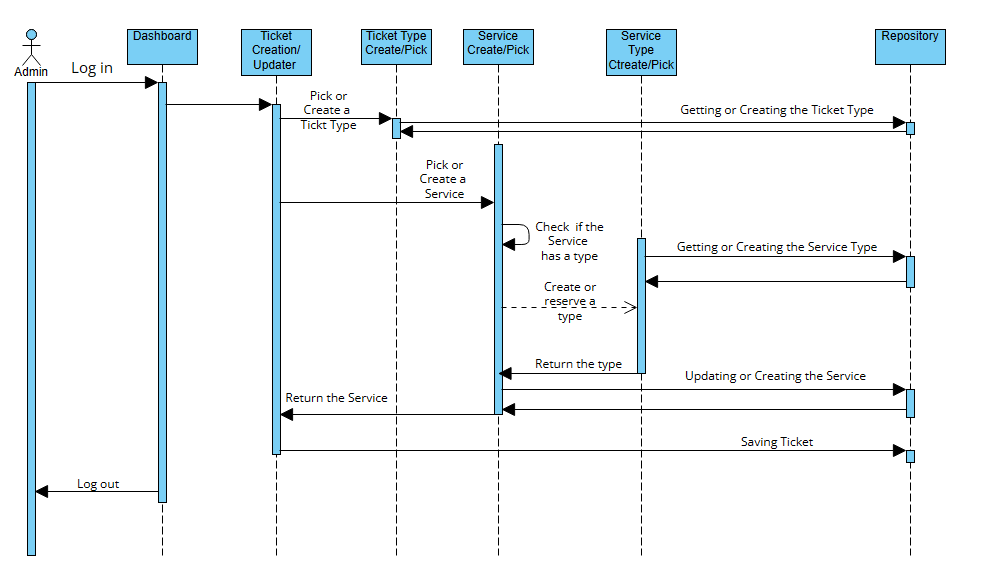
Картина, която съдържа диаграма, екранна снимка, линия, текст

Описанието е генерирано автоматично

Фиг. 21. Диаграма на логиката как един билет при изтегляне и съответно извикване попада на правилното място.

Логиката относно филтрирането на системата за контрол на клиентите и техния ред се осъществява по следния начин. Когато имам множество билети и множество души на опашка и те чакат за едни и същи гишета ( Фиг. 21). Тогава когато работник натисне че е готов да обслужва следващия клиент то системата първо проверява за кой услуги(Фиг. 21 „Favor“) е настроено гишето. След което кой билети се отнасят за тези услуги, тъй като може да има билет с множество услуги, и на края проверява чрез филтрация кой билет е следващ за това гише. Имайки предвид че филтрацията е синхронизирана и няма да позволи грешка при извикването. Тук може и да се наблюдава двуслойни неврони връзки ако се настроен системата по препоръчителния начин. Тоест имам много услуги към много билети се репрезентира като вход на неврона а персоналните билети ( Фиг, 21 „Personal Ticket“) е изхода на неврона. А метода на филтрация е следващата стъпка след неврона. Тоест когато се търси кой билет за кое гише е отново имаме множество входове с един изход. Подобни нива на сложност, която се постига с малко на брой настройки, водят до гъвкавостта на приложението. Затова може да се използва една система едновременно за кино, за ресторант и за административно звено като община и други подобни.

Най-важните аспекти, които са ключови за реализацията за системата и нейната гъвкавост, се свеждат до нивото на приоритет на билета и кога бива изтеглен. Модела позволява лесно да се вмъкнат и резервационни билети.



Фиг. 22. Диаграма на последователност представляваща логиката за администриране на системата за контрол на клиентите

Администрацията на системата е едното нещо, което е заложено потребителя свободно да може да използва така че да оптимизира обслужването на клиентите си. Така оптимизирайки множество елемента от фирмата едновременно. Но за да гарантираме гъвкавост и висок пазарен дял трябва системата да има множество стъпки за избор. Така се среща дилема за колко е най-малкия брой стъпки, който да гарантират едновременно леснота на администрация и гъвкавост на видовете обслужване на клиенти.

Решението предприето от страна на логиката е тя да бъде напълно покрита. Тоест контролите, които могат да бъдат допуснати чрез заявки от потребителя, да бъдат максимално изчерпателни докато потребителския интерфейс да направи процеса на настройка на системата по-поносим и бъз за администраторите на системата. Диаграмата за последователност на фигура 22 се отнася към всички нужни заявки към сървъра за добавяне на нов вид билет. А лекотата и самото потребителско ( администраторско) преживяване да се настрой от потребителския интерфейс.

Самите връзки помежду класовете на това нива са сведени до минимално. Но когато се вземе в предвид че и главната логика на опашката е в това ниво. Клас ( Фиг .23 „QueueService“) отговарящ за множество от логиката в системата е свързан с нужните за реализацията на логиката при обслужване на клиентите на опашката. Но при разглеждането на връзки с други нива на системата можем също да открием връзки за успешното осъществяване на логиката на опашката. На това ниво се намират и повечето функции, който се използват за работа с билетите и включват функции за:

* Теглене на билети по услуга;
* Визуализация на билети;
* Извикване на следващ клиент на опашката;
* Визуализация на услугите.

Тези фикции са подготвени по такъв начин че да бъдат лесно приети от следващото ниво над тях. Тоест параметрите, който връщат, са структурирани или като базови класове или като класове данни ( payloads[[20]](#footnote-20)) за по-лесна работа от контролера.

Картина, която съдържа текст, Шрифт, Паралелен, черно и бяло

Описанието е генерирано автоматично

Фиг. 23. Клас диаграма на всичките сервизи, който са употребени в разработката на системата за контрол на клиентите.

1. **Контролен слой на програмния код**

Контролер новото е директната връзка с потребителския интерфейс и представлява корекцията, през която всеки един желаещ да комуникира със системата за контрол на клиентите ще премине. Връзката със сървъра става след като той бива стартиран и инициализиран на даден домейн. След това за всеки контролер си има адрес ( Uniform Resource Locator-URL) на който се приемат и изпращат всички заявки към сървъра. Тези заявки спрямо адреса могат да се формулират по множествено начини, но за разработката на проекта е предържано към стандарта за писане на заявки.

Https:// име на модула/ версия на котролера/ име на пакета/ име на контролера/ променлива ако има такава

Пост ( Post) заявки служат за регистриране на нещо ново, тоест да се дови към системата. Този вид заявки служат главно при администрирането на системата, където ще се създават нови услуги, билети, потребители и др. При пост заявките е нужно и да има тяло ( body[[21]](#footnote-21)) на заявката и то да бъде в стандарта на JSON (JavaScript Object Notation). Този вид тяло на заявката е стандартизира и се използва и при отговор на сървъра. Всички пост заявки от страна на администриране на системата се нуждаят от автентикация.

Гет ( Get) заявките служат за вземане на конкретна информация от сървъра и са отново стандартизирани. В проекта се използват за главно за да се визуализира информация от базата относно системата за контрол. Те не се нуждаят от тяло на заявката но често получават списък от желания обект. Всички гет заявки от страна на администриране на системата се нуждаят от автентикация.

Пут ( Put) и заявки за изтриване съществуват и при администрацията и при обслужването на опашката на системата. Пут заявките служат за поправяне на обект или да се добави нещо към този обект. Като за пример пут заявки е използвано когато даден клиенти е обслужен и му се приключва билета. За да се оформи такава заявка най-често има тяло и член променлива в адреса на заявката. Пример:

localhost:8080/api/v2/counter/add/ticket/:favorId/:counterid

Сигурността на системата се осъществява чрез вградените инструменти от Java spring технологията и конкретни зависимата библиотека „Spring security“. Която позволява имплементирането на множество филтри. Филтрите който са осъществени за поетка включват:

* Филтър за разграничение помежду работник и администратор ( по роля);
* Филтър за JWT (JSON Web Token) автентикация;
* Настройка за предаване на информация през няколко източника (Cross-origin resource sharing CORS);

Филтрирането чрез токени е често срещани и широко използвано. В токените можеш да слагаш множество информация, която е кодирана, и директно свързана с информацията в user базов клас.

Картина, която съдържа текст, Шрифт, разписка, Паралелен

Описанието е генерирано автоматично

Фиг. .24. Клас диаграма на всички контролери.

### Тестване на системата

Тестването на системата за контрол на клиентите и тестване в дълбочина е нужна, тъй като е заложено множествено функционалност и гъвкавост при множеството от ситуации, в които ще бъде разгъната. Затова за осигурено цялостност на тестовете и е наложително те да включват: единични тестове ( unit tests), интеграционни тестване (Integration tests), от край до край тестване ( end-to-end testing), тестове при разгъване , функционални тестове ( Functional Testing), тестване на производителността ( performance testing),тестове при взаимодействие с базата от данни. Тази поредица от тестове е високо ценена и употребяема в практиката при разработване на приложения.

Единичните тестове представляват такива тестове, който служат за тестване на малки части логика по кода. Тоест най-често се извършва от програмиста, които е написал кода първоначално. Това се прилага така защото кода който е написал може да няма нищи общо с бизнес логиката но да играе роля при цялостната употреба на приложението. Този вид тестове са ключови, защото те трябва да се покрият в множество сценарии с множество параметри. Така гарантират че с продължението на писане на код то тези класове и функции, който се тестват, ще са устойчиви и на мащабируемостта. Единични тестове могат да се извършват по начин по който да помогнат за цялостното тестване на системата. Тестовете извършени по този начин, за системата за контрол на клиентите, са се вградили постепенно през всяка нова итерация в функционални и интеграционни тестове. Множество индивидуални тестове са извършени върху проекта. Започвайки от базовите класове, последвани от индивидуални тестове за бизнес кода.

Функционални тестове представляват такива тестове, които служат за до моментния етап на направени промени към кода. Тоест те са променят за всеки цикъл ( спринт) през който преминава програмния продукт. Те служат за валидация на кога преди да бъде добавен към работещата версия до момента. Както и да се измери работната скорост и успеваемост на проекта. Тук се тестват едновременно фундаментални аспекти на програмата, както и функции на програмата, които играят роля на допълнения и не са градивна част от ядрото, но са нужни за цялостната завършеност на продукта. Функционалните тестове по проекта са извършени в период от няколко дни за да се провери как различни елементи от системата ще работят заедно. Това се постига когато се вземат единичните тестове и се навържат един към друг за да формулират един строго последовател тест.

Интеграционните тестове тестовете представляват тестове на индивидуалните нива, като на пример тестване само на заявки към базата от данни. Този вид тест служи за да гарантира че дадения слой от проекта може да работи самостоятелно ако му се подават правилните данни. Така се гарантира устойчивост на кода, който се пише на всяко едно ниво, и че ако стане проблем ще може да се сведе до това че или не е изпратен правилен сигнал или че сигнала не е подходящ за сценария. Интеграционните тестове е наблегнато в проекта поради метода и идеологията, които са избрани за неговата разработка. Естеството на слоевете на програмата гарантират че ще има множество интеграционни тестове. За системата за контрол на клиентите са се извършили интеграционни тестове на нивата за: заявки към базата от данни, към сервизите на системата (Фиг. 23), към контролерите на системата (Фиг. 24) и към потребителския интерфейс. Този вид тестване може да покри до 97% от функционалния код на един клас. При писането на такъм вид тестване трябва да се има в предвид как едни клас и неговите функции ще реагират към набор от правилни и неправилни сигнали. Така се постига с високо покриване на кога и неговите редове.

Край до край тестване представляват такива тестове, които тестват едно приложение от неговото начало до неговия край, или така нареченото тестване по сценарии. В този вид тестване се поглежда с една идея как крайния потребител би използвал приложението. Тоест гледат се множество сценарии от такива, който са правилни до крайно грешни. Когато се изпълняват подобен вид тестване се разбира и как крайния потребител би имал проблем с цялостното потребителски изживяване. За тази цел е използван програмата „Postman“, която служи като инструмент за директно тестване на крайните връзки на контролера на сървъра. Поредността от действия играе голяма роля при писането на подобни тестове, тъй като се предполага че и клиента ще следва тази логика при използването на системата. Поредността ни е заложена като фундаментална за употребата на приложението ( Фиг. 25). Видовете край до край тестове извършени по системата за контрол на клиентите са:

* Тест с гише с множество услуги- Тук се гледа система, която и мама еднакви услуги, който отиват на еднакви гишета. Тоест се представят нови услуги но се обслужват от същи персонал.
* Тест с множество услуги и множество гишета препокривайки се- Тук има гишета с различен набор настроени услуги който могат да бъдат обслужени но се препокриват с други гишета, който имам сходни но не и пълния набор от сходни услуги (Фиг. 26).
* Тест с множество услуги и множество гишета без препокриване- Тук по подобие на предишния тест имаме множество гишета и множество услуги но те не се покриват по никакъв начин.

Картина, която съдържа текст, екранна снимка, софтуер, Мултимедиен софтуер

Описанието е генерирано автоматично

Фиг. 25. Изображението на настройката на автоматични тестове преди тяхното стартиране в опреден ред.

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

Фиг. 26. Изображение на автоматичен тест, който включва инициализация на система с множество гишета и множество услуги тествано как обслужва с множество билети.

## Реализация на потребителски интерфейс

Разработката на потребителския интерфейс е ключова за реализацията на успешен проект. Потребителския интерфейс е първото и последното нещо което потребителя вижда когато борови с приложението. Тоест изисква се наблягане върху фините части на човешкото взаимодействие с програмен продукт. Когато се работи по потребителски интерфейс трябва да си отговори на няколко въпроса:

* За кого правим интерфейса;
* Каква работа ще извършва интерфейса;
* Как ще контактува човека с интерфейса;
* Кои принципи на дизайна ще са нужни за реализацията на интерфейса.

Когато се отговарят всичките тези въпроси и се приложат върху дадени персони за част от система. Започва да се изгражда потребителското изживяване (User experience-UX). Това относно как човека като цяло ще използва системата за контрол на клиентите дали това ще бъде в аспекта на работник или на клиенти на системата. Когато правилно се помисли как човек общува със системата това води само и единствено до по-бърза употреба на системата ( user efficiency) от всички страни.

Започвайки дизайна на потребителския е взето в предвид и начина на имплементиране на програмен код. Тоест употребата на JavaScript и html. Когато се има и това в предвид може да се започне да се разглеждат сценариите на интерфейса. След като си знаме интерфейса на контролера как изглежда и как да го достъпи има свобода на креативността при дизайна на потребителския интерфейс. Следва да се открие клиентите, които ще употребяват системата. Това е постижимо чрез употребата на персони[[22]](#footnote-22).

Първата персона която се разглежда е тази на типичен работник на гише (Фиг. 27). При имплементирането на тази персона е приложен принцип че човека употребяващ системата е средно мотивиран да работи но с над средна грамотност с технологиите. В този случай персоната ще притежава динамичен интерфейс къде ще може да вика номера на билети и ще ги обслужва подобаващо. Тоест трябва от гледна точна на дизайн да има ясени шрифтове на текста, както и леки за очите цветове защото човека ще работи дълги часове пред тази система.

Картина, която съдържа текст, компютър, човек, екранна снимка

Описанието е генерирано автоматичноФиг. 27 Персона на работничка на системата за контрол на клиентите.

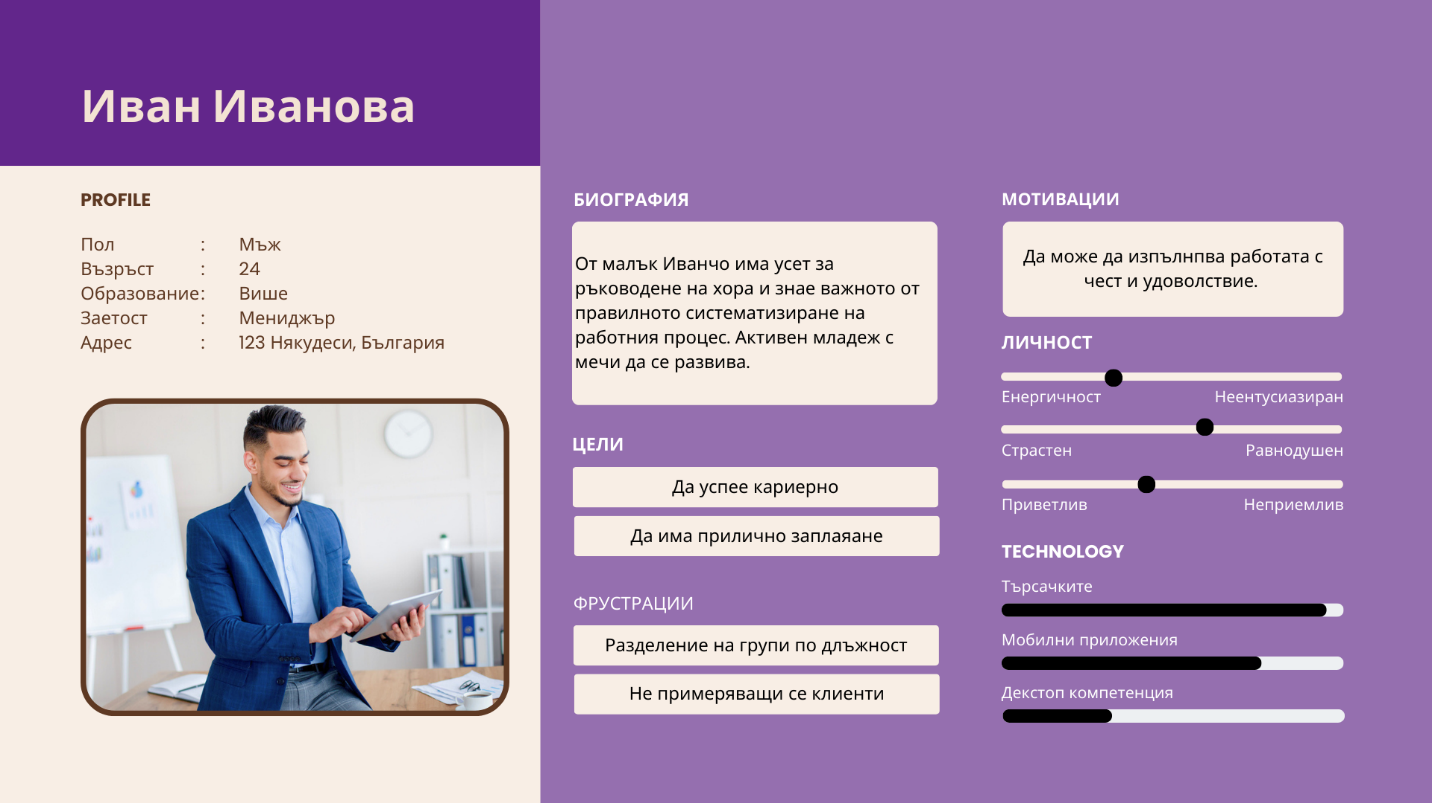
Следващата персона, която е разгледана, е на касиер (Фиг. 28). Подобна на персоната на работник на гише тази персона представлява работник, който ще използва системата за викане на клиенти и съответно обложвайки ги. Но в този случай поради възрастта на служителя се вземат на предвид технологическата компетентност, която е ниска. Тук е взето и решението за опресняване на начина на извикване на клиенти към гишето.

Картина, която съдържа текст, екранна снимка, човек, Уебсайт

Описанието е генерирано автоматично

Фиг. 28. Персона на работничка на гише тип каса.

Следващата персона, която се разглежда е от страна на оперативния персонал или така наречената организация на контрола ( Фиг. 29). Ролята на мениджъра е ключова за реализация на теорията на опашките, която гласи че трябва се постигне баланс помежду потока от работа и обслужващия персонал. Тоест преценката за персоната е критичната. Тя трябва да репрезентира примерен персонал в ролята на мениджър. За това е прието тя да е на мъж в началото на своята кариера, поради нужната квалификация за изпълняването на ролята. Персоната е високо мотивирана и техническо компетентна. Тоест можем да се очаква да се възползва от статистиките в тяхната цялост и за това да се наблегне върху тяхната разновидност. Характерно за тази работна роля е и следенето на персонала тоест можем да изведем че ще е желателно да се добави колко работни часове отделя един работни за даден билет и колко билета обработва за една негова смяна. От там персоната може да си редактира и до настройва опашката така че да е най-ефективна. От тази персона също може да се изведе факта че ще бъде от полза да може да се следи опашката от дистанция чрез мобилно устройство, тоест потребителят на интерфейса ще очаква той да е гъвка и четим от различен набор устройства а не само от компютър. Също така е заложено да се добави модела на Дюпон заради неговата популярност в менажирането и организацията на контрола. Така само от една персона се извеждат много изисквания за интерфейс на програмния продукт.



Фиг. 29. Персона на мениджър.

Последна персона, която е преценено че ще е от нужда за оператор, който върши двойна роля. Това е точно работник тип охрана, който едновременно проверява билети и следи за реда ( Фиг. 30). Подобна е ролята на кондуктор на влак или автобус но поради пазарния дял, който цели програмния продукт, е преценено да се изработи персона за охрана заради нейната разпространеност в пазарния дял. Представяната персона на охранител е пример за чекиращо гише където работника е мобилен и трябва едновременно да проверява билети и да следи за реда. Тази ситуация показа че потребителския интерфейс трябва да бъде лесно четим и разбираем на пръв прочит. Лесното разбиране и четене на един потребителски интерфейс е изграден от: четими шрифтове, белите пространства и цветна координация. Тоест че за персона като охранителя не му е нужен много функционалност а бърза рабат. За това е подбрано когато даден работник е на гише което чекира билети то да сигнализира за редовен билет в зелено а за грешен, изтекъл или нередовен билет да сигнализира в червено. Метода за чекиране може да е по различни видове дали по номер на билет или баркод на билета. Тоест при получаване на билет, дали ще бъде закупен или просто изтеглен, то той ще отговаря за реда и контрола на опашка и след това някой като собственика или мениджъра може да погледне статистиките към дадените маркирани билети. Взето в предвид и техническата грамотност на персоната така че да е подходяща за бърза и устойчива работа.

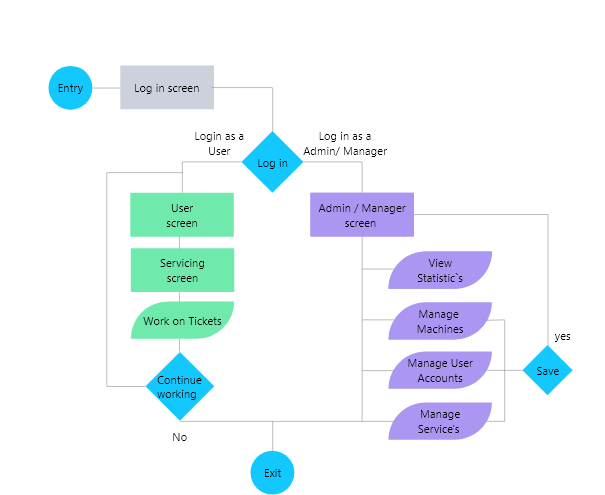


Фиг. 30. Персона представяща охрана на обект и контрол на опашката.

След вземане на предвид широката гама от персони, които са разгледани, можем да се приложат при конкретни функционалности както и тематични забележки към потребителския интерфейс. Едно от решенията, които е взето да не се разглеждат конкретни персони за клиентите на системата за контрол. Поради естеството на системата се подразбира че клиентите ще са от всички различни прослойки на обществото и с различни предпочитания. Тук предпочитанията не играят роля, тъй като те по един или друг начин ще използват системата за да достъпът услугата които желаят. Но това не е знак да се предприемат дръзки решения за дизайна от страна на клиента. А даже да се подходи по метод, който масовото общество ще разбере. Това включва и управление за незрящи, като звукова и допирна сигнализация. Така един метод за прилагане на системата за незрящи ще е да се съобщава звуково когато се извиква билет и някои билети да имат дупки в тях така че да са разчети ми чрез Брайловата[[23]](#footnote-23) азбука.

Относно дизайна на физичните билетите те ще включват дата и час на издаване, име на услугата, кои гишета ги обслужват и колко души има пред тях. За метода на разпечатване на билети закупилия системата ще има избор да ползва какъвто си поиска принтер за разпечатването на физични билети. Системата автоматично ще взема зададения принтер по подразбиране за разпечатване. А за допълнителни настойки ще може да се достъпи чрез административния пален на системата.

От другата страна има и виртуалните билети, които ще са достъпни чрез уеб страница, където са изписани всички услуги на системата така че клиента да може да подбере желаната услуга и да си изтегли билет. След тегленето на билет клиента е препратен на уеб страница където изписва номера н билета и чакащите в реално време. Това позволява на клиента да може да следи реда на билетите без да бъде физически там. Когато билета бива извикан клиента бива информиран че му е реда да се яви на гишето, което го е извикало. Както и ако клиента присъства на място ако е настроено информативен дисплей той ще е бъде казан на глас от информативния дисплей. Информативния дисплей представлява екран ( телевизор или уеб страница), които е настроен да показва реда и на кои гише те биват викани.

Преди осъществяването нужните уеб страници, които ще се разработени, първо се преглежда реда в които те ще са употребени. Вземайки персоните в предвид ще разгледаме реда по следния начин (Фиг, 31). Първо разглеждайки от страната на работния персонал и административния. След логване в системата за контрол на клиентите спрямо ролята те са прехвърляни или към административния панел или към даденото гише на което ще работят. Ако потребителя е клиент той ще бъде преведен към гишето което е предназначено. От там ще може да обслужва различните билети настроени за гишето, ако има изтеглени такива. Но ако потребителя е администратор той ще е преправен към страницата за редактиране на елементите на опашката ката: гишета, услуги, артиколи, машини ( ако има такива) и статистиката по системата. След като се знаем всичко това може да се премине към разглеждането на страната отговорна за теглене на билети.

Фиг. 31. Диаграма на потока за администратор или работник.

Относно страната на клиента, която служи за теглене на билет по услуга и съответно изчакване и следене на реда на чакащи за гишета за да може да бъде обслужен ( Фиг. 32). Тази част от системата е най-линейната от всичките. Тук клиента или си изчаква реда или си тръгва. Но когато сме помислили по всички други елементи по системата тази част на клиента е най-лесна. Така когато е планирано това преживяване да е оптимизирано за бързина и минимална объркващия за клиента се очаква само позитивни резултати от употребата му .

Картина, която съдържа текст, екранна снимка, диаграма, Шрифт

Описанието е генерирано автоматично

Фиг. 32. Диаграма на потока за теглене на физични и виртуални билети.

Анализа, които е извършен по персоните и те как ще навигират приложението, е от ценно значение при синтезирането на дизайна на потребителския интерфейс. Инструмента, който е използван за изработка на идеята на интерфейса преди да се започне неговото програмиране е Adobe Xd. Решението да се използва е прието поради неговата гъвкавост и употреба на пазара и в обществото на дизайнерите. При прилагането на схемите е започната от преценяване на цветовата гама и шрифтовете, който ще се използват в приложението от гледна точка на консистентност. За избора на цветове е подходящо те да са в по-студената гама, затова е избрано постелено синьо за доминиращ цвят съчетано с бяло за празните пространства и на края бебешко синьо за сигналните елементи ( Фиг. 33). Подбран е по модела 60-30-10 поради неговата утвърдена репутация в сферата на потребителските интерфейси. Модела представлява гама от три цвята като 60% е един цвят е доминантен в мащаба на интерфейса, 30% се използва за предаване на промяна или различие по между елементите в интерфейса и 10% се използва за сигнални елементи където искаме да привлечем вниманието на потребителя. Видове сигнални елементи в един потребителски интерфейс биват бутони, прогрес барове и сходни елементи.



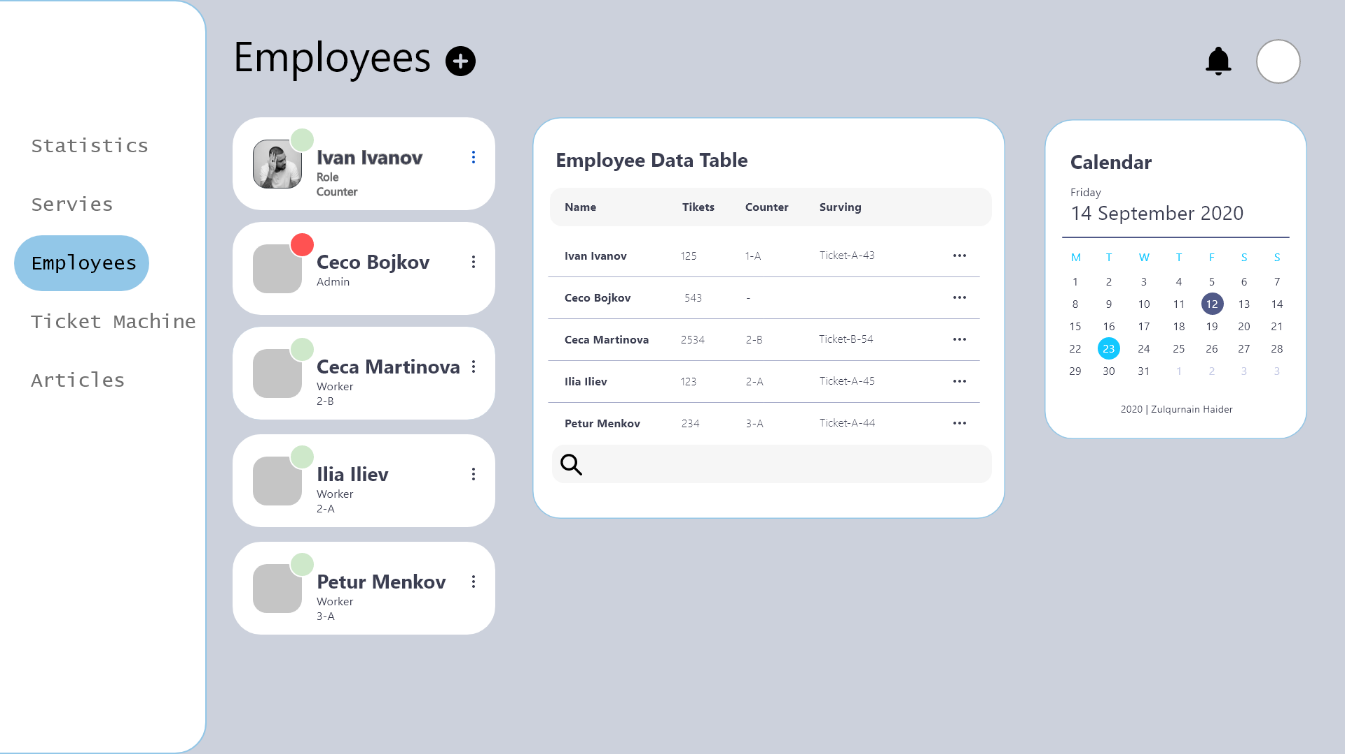
Фиг. 33. Гамата от цветове използвани за разработка на потребителския интерфейс.

След избора на цветове се преминава към избора на подходящи шрифтове ( Фиг. 34). Първия „Nirmala UI“ се използва в чисто черен формат, като заглавия на страници и опции в менютата. Избран е зареди неговия брутализъм[[24]](#footnote-24), което води до бързо разчитане и открояване на различните символи един от друг. Следващия шрифт „Lucida console“ е комбиниран със сив цвят за да се открои от черния шрифт споменат преди и да покаже допълнение в моменти където дадена информация е второстепенна към главна в черно. Последния шрифт „Bilo“ е съчетан с главния шрифт там където е нужно да се покаже тънка разлика, като например различните видове статистически графики ( Фиг.35) или при изброяването на множеството работници ( Фиг. 36.). Представени са и множество от дизайна в приложната част на разработката. Там са представени идеята за цялостната визия на потребителския интерфейс и може да се стръвни как е пренесено синхронизацията на всичките уеб страници за цялостна консистентност на интерфейса.

Фиг. 34. Използваните шрифтови семейства с техния размер и цветове.

Картина, която съдържа текст, Шрифт, екранна снимка, дизайн

Описанието е генерирано автоматично

Фиг. 35. Статистика от дизайна на потребителския интерфейс.

Фиг. 36. Реализиран дизайн на страницата за побойници на системата за контрол на клиентите в Adobe Xd.

Програмния код използван за реализация на потребителския интерфейс е написан на JavaScript и е разгънат на Vite сървър, който компилира и пуска сървъра на локален източник. Страниците, който са представени чрез Adobe Xd са реализирани чрез чист html и SCC ( Cascading Style Sheets). Използвайки чист html и scc позволява много подходи за дизайна на уеб страница и нисък размер на крайния продукт. Но липсва гъвкавост, която други платформи като React[[25]](#footnote-25) или Angular[[26]](#footnote-26) предлагат.

Реализацията на малкото логика, която интерфейса има, се е извършва чрез JavaScript класове и функции строго свързани към дадени действие на потребителя. Заложено е в проекта потребителския интерфейс да има възможно най-малко логиката, тоест единствената логика да е свързана с приемане на заявки от поробителя, за обработка на съобщения от сървъра и за визуализация на информация. Всички заявки, който извършва приложението от и към сървъра се синхронизират за да не се изпуска никаква информация или да се покаже стара или ненужна за съответния сценарии информация.

Визуалното тестване на потребителския интерфейс е процеса по който един потребител би виждал системата и би я употребявал на множество уеб браузери и устройства, като мобилни телефони, компютри и физични обекти. Визуалното тестване на системата се случва по следния начин. Първо за да се стартира сървъра отговорен за бизнес логиката и базата от данни. След това се настройва да симулира реална употреба на системата, като на пример административна институция. След което се стартира потребителския интерфейс и се инспектира визуално на множество различни платформи дали отговаря на изискванията по, който се водим като проектанти. Използвайки сървър от рода на Vite позволява да се редактира един уеб потребителски интерфейс в реално време и да се опреснява след запазване на съответните промени. Това спестява презареждането на сървъра и съответно съкращава времето за разработка. Този метод работи само когато искаме да погледнем дали нещо е правения цвят, шрифт или подравнено адекватно. Но за по сериозни проверки, като на пример автентикация на потребител или изпращане и приемане на правилните заявки, се използват автоматични тестове.

Автоматично тестването на потребителския интерфейс се извършва чрез употребата на инструменталната библиотека на Selenium, която е широко използвана в сферата на тестване на потребителски интерфейса. Инструменталната библиотека може да бъде ползвана на множество езици, но за целта на тестване на системата за контрол на опашките е подходящо чрез java езика. Процеса на автоматично тестване на потребителски интерфейс е сходен с този на тестване на логика в класове или фикции. След зареждане на системата и нейната настройка с писането на директен код може да проверим как един интерфейс реагира в набор от различни ситуации спестявайки време за ръчна проверка. Подобни тестове са полезни когато има множество промени по дадени елементи но без те да си губят функциите. Тоест имаме функционалност която не се променя а промените биват чисто визуални.

# Заключение

Основните характеристики на системата за контрол на клиентите в нейната същност е проектирана на оптимизира начина на работа с клиенти по такъв начин че клиента да получава повече както и клиента. Тоест по вече изградени методики в човешкото битие, като това да чакаш на опашка, да се обърне по такъв начин че да е от полза за всички. Така характеристиката на проекта представлява система за оптимизиране работния процес без да се нуждае дадено предприятие или институция от допълнителна работна рака и да може да поема по-голям набор от клиенти по-ефективен и без загуби.

Особености на представеното инженерно решение се състоят в начина по който се интегрирали в проблема с изчакване на реда на опашка. Първо идеята че хората чакат на опашки за множество услуги навежда към теорията на опашките, която гласи че потока от сигнали ( клиенти в този случай) ако бъде систематизирана, то с по-малък работен състав ще може да се обработят сигнали по енергийно ефективен. Когато това се приложи в практиката се получава точно система за контрол на клиентите където клиентите са систематизирани по услугата, за която желаят да бъдат обслужени.

Предимства на такава система за контрол на опашките е строго обвързана с икономическите характеристики на едно предприятие. Тоест от малък персонал на рака да се вадят повече брутна печалба. Предимство е и за менажиращия персонал където можем да имам много по малко менажери за повече обслужващ персонал. Благодарене на статистките и следенето в реално време. Недостатъци при подобни система е че потребителя закупи системата ако не настоя правилно системата или прецени че за неговата институция. Даже и предприятие се нуждае от система а не разполага с нужда от нея а само усложнява нещо по-просто от колкото представлява.

Резултати от експерименти гласят че множество фирми днешно време се нуждаят от подобна система. Това се дължи поради навалящия набор от работен персонал, който е склонен да работи административни или обслужващи роли. При тестването на системата в различни сценариите можем да видим как един работещ персонал може да обслужва до 15 души чакащи за услуга то 35 минути за чакане и да няма загуби от клиенти. Както едни огромен бонус за приложението е широкия набор от приложения, който са възможни за този един продукт.

По-нататъшна работа върху решаването на зададения проблем ще лежи върху това да може продукта да се развива в множество нации където стандартите за обслужване варират от нация към нация. За конкретни технологически добавки към кода ще се наблегне над добавяне на различни модули за допълнителна логика при по-специализирани системи.

# Използвана литература

* Статистически анкети за чакане на опашка:
* Wait while, Consumer Survey: The State of Waiting in Line (2022), 26.06.2022г.
* A Study of the Queueing Systems M/G/1 and GI/M/1: Uggappakodi Narayan Bhat (2013)
* The Use of Dupont Analysis by Market Participants: Mark T. Soliman · 2008
* Нов прочит на финансовия анализ от позициите на стойността: доц.д-р Виолета Касърова: (научен ръкопис депозиран в ЦНТБ № 169/2008 г.)
* Методология и методика за анализ на рентабилността на собствения капитал (интегриране на модел „Дюпон“ с модела за анализ чрез ефекта на финансовия лост): Проф. д-р Кръстьо Чуков, УНСС, катедра „Счетоводство и анализ“(03.2021 г.)
* Доходност на бизнеса – методологични и приложни аспекти на анализа контрола: доц. д-р Любомир Тодоров, Преподавател в УНСС – София (02/2017 г.)
* UX design processes and documentation: Dominic Pacholszyl
* Software
* Title: UX Design Process Best Practices: Jerry Cao, Ben Gremillion, Matt Ellis (2015)
* Development, Design and Coding: John F. Dooley ( 2017)
* Design Patterns: Elements of Reusable Object-Oriented Software: Erich Gamma, ‎Richard Helm, ‎Ralph Johnson ( 1994)
* Spring Data: Mark Pollack, ‎Oliver Gierke, ‎Thomas Risberg ( 2012)
* Spring Security in Action: Laurentiu Spilca ( 2020)
* Security with Spring: Baeldung ,Last updated ( 29.12 2022)
* Testing Computer Software: Cem Kaner, ‎Jack Falk, ‎Hung Q. Nguyen ( 1999)
* Testing in Spring Boot: Baeldung, ( Last updated: June 15, 2023)

# Приложения

Употреба за всички контролери към системата

**Автентикация.**

Служи за всички елементи в системата свързани с административна работа.

***POST /api/vi/auth/authenticate***

Заявката се нуждае от тяло от вида:

email - String

password - String

Примерно тяло:

*{*

*"email":"ivoAdmin@mail.com",*

*"password":"12345678"*

*}*

Отговора от тази заявка се е съдържанието на JWT токена. Времетраенето на токена е 12 часа.

**Артикул**

***GET /api/v2/article***

Връща лист от артиколи, като отговор от заявката. Нуждае се от автентикация.

***POST /api/v2/article***

Позволява ти да създадеш нов артикул. Нуждае се от Json тяло с характеристиките:

name- Long - Requerd

description- String - Requerd

workStart - SQLTime HH:mm:ss format -Requerd

wrokEnd - SQlTime HH:mm:ss format -Requerd

type - String -Requerd

Примерно тяло:

*{*

*"name":"name",*

*"description":"some description",*

*"workStart":"08:30:00",*

*"workEnd":"15:30:00",*

*"type":"type"*

*}*

Връща съобщение съдържащо идентификационен номер на ново създадения артикул.

***PUT /api/v2/article/:articleId***

Позволява да се променя артикул, който вече е бил създаден. Нужно е JSON тяло.

name- Long - Requerd

description- String - Requerd

workStart - SQLTime HH:mm:ss format -Requerd

wrokEnd - SQlTime HH:mm:ss format -Requerd

type - String -Requerd

Примерно тяло:

*{*

*"name":"name",*

*"description":"some description",*

*"workStart":"08:30:00",*

*"workEnd":"15:30:00",*

*"type":"type"*

*}*

Връща съобщение съдържащо идентификационен номер на артикула.

***PUT /addFavor/:articleId/:favorId***

Връща идентификационен номер на довявания артикул към услугата.

***DELETE /api/v2/article/:articleId***

**Гишета ( Counter)**

***GET /api/v2/counter***

Връща лист от гишета, като отговор от заявката. Нуждае се от автентикация.

***POST api/v2/counter***

Позволява да се създаде ново гише. Нуждае се от Json тяло с характеристиките:

name - String - Required

discription -String - Required

number - Integer – Required

Примерно тяло:

***{***

***"name":"counte1",***

***"description":"Some description",***

***"number":1***

***}***

Връща идентификационен номер на създаденото гише.

***PATCH api/v2/counter/:counterId***

Позволява да се променя гише, което вече е било създадено. Нужно е JSON тяло.

name - String - Required

discription -String - Required

number - Integer – Required

Примерно тяло:

*{*

*"name":"counte1",*

*"description":"Some description",*

*"number":1*

*}*

Връща съобщение съдържащо идентификационен номер на ново създаденото гише.

***PUT api/v2/counter/add/ticket/:favorId/:counterId***

За да се добави нова услуга към гишето.

***DELETE /api/v2/counter/:counterId***

**Услуги ( Favor)**

***GET api/v2/favor***

Връща лист от услуги, като отговор от заявката. Нуждае се от автентикация.

***POST api/v2/favor***

Позволява да създадеш нова услуга. Нуждае се от Json тяло с характеристиките:

name - String - Required

discription -String - Required

workStart - SQLTime HH:mm:ss format - Requerd

wrokEnd - SQlTime HH:mm:ss format - Requerd

idsOfTypeOfFavors - Integer Array - Requerd

Примерно тяло:

*{*

*"name":"AdminWork",*

*"description":"some description",*

*"workStart":"07:00:00",*

*"workEnd":"19:00:00",*

*"idsOfTypeOfFavors":[1]*

*}*

Връща идентификационен номер на новата услуга като съобщение.

***PATCH api/v2/favor/:favorName***

Позволява да се променя услугата, който вече е бил създаден. Нужно е JSON тяло.

name - String - Required

discription -String - Required

workStart - SQLTime HH:mm:ss format - Requerd

wrokEnd - SQlTime HH:mm:ss format - Requerd

idsOfTypeOfFavors - Integer Array - Requerd

Примерно тяло:

*{*

*"name":"AdminWork",*

*"description":"some description",*

*"workStart":"07:00:00",*

*"workEnd":"19:00:00",*

*"idsOfTypeOfFavors":[1]*

*}*

Връща съобщение съдържащо идентификационен номер на услугата.

DELETE /api/v2/favor/:favorId

**Машини ( Machine)**

***GET api/v1/machine***

Връща лист от машини, като отговор от заявката. Нуждае се от автентикация.

***POST api/v1/machine***

Позволява да създадеш нова машина. Нуждае се от Json тяло с характеристиките:

name - String - Required

type -String - Required

favorId -Long - Required

Примерно тяло:

*{*

*"name":"Roller",*

*"type":"Gate",*

*"favorId":1*

*}*

Връща идентификационен номер като съобщение на новосъздадената машина.

***PATCH api/v1/machine/:machineName***

Позволява да се промени машина, който вече е била създадена. Нужно е JSON тяло.

name - String - Required

type -String - Required

favorId -Long - Required

Примерно тяло:

*{*

*"name":"Roller",*

*"type":"Gate",*

*"favorId":1*

*}*

Връща съобщение съдържащо идентификационен номер на машината.

***DELETE /api/v2/machine/:machineId***

**Персонален билет ( Personal Ticket)**

***GET api/v2/personalticket***

Връща лист от персонални билети, като отговор от заявката. Нуждае се от автентикация.

***GET api/v2/personalticket/:ticketId***

Връща точно определен персонален билет според зададен идентификационен номер, като отговор от заявката. Нуждае се от автентикация.

***POST api/v2/personalticket***

Позволява да създадеш нов персонален билет. Нуждае се от Json тяло с характеристиките:

issueTime - SQLTime HH:mm:ss format - Requerd

Примерно тяло:

*{*

*"issueTime":"12:00:00"*

*}*

Връща съобщение съдържащо идентификационен номер на ново създадения персонален билет.

***PATCH api/v2/personalticket/:personalTicketName***

Позволява да се променя персонален билет, който вече е бил създаден. Нужно е JSON тяло.

name - String - Required

type -String - Required

favorId -Long - Required

Примерно тяло:

*{*

*"name":"Roller",*

*"type":"Gate",*

*"favorId":1*

*}*

Връща съобщение съдържащо идентификационен номер на персоналния билет.

***PUT api/v2/finish/:ticketNumber***

Приключва персонален билет спрямо работещия потребител.

***DELETE /api/v2/personalticket/:personalTicketId***

**Билет ( Ticket)**

***GET api/v2/ticket***

Връща лист от билети, като отговор от заявката. Нуждае се от автентикация.

***GET api/v2/ticket/:favorId***

Връща списък съдържащ всички билети от зададения идентификационен номер на услугата, който са настроени с открития по идентификационен номер на билети.

***POST api/v2/ticket***

Позволява да създадеш нова билети. Нуждае се от Json тяло с характеристиките:

name - String - Requerd

workStart- SQLTime HH:mm:ss format - Requerd

workEnd - SQLTime HH:mm:ss format - Requerd

favorId - Long - Requerd

typeId - Long -Requerd

Примерно тяло:

*{*

*"name":"ticketExample",*

*"workStart":"07:00:01",*

*"workEnd":"19:00:02",*

*"favorId":1,*

*"typeId":1*

*}*

Връща съобщение съдържащо идентификационен номер на билета.

***PATCH api/v2/ticket/:ticketName***

Позволява да се променя на билет, който вече е бил създаден. Нужно е JSON тяло.

name - String - Requerd

workStart- SQLTime HH:mm:ss format - Requerd

workEnd - SQLTime HH:mm:ss format - Requerd

favorId - Long - Requerd

typeId - Long -Requerd

Примерно тяло:

*{*

*"name":"ticketExample",*

*"workStart":"07:00:01",*

*"workEnd":"19:00:02",*

*"favorId":1,*

*"typeId":1*

*}*

Връща съобщение съдържащо идентификационен номер на билета.

***PUT addFavor/:idTicket/:idFavor***

Adds a favor to the ticket. Requirs authentication.

PUT addFavor/:idTicket/:idPersonalTicket

Adds a personal ticket to the ticket. Requirs authentication.

DELETE /api/v2/ticket/:ticketId

**Тип на билета( Tciket Type)**

***GET api/v2/tickettype***

Връща лист от типове билети, като отговор от заявката. Нуждае се от автентикация.

POST api/v2/tickettype

Позволява да създадеш нов тип билет. Нуждае се от Json тяло с характеристиките:

name - String - Requerd

description - String - Requerd

Примерно тяло:

{

"name":"Adminstation",

"description":"some description"

}

Връща съобщение съдържащо идентификационен номер на тип на билет.

PATCH api/v2/tickettype/:ticketTypeName

Позволява да се променя тип на билета, който вече е бил създаден. Нужно е JSON тяло.

name - String - Requerd

description - String - Requerd

Примерно тяло:

{

"name":"Adminstation",

"description":"some description"

}

Връща съобщение съдържащо идентификационен номер на тип на билет.

DELETE /api/v2/tickettype/:ticketTypeId

**Потребител ( User)**

GET /api/v1/user

Връща лист от потребители, като отговор от заявката. Нуждае се от автентикация.

POST /api/v1/user/register

Позволява да създадеш нов потребител. Нуждае се от Json тяло с характеристиките:

firstname - String - Requerd

lastname - String -Requerd

email - String - Requerd

role - String - Requerd

password- String -Requerd

Примерно тяло:

{

"firstname":"Ivo",

"lastname":"Rumenov",

"email":"ivoUser@mail.com",

"role": "USER",

"password":"12345678"

}

Връща съобщение съдържащо идентификационен номер на тип на билет.

PUT /api/v1/user/register/addCounter/:counterId

Позволява ти да добавиш ново гише за работа на потребител.

DELETE /api/v1/user/:userId

**Опашка (Queue)**

GET api/v1/queue/nextInLine/:counterId

Връща лист от билетите чакащи на опашка, като отговор от заявката. Нуждае се от автентикация.

PUT api/v1/queue/open/counter/:counterId

Отваря гише.

DELETE api/v1/queue/close/counter/:counterId

Затваря гише.

GET api/v1/queue/waiting ForCounter/:counterId

Връща всички персонални билети за това гише.

**Теглене ( Draft)**

Автентикация за този контролер не е нужен.

GET api/v1/draft/favor

Връща списък със всички услуги.

Get api/v1/draft/waitingForCounter/:counterId

Връща всички активни персонални билети чакащи за обозначеното гише.

GET api/v1/draft/lastPersonalTicket

Връща последния изтеглен персонален билет.

POST api/v1/draft/:ticketId

Създава но персонален билет спрямо избора на клиент от потребителския интерфейс.

id - Long - Not Requerd

issueTime - SQLTime HH:mm:ss format - Requerd

Примерно тяло:

{

"id":123123,

"issueTime":"12:00:00"

}

1. Метод на Ансов- е икономически метод за анализ на конкуренцията [↑](#footnote-ref-1)
2. Postgresql- известен също като Postgres, е безплатна система за управление на релационни бази данни с отворен код. [↑](#footnote-ref-2)
3. ACID- Е широка използвана методология за работа с бази от данни. [↑](#footnote-ref-3)
4. Java- Широко разпространен обектно ориентиран програмен език. [↑](#footnote-ref-4)
5. Spring- Твърдо свързани библиотеки за усилване работата на програмния език java. [↑](#footnote-ref-5)
6. Apache- Спомагателен инструмент за компилиране и пакетиране на java код. [↑](#footnote-ref-6)
7. Tomcat- Сървър за разгъване на java базиран уеб сървър. [↑](#footnote-ref-7)
8. Maven- Инструмент за обединяване и пакетиране на библиотеки използвани в проекта. [↑](#footnote-ref-8)
9. Бъгове- Вид кредитни грешки в програмния код. [↑](#footnote-ref-9)
10. Vanila JavaScript- Е функционален програмен език главно използван за уеб интерфейси. [↑](#footnote-ref-10)
11. Html- Файлов формат за визуализация на уеб страници. [↑](#footnote-ref-11)
12. Wait While- Институт за измерване на количеството от часове, който хората отделят за чакане. [↑](#footnote-ref-12)
13. NFC- Технология базирана на ниско честотни радио вълни. Главно използвани в картите за сигурност. [↑](#footnote-ref-13)
14. Контролер- Част от логическия слой на програмния продукт. [↑](#footnote-ref-14)
15. Енум- Вид променлива, която се използва в програмирането за да съхранява набор от константи. [↑](#footnote-ref-15)
16. Теория на опашките- Е математически алгоритъм за пресмятане на входен и изходен поток от данни. [↑](#footnote-ref-16)
17. Турникет- Вид пропускателен пункт, който пропуска един след друг индивидуално. [↑](#footnote-ref-17)
18. Gini-е мярка за разделението или разнообразието на класовете във възлите на дървото. [↑](#footnote-ref-18)
19. Строител ( Builder)- Вид логически шаблон за работа с обекти. [↑](#footnote-ref-19)
20. Payload- Клас, който служи за запис на обекти и техните данни. Те не съдържат никаква логика вътре в себе си. [↑](#footnote-ref-20)
21. Body- Така нареченото тяло е файл, който се изпраща заедно с заявката. Най-често се използва за да се прехвърлят обекти помежду потребителския интерфейс и сървъра. [↑](#footnote-ref-21)
22. Персона- Представяне на сходна личност до реална в пълен житейски характер. [↑](#footnote-ref-22)
23. Брайлова азбука-Брайлова азбука се нарича тактилна система за писане и четене, използвана от слепите и слабовиждащите хора по света. [↑](#footnote-ref-23)
24. Брутализъм- Вид дизайн. Главно използван в архитектурата. [↑](#footnote-ref-24)
25. React- Екосистема за програмиране на потребителски уеб интерфейси. [↑](#footnote-ref-25)
26. Angular- Екосистема за програмиране на потребителски уеб интерфейси. [↑](#footnote-ref-26)