**ТЕХНОЛОГИЧНО УЧИЛИЩЕ “ЕЛЕКТРОННИ СИСТЕМИ”**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: ERP за сервизна дейност

Дипломант: Научен ръководител:

*Александър Маринов Мартин Тасков*

СОФИЯ

2020

Дата на заданието: 15.11.2019 г. Утвърждавам:..............................

Дата на предаване: 15.02.2020 г. /проф. д-р инж. Т. Василева/

**ЗАДАНИЕ**

**за дипломна работа**

на ученика Александър Иванов Маринов 12A клас

1.Тема: ERP за сервизна дейност

2.Изисквания:

2.1. Responsive уеб сайт за управление на сервиз

2.2. История на ремонтите, водене на worklog (с текст, снимки, поръчани части)

2.3. Клиентска част за следене на ремонта и история на ремонти

2.4. Онлайн плащане на ремонта

2.5. Доставка с търсене на адрес и визуализация с google maps

2.6. Генериране на pdf фактура или printer friendly страница и изпращане на email

2.7. Интеграция със спедиторско API

3.Съдържание 3.1 Обзор

3.2 Същинска част

3.3 Приложение

Дипломант :...........................................

Ръководител:..........................................

/ Мартин Тасков /

Директор:................................................

/ доц. д-р инж. Ст. Стефанова /

УВОД

ERP представлява интегрирана софтуерна информационна система, която се грижи за планирането и управлението на всички ресурси в едно. Системата има за цел да обхване всички звена и бизнес процеси от живота на една компания.

Интегрирана система е тази, в която информацията се въвежда веднъж и без допълнителна обработка (в реално време) се отразява във всички модули и подсистеми, като става достъпна за всички потребители.

ERP замества традиционните информационни системи, като ги обединява в единна, унифицирана програма, разделена на взаимно свързани модули според отделните сектори в едно предприятие. Предназначена е за автоматизиране на управлението и отчетността.

Повечето системи от този тип разрешават имплементацията на отделни модули, покриващи един или повече бизнес процеси. В дипломната си работа съм създал само модулите свързани със сервизната дейност по приемането, ремонта и издаването на дефектирал уред, както и управлението на складовите наличностти.

Реших да създам този сервизен модул за гаранционно и извън гаранционно поддържане и ремонт на бяла и черна техника, защото след мое проучване на пазара на подобни продукти и разговори с хора от бранша се оказа, че се предлагат основно два вида софтуер – с много добри параметри и функционалност, но с висока цена или с по-ниска цена, но не достатъчно функционални или пък много тромави. Останалите модули от ERP системата не са предмет на разработката ми, поради следните причини: нямам достатъчно знания в съответните области; в основата си тези модули са стандартни/ напр. счетоводем модул/ или типа дейност, която покрива модула, е по-изгодно да бъде аутсорсната към външна фирма / напр. куриерски фирма/.

Създаденият от мен продукт е насочен основно към малкия и среден бизнес, тъй като считам, че той ще се търпи развитие в следващите няколко години/ особено в районите с по-малобройно население/. Модулите са изградени на основата на едни от най-новите софтуерни продукт.

Целта им е да се автоматизира създаването на ремонтна карта, като се избегнат грешките при попълване на модел, сериен номер и продуктов код/ IMEI за дефектиралите уреди. Също така максимално опростения му, но с достатъчна функционалност за този вид дейност, дизайн ще доведе до повишаване ефективността на персонала, опростяване на административните дейности, извършвани от техниците, което от своя страна ще доведе до повишаване на ефективността им при ремонта на уредите.

Системата позволява към нея да бъдат прикачени и допълнителни модули, в зависимост от изискванията на съответния клиент, но това ще доведе до оскъпяване на продукта като цяло.

Информацията за клиентите са защитени от секюрити модул, разработен от мен, в съответствие с европейските принципи за защита на личните данни.

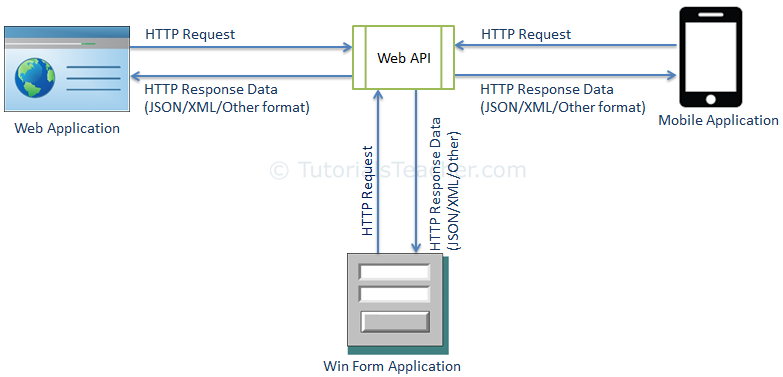
**ПЪРВА ГЛАВА**

Преглед на използваните технологии и съществуващи решения

* 1. Обзор на използваните технологии
     1. ASP.NET Web API

API (Application Programming Interface) е тип интерфейс, който има набор от функции, които позволяват достъп до специфични свойства или данни на приложение, операционна система или други сървиси. Web API както името показва, е API, което може да бъде достъпено чрез HTTP протокола.

ASP.NET Web API е extensible framework за изграждане на HTTP базирани сървиси, които могат да бъдат достъпени от различни приложения на различни платформи като уеб, уиндоус, мобилни и други. Принципа на действие е показан на фиг. 1.1.



фиг. 1.1

* + 1. SignalR

SignalR е библиотека с отворен код, която опростява процеса по добавяне на финкционалности към дадено приложение в реално време (real – time web functionality (RTWF)) и е направена, за да бъде използвана от разработчици на ASP.NET. RTWF е способността на даден сървър да качва ново съдържание в приложението и то да става достъпно веднага за всички свързани клиенти вместо клиента да прави нов request за данни.

SignalR може да бъде използвана за добавяне на всякакъв тип RTWF към вече съществуващо ASP.NET приложение. Най – често давания пример за това е чат приложение, но може да бъде направено много повече. Вески път, когато потребител обнови уеб страница, за да види нови данни, или страницата имплементира long polling, за да вземе нови данни, това е кандидат за използването на SignalR.

SignalR също така предоставя възможността за напълно нов тип уеб приложения, при които се налага обновявания много често от сървъра като например real-time gaming.

SignalR предоставя просто API, което създава server-to-client remote procedure calls (RPC), което извиква JavaScript функции в клиентските браузъри от server-side .NET code. SignalR също така съдържа API за управление на връзката и връзка между групи. Принципът на действие е показан на фиг. 1.2.



фиг. 1.2

SignalR handles connection management automatically, а това позволява на данните да бъдат разпрострени до всички клиенти едновременно, както и до специфичен клиент. Връзката между сървъра и клиента е постоянна, за разлика от класическата HTTP връзка, която трябва да се подновява за всяка комуникация.

SignalR поддържа “server push“ functionality, при която кодът в сървъра може да бъде извикан от кода при клиента в браузъра, използвайки RPC вместо request-response модела, който може да се види във всеки сайт днес. Тази бибиотека е вградена по default в по – долу описаната технология.

* + 1. Blazor

Blazor е фреймуърк за създаване на интерактивен client-side web UI, използвайки .NET: създаване на интерактивни UIs, използвайки C# вместо JavaScript (JS); server-side and client-side логика на приложението, написана на .NET; render UI като HTML и CSS и поддрежка за много браузъри, включително мобилни.

Използването на .NET за client-side web development предоставя следните предимства:

* Писане на C# вместо на JavaScript
* Използва съществуващата .NET екосистема от .NET библиотеки
* Share app logic across server and client
* Предимствата на производителността, надежността и сигурността на .NET.

Типовете проекти, който могат да се изграждат с този фреймуърк, са два: Blazor Server App и Blazor WebAssembly, но първо ще бъде обяснено какво е Razor Component.

* + - 1. Razor Components (RC)

Blazor приложенията са баизрани на компоненти. Компонент в Blazor елемент от UI като страница, изскачащо прозорче или форма за попълване на данни.

Компонент са .NET класове, компилирани в .NET assemblies, които дефинират flexible UI rendering logic, обработват user events, могат да бъдат вложени и преизползвани.

Класът на компонентите обикновено се записва под формата на страница за маркиране на Razor с разширение на файла .razor. Компонентите в Blazor официално се наричат Razor Components. Razor е синтаксис за комбиниране на HTML маркиране с C # код, предназначен за производителност на разработчиците. Razor Pages и MVC също използват Razor. За разлика от Razor Pages и MVC, които са изградени около модела на заявка – отговор, компонентите се използват специално за логиката и състава от страна на клиента от потребителския интерфейс.

Фиг. 1.3 демонстрира компонент (Dialog.razor), който може да бъде вложен в друг компонент:

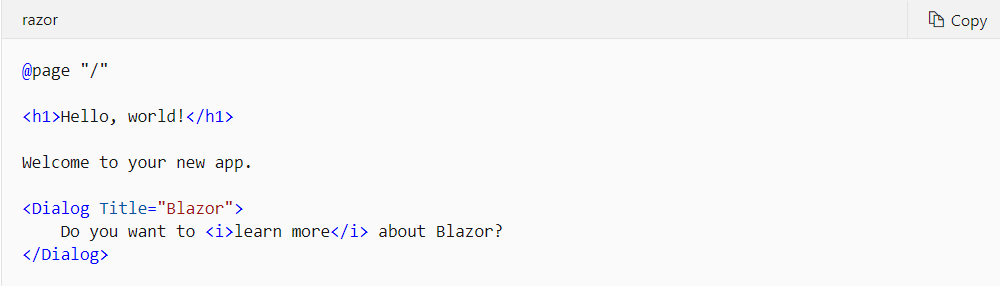


фиг. 1.3

Съдържанието на тялото на диалога (ChildContent) и заглавието (Title) се предоставят от компонента, който използва този компонент в своя потребителски интерфейс. OnYes е метод на C #, задействан от събитието onclick на бутона.

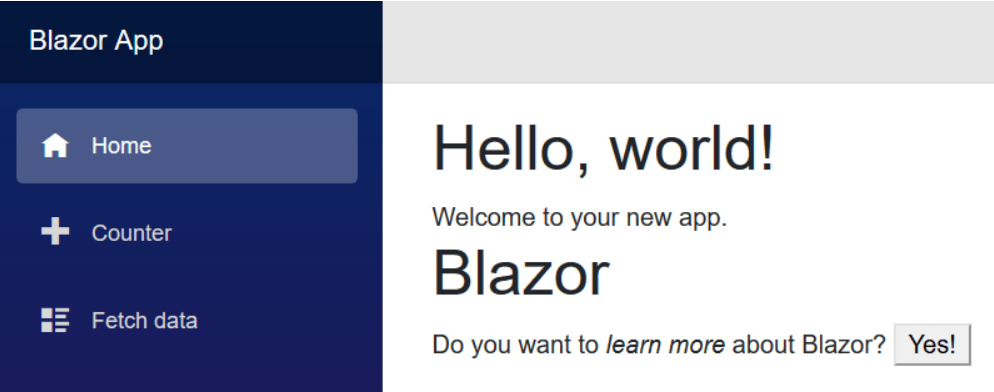
Blazor използва естествени HTML тагове за UI композиция. HTML елементите определят компоненти, а атрибутите предават стойности на свойствата на компонента.

В следващия пример компонентът Index използва компонента Dialog. ChildContent и Title се задават от атрибутите и съдържанието на елемента <Dialog> (фиг 1.4).



фиг. 1.4

Диалогът се визуализира, когато родителят (Index.razor) бъде достъпен в браузъра (фиг. 1.5)



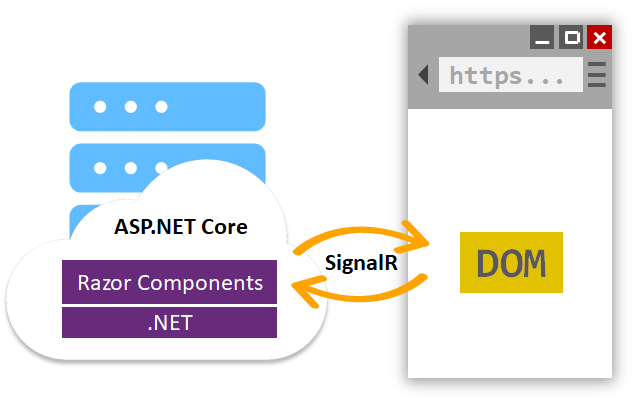
фиг. 1.5

* + - 1. JavaScript interop

Приложения, при които се налага използването на third-party JavaScript libraries и достъп до APIs в браузъра, компонентите си взаимодействат с JS. Компонентите имат способността да използват всяка библиотека или API, което JS е способен да използва. C# код може да извиква JS код, както и JS код да извиква C# код.

Типовете проекти, който могат да се изграждат с този фреймуърк, са два: Blazor Server App и Blazor WebAssembly.

* + - 1. Blazor Server App (BSA)

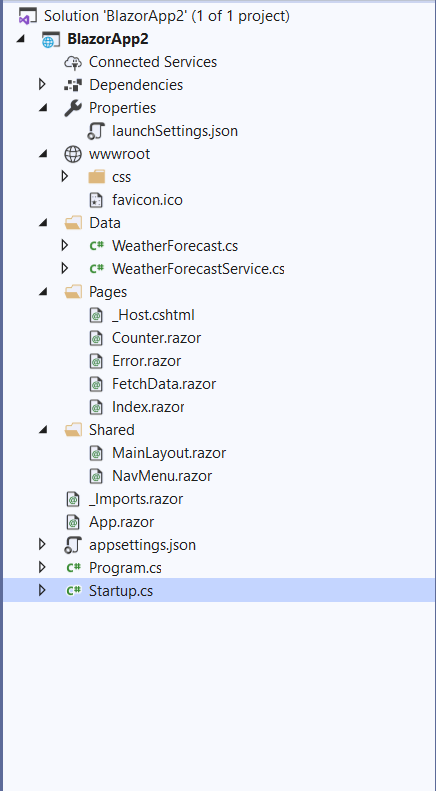


фиг. 1.6

На фиг. 1.6 е показано на как работи едно BSA приложение. Blazor отделя component rendering logic от начина на прилагане на актуализациите на UI. Blazor Server предоставя поддръжка за хостинг на Razor компоненти на сървъра в ASP.NET Core приложението. Актуализациите на UI се обработват чрез SignalR connection.

The runtime обработва изпращането на UI събития от браузъра до сървъра и прилага актуализации на UI, изпратени от сървъра обратно към браузъра след стартиране на компонентите.

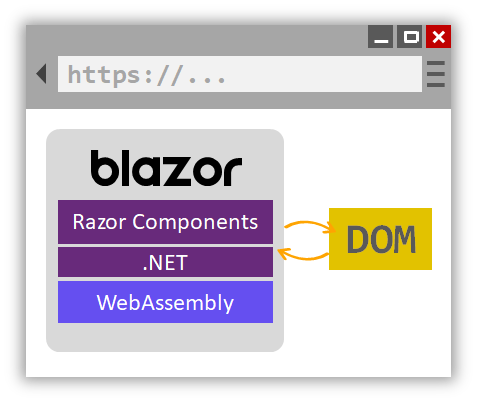
Връзката, използвана от BSA за комуникация с браузъра, се използва и за обработка на JS interop calls.



фиг. 1.7

На фиг. 1.7 е показано каква е структурата на едно BSA приложение. То е разделено на следните части:

* wwwroot – в нея се съдържа всичкия css, js, icons и т.н., които са използвани в приложението
* Data – там се съдържат всички модели и сървиси, а сървисите трябва да се добавят в Startup.cs.
* Pages – в нея се съдържат всички страници, което е BSA приложение е възможно да има и те са .razor файлове, които са обяснени как работят по – нагоре. \_Host.cshtml е Razor Page, в която се добавят всички връзки към css и js файллове.
* Shared – в нея се съдържат файловете NavMenu.razor и MainLayout.razor. В NavMenu.razor се пишат всички елементи на менюто. В MainLayout.razor се пише как ще изглежда основната страница на приложението.
* Startup.cs и Program.cs – съответно съдържа конфигурацията на цялото приложение и компилира цялото приложение.
  + - 1. Blazor WebAssembly (Wasm)



фиг. 1.8

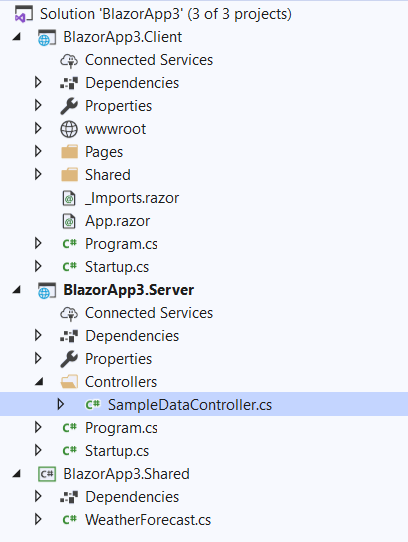
На фиг. 1.8 е показано как работи едно Wasm приложение. Blazor WebAssembly е SPA framework за изграждане на интерактивни client-side уеб приложения с .NET. Blazor WebAssembly използва отворени уеб стандарти без плъгини или преобразуване на код и работи във всички съвременни уеб браузъри, включително мобилни браузъри.

Изпълнението на .NET код в уеб браузърите става възможно от WebAssembly (съкратено Wasm). WebAssembly е компактен bytecode формат, оптимизиран за бързо изтегляне и максимална скорост на изпълнение. WebAssembly е отворен уеб стандарт и се поддържа в уеб браузъри без plugins.

Кодът на WebAssembly може да има достъп до пълната функционалност на браузъра чрез JS, наречен оперативна съвместимост на JavaScript (или JS interop). .NET код, изпълнен чрез WebAssembly в браузъра, работи в JavaScript sandbox на браузъра със защитите, които sandbox осигурява срещу злонамерени действия на клиентската машина.

Когато Wasm приложението е компилирано и работещо в браузъра:

* C# и Razor файловете се компилират в .NET assemblies
* The assemblies and the .NET runtime се свалят в браузъра
* Wasm стартира .NET runtime и конфигурира runtime, за да зареди монтажите за приложението. Wasm runtime използва JS interop за манипулация DOM дървото и извиквания на API в браузъра



фиг. 1.9

На 1.9 e показано каква е структурата на едно Wasm приложение. То се състои от три модула:

* Client – в него се съдържат целия front-end на приложението.
* Server – в него се съдържа цялата back-end логика
* Shared – в него се съдържа всички модели на приложението

В .NET Core 3.1 Wasm не е налично като тип проект, защото дава грешки, които са от .NET Core 3.

* + 1. Google Firebase

Google Firebase е Google – backed софтуер за разработка на приложения, който дава възможност на разработчиците да разработват приложения за iOS, Android и Web. Firebase предоставя инструменти за проследяване на аналитиката, отчитане и коригиране на сривове на приложения, създаване на маркетинг и експеримент с продукти.

Firebase предлага редица услуги, включително:

* Analytics – Google Analytics за Firebase предлага безплатно, неограничено отчитане за повече от 500 отделни събития. Analytics представя данни за поведението на потребителите в приложения за iOS и Android, което позволява по-добро вземане на решения за подобряване на производителността и маркетинга на приложенията.
* Authentication – Firebase Authentication улеснява разработчиците при изграждането на защитени системи за authentication и подобрява практиката на влизане и вграждане за потребителите. Тази функция предлага цялостно решение за идентичност, поддържащо имейл акаунти и акаунти за парола, оторизиране на телефона, както и Google, Facebook, GitHub, вход в Twitter и други. Този продукт на Firebase е използван в настоящата дипломна работа.
* Crashlytics – Firebase Crashlytics е real-time crash reporter, който помага на разработчиците да проследяват, дават приоритет и да коригират проблеми със стабилността, които намаляват качеството на техните приложения. С Crashlytics разработчиците отделят по-малко време за организиране и отстраняване на неизправности и повече функции за изграждане на време за своите приложения.
* Realtime database – Firebase Realtime database e cloud-hosted NoSQL база данни, която позволява съхраняването и синхронизирането на данни между потребителите в реално време. Данните се синхронизират за всички клиенти в реално време и все още са достъпни, когато приложението е офлайн.
* Cloud Firestore – Cloud Firestore е flexible, scalable база данни за разработка на мобилни устройства, уеб и сървър от Firebase и Google Cloud Platform. Подобно на Firebase Realtime Database, тя поддържа данните синхронизарани между клиентските приложения чрез realtime listeners и предлага офлайн поддръжка за мобилни и уеб, за да може да се изграждат отзивчиви приложения, които работят независимо от латентността на мрежата или интернет връзката. Cloud Firestore също предлага безпроблемна интеграция с други продукти на Firebase и Google Cloud Platform, включително Cloud Functions. Този продукт на Firebase е използван в настоящата дипломна работа.
  + 1. Google Cloud Platform (GCP)
       1. Cloud Computing (CC)

CC, казано накратко, е достъп до и съхраняване на данни през интернет, вместо да бъдат съхранявани на твърд диск.

Предлага услуги като съхранение, база данни, работа в мрежа и повече през Интернет, за да предостави по – бързи, иновативни и flexible resources на своите клиенти. Клиентите плащат само за ресурсите, които използват, като по този начин им помага да намалят оперативните си разходи и да управляват бизнес инфраструктурата си по – ефективно.

* + - 1. GCP

GCP е набор от Cloud Computing сървиси, предлагани от Google. Платформата предоставя различни услуги като изчисляване, съхранение, работа в мрежа, големи данни и много други, които работят на същата инфраструктура, която Google използва вътрешно за своите крайни потребители като Google Търсене и YouTube.

GCP предлага голям набор от функции, включително:

* Compute Services
  + Google App Engine – платформа като услуга за deploy на Java, PHP и други приложения. Това е Cloud Computing платформа за разработване и хостинг на уеб приложения в Google-managed data centers. Предлага функция за автоматично мащабиране, т.е. с увеличаването на броя на заявките за приложение, App Engine автоматично разпределя повече ресурси за приложението, за да се справи с допълнителното търсене.
  + Compute Engine – инфраструктура като услуга за стартиране на виртуални машини на Microsoft Windows и Linux. Това е компонент на платформата Google Cloud, която е изградена на същата инфраструктура, която управлява Google търсачката, YouTube и други услуги. Compute Engine е използван в настоящата дипломна работа.
  + Kubernetes Engine: Той има за цел да предостави платформа за автоматизиране на deployment, scaling и операции с контейнери за приложения в клъстери от хостове. Работи с широк набор от инструменти за контейнери, включително Docker.
* Storage Services
  + Google Cloud Storage – интернет услуга за съхранение на файлове в Интернет за съхранение и достъп до данни в инфраструктура на GCP. Услугата съчетава ефективността и мащабируемостта на Google Cloud with advanced security and sharing capabilities.
  + Cloud SQL – уеб услуга, която позволява създаването, конфигурирате и използването на релационни бази данни, които живеят в Google Cloud. Поддържа, управлява и администрира базите данни.
  + Cloud Bigtable - бърза, напълно управлявана и високо мащабируема NoSQL database service. Той е предназначен за събиране и задържане на данни от 1 TB до стотици PB.

**ВТОРА ГЛАВА**

Изисквания към програмния продукт, аргументация за използваните технологии и структура на приложението

* 1. Аргументация за използваните технологии
     1. Blazor

Blazor е SPA фреймуърк за разработване на уеб приложения. В настоящата дипломна работа е избран поради следните причини:

* Напълно съвместим е с .NET libraries и .NET tooling. За разлика от client-side Blazor (Wasm), който се предлага с непълни възможности за отстраняване на грешки, грешките на server-side Blazor могат да бъдат отстранявани по абсолютно същия начин като всяко друго ASP.NET приложение. По същия начин, всички други стандартни инструменти, които могат да бъдат използвани с всякакви други видове .NET приложения, могат да се използват с server-side Blazor.
* Използва точно същия синтаксис като client-side Blazor(Wasm). Въпреки че Blazor от страна на сървъра може да не е най-добрият избор за определени сценарии, когато client-side Blazor е полезен, все още първоначално приложението да се напише на server-side Blazor, за да се преодолеят ограничените възможности за отстраняване на грешки на client-side Blazor. След като приложението е готово за release, кодът от приложението се копира в проекта, който е clients-side Blazor. Синтаксисът позволява да бъдат написани class libraries, които ще работят с server-side Blazor или client-side Blazor.
* Малък размер на клиентски компоненти. Когато става дума за изтегляне на клиентски компоненти на server-side Blazor, те ще включват само сравнително малко количество HTML и JavaScript, нито един от които не е задължително да бъде написа.
* Works with thin clients. Server-side Blazor не го интересува в кой браузър се показва. Клиентът получава само стандартни HTML и JavaScript, които работят практически навсякъде. Следователно server-side Blazor e съвместим с почти всеки браузър.
  + 1. Web API

В настоящата дипломна работа ASP.NET Web API е избрано, защото Blazor WebAssembly(Wasm) все още е в beta версия и не е напълно изправно. Затова чрез Blazor server app и Web API се симулира Wasm.

* + 1. Google Firebase

Firebase е добър избор, защото като продукт на Google често подлежи на обновяване и коригиране на проблеми, за някои услуги може да се ползва безплатно, като цяло работи доста прилично и просто, заради това много типове проекти са с Firebase. Firebase е избран, защото се намира в облачното пространство, а с навлизането на квантовите компютри няма да се налага миграция към облачните пространства.

* + - 1. Cloud Firestore

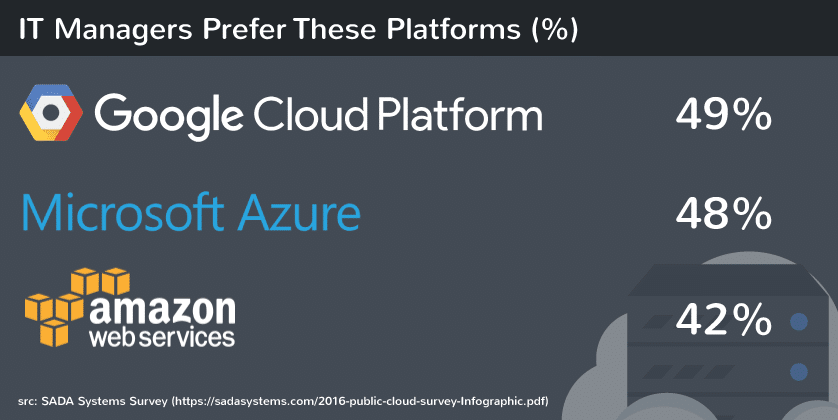
Google Firebase Cloud Firestore е пуснат официално през 2018 година като част от Google Firebase Platform. Firestore e schema less база данни, а това означава, че няма строги правила за дефиниране на съхраняваните данни. Всеки ред от базата данни може да има различни колони, а това е много голямо предимство за голяма част от разработчиците на приложения. Във Firestore данните се пазят в документи, които представляват двойки ключ-стойност, а от своя страна това предлага бързо търсене. Firestore е хоринзонзонтално нарастваща база данни, а това означава нарастване чрез добавяне на повече машини в ресурна на приложението и всеки node (машина) съдържа част от данните, а това не налага закупуването на повече сървъри. Разделянето на node-ове на базата данни става автоматично и е скрито от разработчиците.

* + - 1. **Authentication**

Firebase Authentication е избран, защото може много лесно да се направи логин система, в която да се включва логване с Google, Facebook, Twitter и други. Освен това, предоставя добра защита за всички на всеки user.

* 1. Google Cloud Platform(GCP)

GCP e избрана, защото според проучване на SADA Systems, 84% от IT мениджърите използват public cloud infrastructure (фиг. 2.1). Някои от най – големите компании като Cisco прогнозират, че до 2020 година 92% от трафика ще представлява трафик в облачното пространство. Освен това, някои бележити компании вече използват GCP – Spotify, HSBC, Home Depot, Snapchat, HTC, Best Buy, Philips, Coca Cola, Domino’s, Feedly, ShareThis, Sony Music, and Ubisoft.

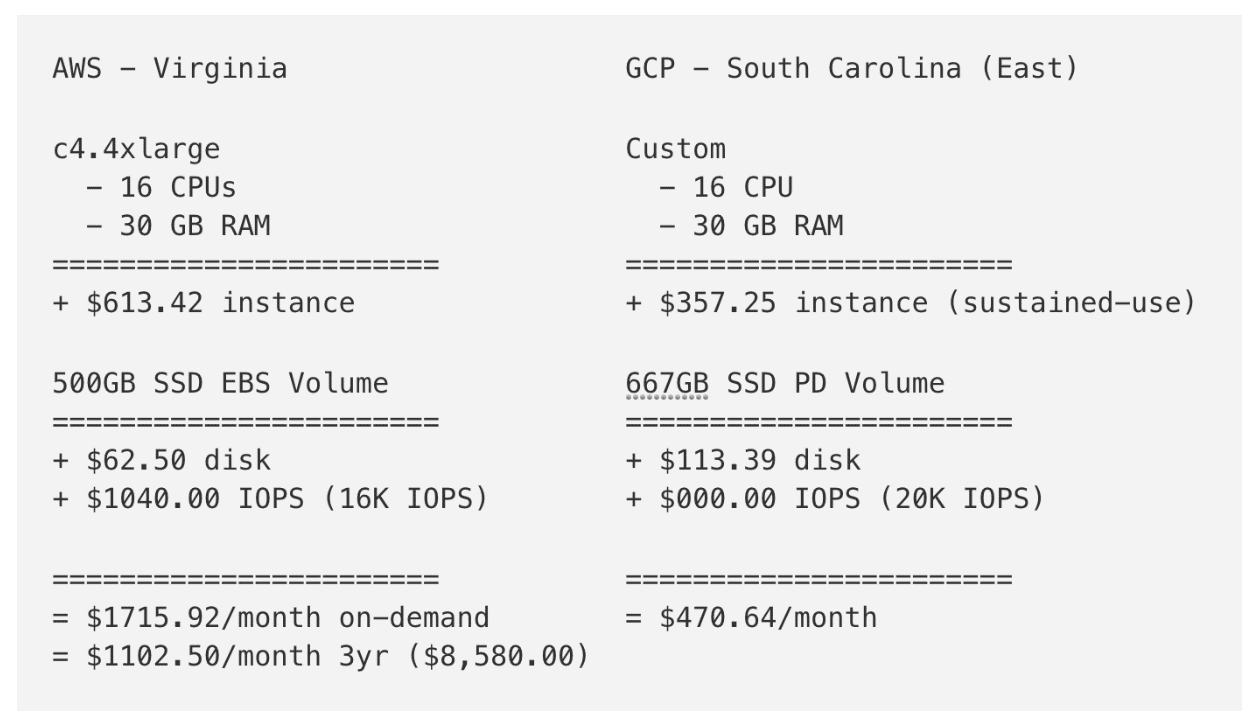


фиг. 2.1

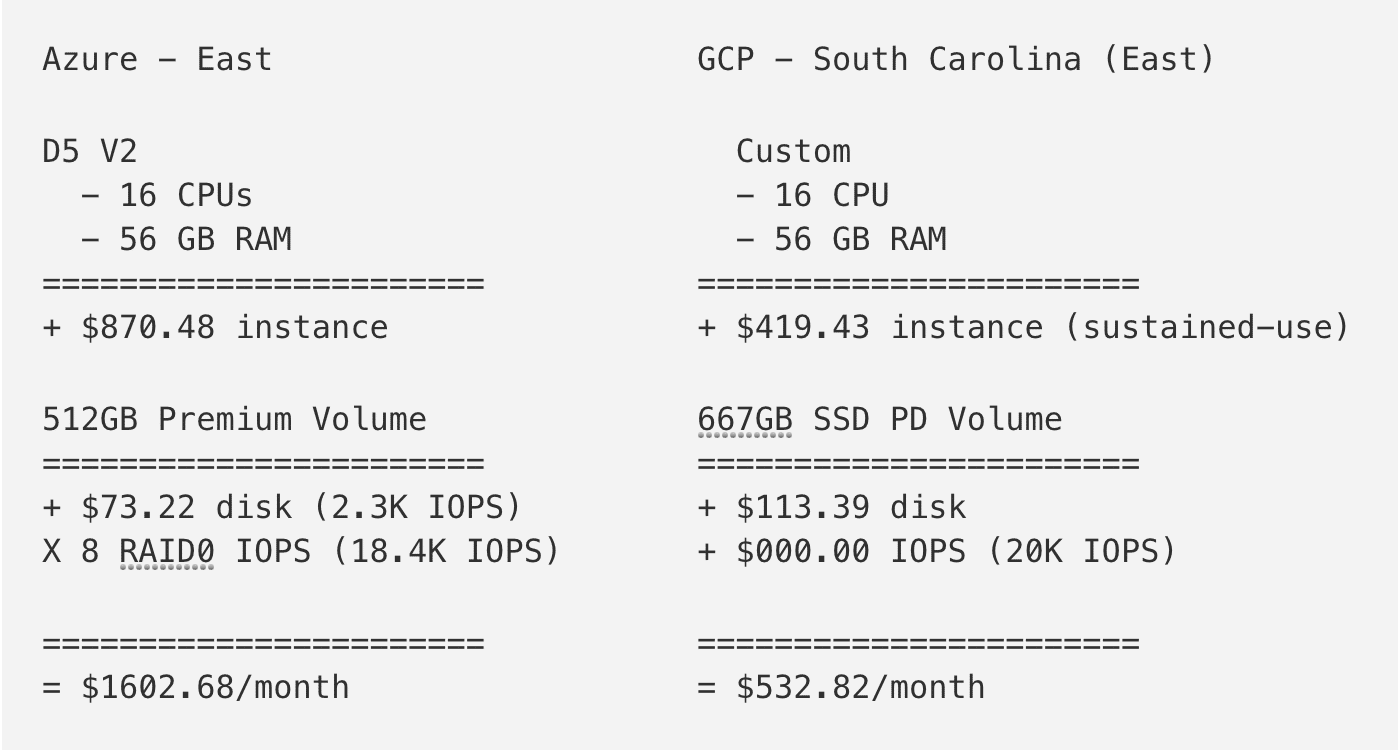
Част от предимствата на GCP пред конкурентите са:

* Better Pricing Than Competitors
* Live Migration of Virtual Machines
* State of the Art Security
* Redundant Backups
  + 1. Better Pricing Than Competitors

Google bills in minute-level increments (with a 10-minute minimum charge), а това означава, че се плаща само за използваното време и се предлагат намаления в цените ако приложението е изплозвано продължително и е натоварено без предварително ангажиране. След 1-месечно използване на VM, се получава отстъпка. Това го прави подходящ за startup компании и за IT enterprises, които искат да намалят разходите си. Сравнения между цените са показани на фиг. 2.2 и 2.3.

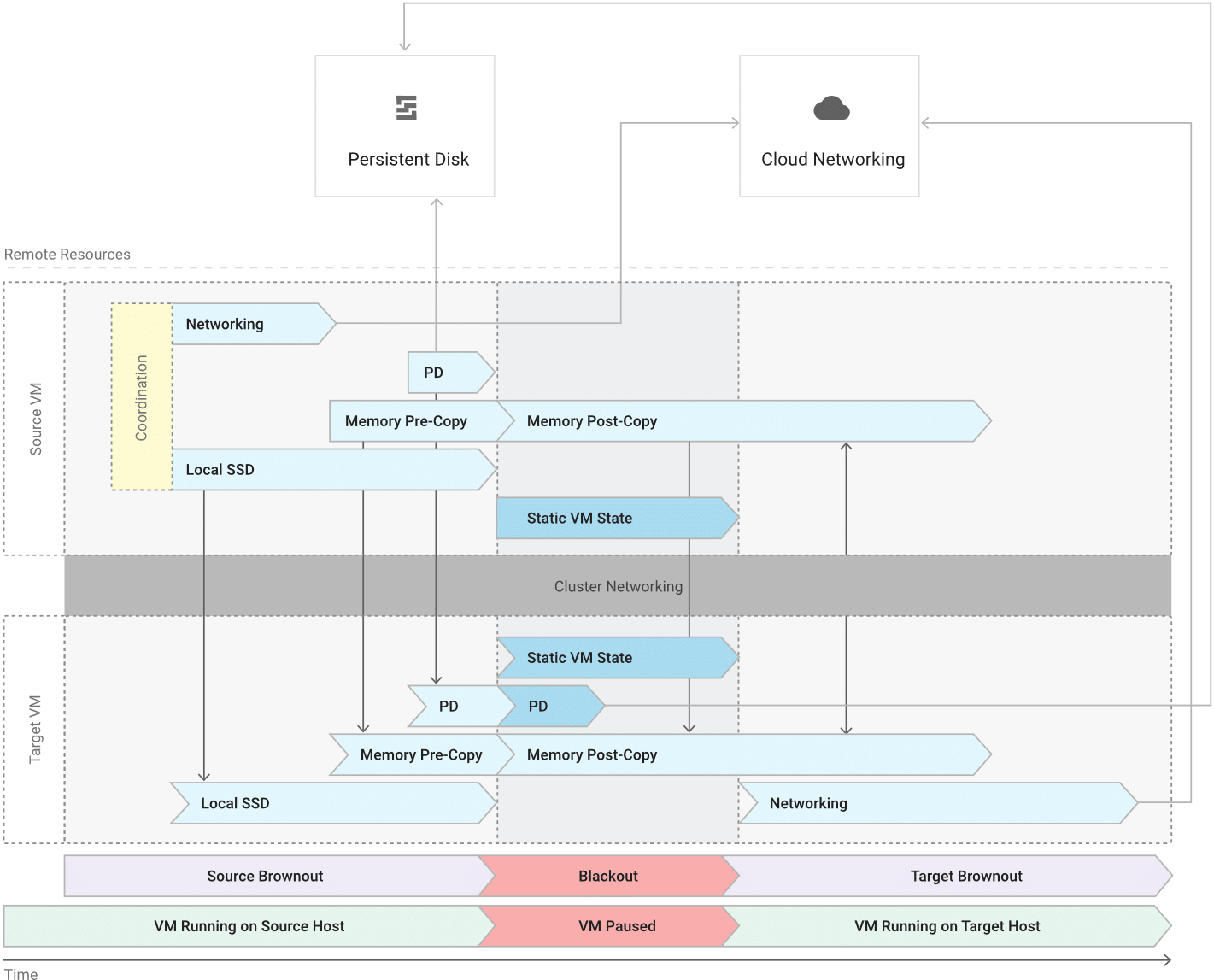


фиг. 2.2

 фиг. 2.3

* + 1. Live Migration of Virtual Machines

Друго огромно предимство за GCP e live migrations of Virtual Machines. Нито AWS, нито Azure, нито по-малки доставчици като Digital Ocean предлагат тази функционалност. Предимствата на live migrations позволяват на инженерите от Google да решават по-добре проблеми като patching, repairing и актуализиране на софтуера и хардуера, без да е необходимо да се притеснявате от рестартиране на машината. Принципът на действие е показан на фиг. 2.4.



фиг. 2.4

Освен live migrations of Virtual Machines, Google предлага и преоразмеряване на persistent disks без никакъв престой.

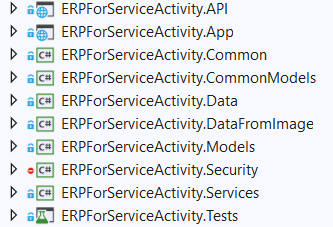
* + 1. State of the Art Security

Друго голямо предимство е сигурността. Изборът на Google Cloud Platform означава, че ще бъде използван модел за сигурност, който е изграден в продължение на 15 години и понастоящем осигурява продукти и услуги като Gmail, Search и др. Понастоящем Google има повече от 500 специалисти в областта на сигурността.

* + 1. Redundant Backups

Google Cloud Storage е проектиран за дълготрайност 99, 999999999% и има 4 различни типа съхранение: Coldline, nearline, регионално и многорегионално съхранение. Съхранява данните излишно, с автоматични checksums, за да се гарантира целостта на данните. Красотата на мултирегионалното съхранение е, че той е и излишен в географски условия, което означава облачно съхранение съхранява данните излишно в поне два региона в рамките на мултирегионалното местоположение на bucket-a. Тъй като се отнася до хостинг на Google Cloud, това гарантира, че никога няма да има загуба на данни, дори в случай на бедствие.

* 1. Структура на приложението

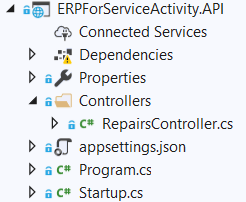


фиг. 2.5

На фиг. 2.5 е показано как изглежда структурата на приложението. Състои се от десет модула:

* ERPForServiceActivity.API – използван е за обработка на заявки
* ERPForServiceActivity.App – главния модул на приложението
* ERPForServiceActivity.Common – съдържа всички използвани константи
* ERPForServiceActivity.CommonModels – съдържа всички binding и view модели
* ERPForServiceActivity.Data – конфигурация на връзката с базата данни
* ERPForServiceActivity.DataFromImage – взимане на информация от снимка
* ERPForServiceActivity.Models – съдържа моделите, които си комуникират с базата данни
* ERPForServiceActivity.Security – съдържа цялата защита на приложението, която няма да бъде описана в настоящата дипломна работа
* ERPForServiceActivity.Services – съдържа логиката за действията в приложението
* ERPForServiceActivity.Tests – съдържа всички тестове, написани за приложението
  + 1. **ERPForServiceActivity.API**

Структурата е показана на фиг. 2.6.

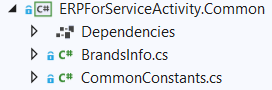
* + - * 1. 

фиг. 2.6

Съдържа следните папки и файлове:

* Controllers – съдържа всички контролери, които са написани и които се използват.
* appsettings.json – съдържа всички настройки, които изискват данни в този файл
* Program.cs и Startup.cs – файлове съответно, който пуска целия модул, и който съдържа настройки на приложението
  + 1. ERPForServiceActivity.App – структурата е като на Blazor Server App от първа глава
    2. ERPForServiceActivity.Common

Структурата е показана на фиг. 2.7.

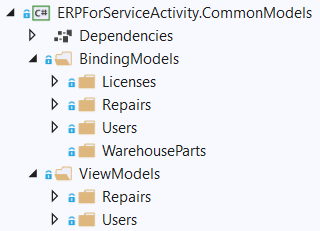


фиг. 2.7

В него се съдържат всички константи, които са използвани в приложението. BrandsInfo.cs съдържа поле за наличните марки като бива използван като лист от стрингове, който се визуализира като падащо меню при създаването на поръчка.

* + 1. ERPForServiceActivity.CommonModels

Структурата е показана на фиг. 2.8.

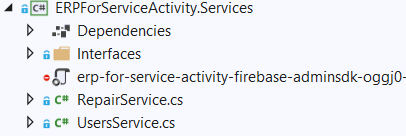


фиг. 2.8

В този модул се съдържат всички модели, които са свързани с попълването на форми, промяна на данни и визуализиране на данни. Разделен е на две части – binding models и view models. В binding models се съдържат всички модели, които се добавят или променят данни. Във view models се съдържат всички модели, които се използват за визуализиране на данни.

* + 1. ERPForServiceActivity.Services

В този модул се съдържат всички класове съдържащи бизнес логиката на приложението. В папката Interfaces се съдържат всички интерфейси, които са имплементирани от съответните класове. За да може да се изплолзват сървисите или класовете, всеки от тях имплементиращ съответен интерфейс, трябва да са добавени в главния модул на приложението. Структурата е показана на фиг. 2.9.



фиг. 2.9

ТРЕТА ГЛАВА

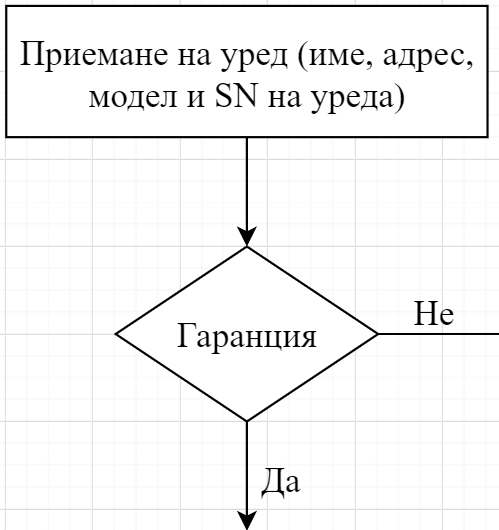
Същност и реализация на приложението

* 1. Какво е рекламация
  2. Приемане на уред

Първата стъпка е приемането на уред в оторизиран сервиз. За да се приеме един уред, са необходими следните данни:

* Име и адрес на клиента – предоставят се, защото понякога трябва да се прави ремонт в дома на клиента, понеже уредът не може да бъде занесен до сервиза.
* Модел и сериен номер на уреда – тези данни се предоставят, защото ако е необходимо поръчването на части за уреда, чрез тях се разбира кои са точните части за него.
  1. Проверка дали уредът е в гаранция или не

За да се провери дали един уред е в гаранция, трябва да се предоставят гаранционна карта и касов бон, които са доказателство, че уредът е купен на посочената от клиента дата. Това е показано на фиг. 3.1.



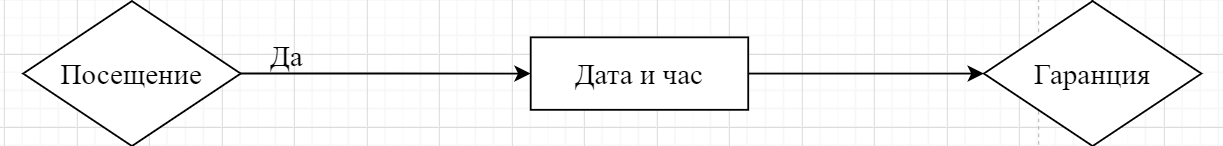
фиг. 3.1

* + 1. В гаранция

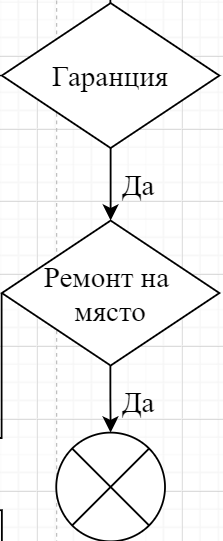
Ако се установи, че уредът е в гаранция, той трябва да бъде върнат на клиента в рамките на 30 дни от датата на приемането на уреда. Процесът по обработка на този тип ремонти е посочен в по – долу описаните стъпки.

* + - 1. Посещение на адрес

Ако ще има посещение на адрес, първо трябва да се уговори дата и час на посещението. Когато се отиде на адрес, най – важното е да се провери дали уредът е в гаранция. Ако е, се проверява дали уредът може да се ремонтира без да се носи до сервиз и ако не е нужно, се извършват нужните ремонтни дейности по уреда и ремонта се приключва, но клиентът може да откаже ремонт на място и тогава уредът се транспортира до сервиза. Ако не е, ремонтът става извънгаранционен. Това е показано на фиг. 3.2 и фиг. 3.3.



фиг. 3.2



фиг. 3.3

* + - 1. Диагностика на уреда

Диагностика на уреда се прави, когато няма посещение на адрес или когато няма да се прави ремонт на място. Отново се проверява дали уредът е в гаранция. Ако е, се прави допълнителна диагностика. Ако не е в гаранция, се издава протокол за отказ гараняция и уредът се ремонтира като извънгаранционен.

* + - 1. Части по ремонт

За един ремонт може да се наложи използването на части. Проверява се дали са необходими части за този ремонт. Ако не е нужно тоест нужно е запояване на кабелче или качване на фабричния софтуер за уреда, директно се уведомява клиентът, че ремонтът по уреда е приключил. Ако е необходима част, първо се проверява дали нужната част я има в наличност, се изписва по този ремонт и след извършване на необходимите ремонтните дейности, се уведомява клиентът, че ремонтът е зъвършен. Ако необходимата част я няма в складова наличност, се поръчва към производител. Проверява се дали трябва да се достави частта и ако да – се завежда в склада на сервиза, след което е възможно да бъде изписана по даден ремонт, след което се извършват необходимите ремонтнти дейности. Ако частта не трябва да се доставя, това означава, че уредът вече има три гаранционни ремонта и има регистриран четвърти основателен се издава протокол за замяна на уреда и ремонтът приключва.

* + - 1. Уведомяване на клиента за приключил ремонт

След приключването на всички ремонтни дейности по поръчката, се уведомява клиентът, че уредът е ремонтиран. След това се пита дали клиентът желае уредът му да бъде доставен до дома му. Ако желае доставка, трябва да уточни дата и час на посещение и се доставя до клиента. Ако не желае доставка, това означава, че клиентът ще дойде до сервиза. И в двата случая дали с или без доставка, клиентът получава уреда си и поръчката приключва. Това е показано на фиг. 3.3.

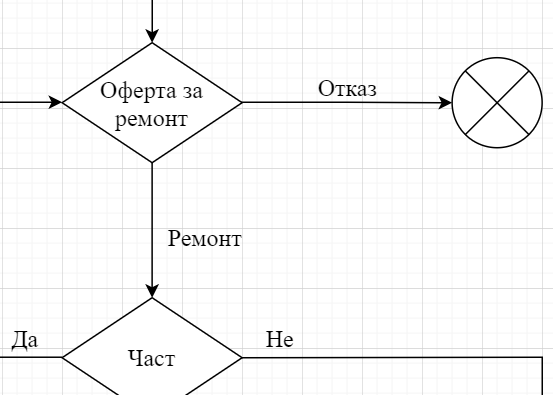
фиг. 3.3

* + 1. Извън гаранция

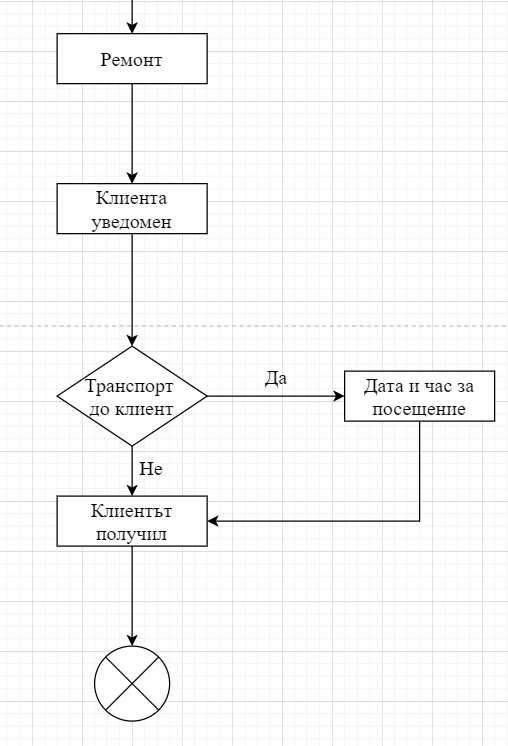
Гаранция на уред се отхвърля, защото има дефект, причинен от клиента и няма срок, в който уредът да бъде върнат на клиената. По една поръчка гаранция може да се отхвърли при следните етапи:

* При приемането на уреда в сервиза
* Когато трябва да се извърши ремонт на място
* След проведена диагностика

Във всеки от горепосочените случаи процедурата по провеждане на извънгаранционен ремонт е една и съща. Започва се с предлагане на оферта за извънгаранционен ремонт. Ако клиентът откаже, уредът се връща на клиента и поръчката приключва. Ако се съгласи, започват ремонтнтите дейности по уреда. Проверява се дали е необходима част за поръчката и ако не е, се извършват ремонтните дейности, след което клиентът се уведомява, че ремонтът е приключил. След това се пита дали клиентът иска доставка на уреда и ако да трябва да уточни дата и час на доставка, а ако не желае доставка, клиентът трябва да дойде до сервиза и да си вземе уреда. След това поръчката приключва. Ако обаче е необходима част по поръчката, се изпълнява процедурата по поръчване на част от гаранционния ремонт. Процедурата е показана на фиг. 3.4 и фиг. 3.5.



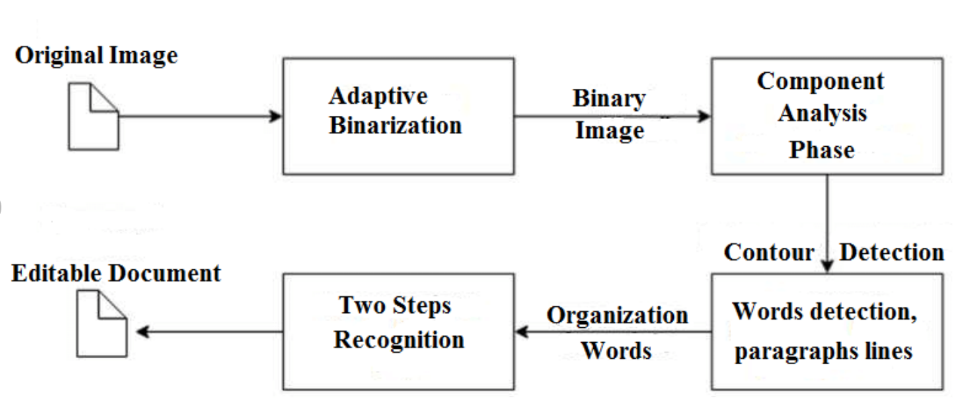
фиг. 3.4



фиг. 3.5

* 1. Принципът на работа на Google Cloud Vision API

Принципът на работа на Google Cloud Vision API е на принципа на Tesseract Optical Character Recognition (TOCR). Принципът на работа е показан фиг. 3.6.

фиг. 3.6

Намирането на думи e направено чрез организиране на текстови линии в петна, а линиите и регионите се анализират за фиксиран висок или пропорционален текст. Текстовите линии се разбиват на думи по различен начин според вида на разстоянието между знаците. След това разпознаването протича като процес с два прохода. В първия пропуск се прави опит за разпознаване на всяка дума от своя страна. Всяка дума, която е задоволителна, се предава на адаптивен класификатор като данни за обучение. След това адаптивният класификатор получава шанс за по-точно разпознаване на текст по-надолу в страницата.

* 1. Обяснение припципа на работа на Paypal API

PayPal предлага REST API за нови интеграции. Тези API използват HTTP методи, структура на RESTful крайни точки, протокол OAuth 2.0 и полезни натоварвания във формат JSON.

PayPal REST API използва протокола OAuth 2.0 за to authorize calls. OAuth е отворен стандарт, който много компании използват, за да осигурят сигурен достъп до защитени ресурси.

Когато e създадена sandbox или на live REST API приложение, PayPal генерира набор от клиентски идентификатор на OAuth 2.0 и секретни идентификационни данни за sandbox or live environment. Когато бъде осъществено извикване с маркер за достъп, set the Authorization header to these credentials for the environment in which you're making the call.

В замяна на тези идентификационни данни, сървърът за оторизация на PayPal връща вашия маркер за достъп в полето access\_token:

{

"scope": "scope",

"access\_token": "Access-Token",

"token\_type": "Bearer",

"app\_id": "APP-80W284485P519543T",

"expires\_in": 31349,

"nonce": "nonce"

}

Включете този маркер на приносител в заглавието за упълномощаване със схемата за удостоверяване на Bearer в обажданията на REST API, за да докажете вашата самоличност и да защитете ресурсите си. Тази заявка за пример включва:

curl -v -X GET https://api.sandbox.paypal.com/v1/invoicing/invoices?page=3&page\_size=4&total\_count\_required=true \

-H "Content-Type: application/json" \

-H "Authorization: Bearer Access-Token"

Access tokens имат ограничен живот. Полето expires\_in съдържа броя секунди, след което маркерът изтича. Например маркер за достъп със стойност на изтичане на 3600 изтича за един час от момента на генериране на отговора.

За да се открие, access token изтича, написаният код трябва да обработва следните две неща:

* Да се следи стойността на expirеs\_in в token response.
* Трябва да обработва http 401 unauthorized. The API endpoint issues this status code when it detects an expired token.

Access token-a може да се използва, докато не изтече. След това трябва да се генерира нов token.

* 1. Принципа на работа на спедиторското API на Еконт

Автоматичното създаване на товарителница е едно от основните предимства на интеграцията на магазина със системата на Еконт. По този начин търговецът ограничава до минимум възможността за грешка при доставка на артикули от магазина и спестява ресурс за ръчно попълване на данните.

Генерирането на товарителница за пратка е съвкупност от зададената конфигурация на пратката, генерирана от системата, и пристигнала под формата на заявка в избрания формат.

* + 1. Необходими изисквания

За да генерирате успешно товарителница, трябва да разполагате със следната информация:

* Валидирани с нашата система място за изпращане и място за получаване. Валидацията включва:
  + Местоположение на изпращане (офис или Еконтомат с правилен код, валиден адрес, сравнен с наличните в базата им данни);
  + Време на изпращане и получаване (т.е. да е проверено дали в избраните дни и часове за получаване и изпращане на пратките куриерите и офисите ни работят);
* Информация за пратката;
* Брой части и описание;
* Валидни типове услуги, които получателят искат да бъдат използвани на база формата на пратката, местата на изпращане и получаване и свързаните допълнителни услуги
* Валидни допълнителни услуги, които да бъдат добавени към доставката (например Преглед, тест и избор);
* Валидно Указание за връщане на пратката в случай на частична доставка или отказана доставка.
  + 1. Интеграция със SOAP/JSON

В зависимост дали ще бъде създадена една или повече товарителници, следва съответно да използват следните методи – createLabel или createLabels. Важно уточнение е, че тези методи се използват и за начисляване на цена и е важно генерирането и валидирането на товарителница, трябва да се окаже конкретния mode – create.

При подаването на заявка, методът изисква следното съдържание:

* Клас ShippingLabel с пълното съдържание на съответните му компоненти;
* В случай че е заявен и куриер, който да дойде да вземе пратката, трябва да бъдат подадени параметрите requestCourierTimeFrom (от колко) и requestCourierTimeTo – до колко часа да дойде куриерът, за да вземе пратката от зададения адрес за взимане на пратката;
* mode – или цел на подадената заявка. Параметърът може да има 3 стойности:
  + calculate – за изчисляване на цена;
  + validate – за валидиране на информация за товарителница;
  + create – за създаване на товарителница.

За да бъде създадена успешно товарителница, за да бъде успешно получена цена и да бъде валидирана самата товарителница, трябва да бъдат изпратени следните параметри:

* Данни за подател (senderClient);
* Адрес на подател (senderAddress);
* Данни за получател (receiverClient);
* Заявка за куриер, ако е налична такава (requestCourierTimeFrom и requestCourierTimeTo);
* Брой части на пратката (packCount);
* Тип на пратката (shipmentType);
* Тегло на пратката (weight);
* Описание на пратката (shipmentDescription);
* Какъв тип е заявката (mode) – за създаване на пратка следва да бъде create
* Инструкции за връщане (клас ReturnInstructionParams)

Към заявката за генериране на товарителница, може да бъдат добавени и класове, с които да бъдат маркирани:

* Указания към приемащ и предаващ куриер (Instruction)
* Допълнителни услуги към пратката (ShippingLabelServices)
* Опис на пратката (PackingListElement)

Съдържанието на класа ShippingLabel е показано в табл. 3.1.

табл. 3.1

|  |  |
| --- | --- |
| **Параметър** | **Описание** |
| shipmentNumber | Номер на пратката |
| previousShipmentNumber | Номер на предходна пратка |
| previousShipmentReceiverPhone | Телефон на получател на предходна пратка |
| senderClient | подател (клиентски профил) |
| senderAgent | Упълномощено лице за изпращане на пратката от клиентски профил |
| senderAddress | Адрес на подател |
| senderOfficeCode | Код на офис за оставяне на пратката (виж Избор на офис за доставка) |
| e-mailOnDelivery | Имейл за уведомление при доставяне на пратката |
| smsOnDelivery | Индикатор дали към пратката да бъде добавено известяване със SMS за получател (Виж имейл или SMS известяване) |
| receiverClient | Получател |
| receiverAgent | Упълномощено лице на получател |
| receiverAddress | Адрес на получател |
| receiverOfficeCode | Код на офис за получаване на пратка (виж Избор на офис за доставка) |
| receiverProviderID | ID на получател |
| receiverBIC | BIC на банката на получател |
| receiverIBAN | IBAN на сметка на получател |
| envelopeNumbers | Брой пликове |
| packCount | Брой части на пратката |
| shipmentType | Тип на пратката (виж Типове пратки, с които искате да доставяме поръчките) |
| weight | Общо тегло на пратката |
| sizeUnder60cm | Предоставя информация дали и трите размера (ширина, височина и дължина) на пратката са под 60 см. Стойността на параметъра трябва да бъде „1“, ако искате да изпращате пощенска пратка като тип (shipmentType) |
| shipmentDimensionsL | Дължина на пратката |
| shipmentDimensionsW | Ширина на пратката |
| shipmentDimensionsH | Височина на пратката |
| shipmentDescription | Описание на пратката |
| orderNumber | Номер на поръчката |
| sendDate | Дата на изпращане на поръчката |
| holidayDeliveryDay | Кога да бъде доставена пратката – да бъде ли доставена в следващия възможен ден, дори и да е празник (отбелязвате със стойност на параметъра halfday), в следващия нормален работен ден (workday) или на специфична дата |
| keepUpright | Показва дали пратката трябва да бъде транспортирана изправена |
| services | Допълнителни услуги, маркирани с класа ShippingLabelServices (виж Допълнителни услуги за търговеца и клиентите на магазина) |
| instructions | Указания, подадени с класа Instruction (виж Указания към куриера, който доставя пратката) |
| payAfterAccept | Показва дали пратката е с Вариант Преглед (виж Преглед) |
| payAfterTest | Показва дали пратката е с Вариант Преглед и тест (виж Преглед и тест) |
| packingListType | Дава информация на системата какъв тип е описанието на съдържанието на пратката (file – отделен файл, digital - дигитален, loading – в товарителница) |
| packingList | В случай че описанието на съдържанието на пратката е digital, следва да подадете клас PackingListElement към заявката, в който да опишете всяка отделна част на самата пратка (виж Преглед, тест и избор) |
| partialDelivery | Показва дали да бъде позволена частична доставка, т.е. дали ще сте използвали услугата Преглед, тест и избор (виж Преглед, тест и избор) |
| paymentSenderMethod | Как ще плати подателят: оставете празно, ако подателят няма да плаща услугата cash – ако плащането ще е в брой credit – ако плащането ще е по договор |
| paymentReceiverMethod | Как ще плати получателят услугата: оставете празно, ако получателят няма да плаща cash – ако плащането ще е в брой credit – ако плащането ще е по договор |
| paymentReceiverAmount | Каква сума ще плати получателят (виж Плащане на пратката) |
| paymentReceiverAmountIsPercent | Какъв процент от сумата за доставка следва да плати получателят |
| paymentOtherClientNumber | Клиентски номер, в случай че плащането ще е от трета страна |

Примерна JSON заявка за генериране на товарителница:

{

"label":{

"senderClient":{

"name":"Иван Иванов",

"phones":["0888888888"]

},

"senderAddress":{

"city":{

"country":{

"code3":"BGR"

},

"name":"Русе",

"postCode":"7012"

},

"street":"Алея Младост",

"num":"7"

},

"receiverClient":{

"name":"Димитър Димитров",

"phones":["0876543210"]

},

"receiverAddress":{

"city":{

"country":{

"code3":"BGR"

},

"name":"Русе",

"postCode":"7010"

},

"street":"Муткурова",

"num":"84",

"other":"бл. 5, вх. А, ет. 6"

},

"packCount":1,

"shipmentType":"PACK",

"weight":5,

"shipmentDescription":"обувки"

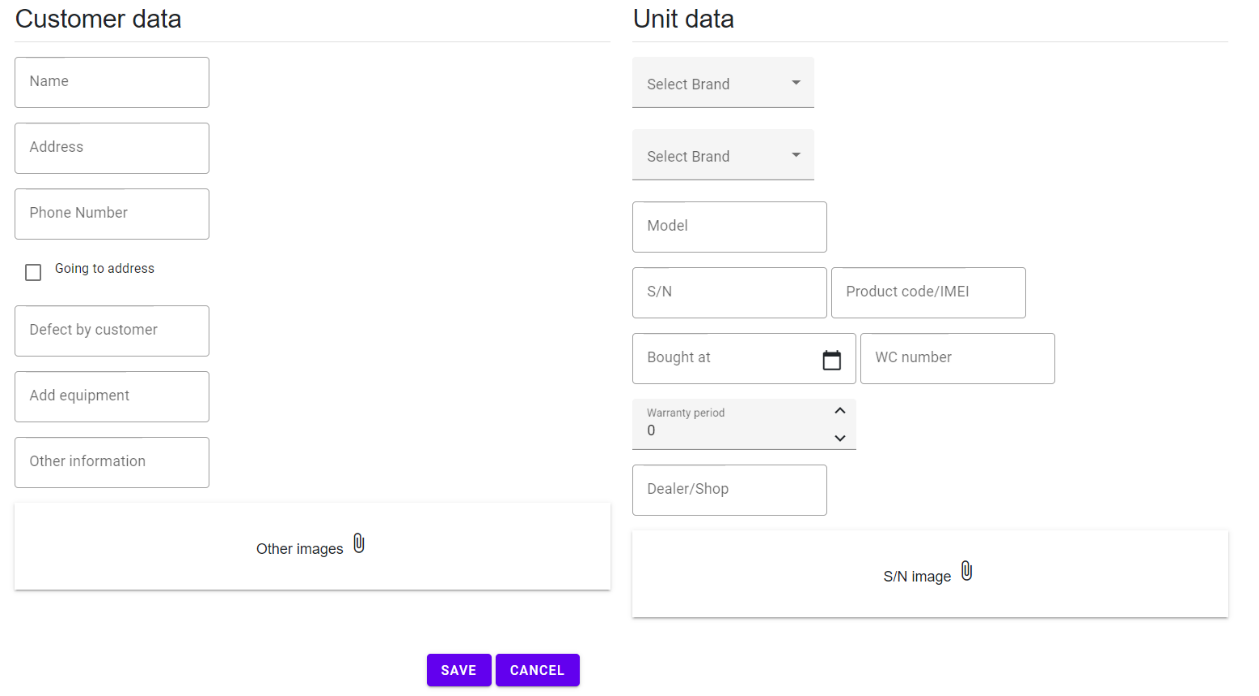
},

"mode":"create"

}

* 1. Реализация на приложението
     1. Дейности, свързани с поръчки или ремонтни карти
        1. Създаване на поръчка

Формата за създаване на поръчка е показана на фиг. 3.7.



фиг. 3.7

В секцията Customer data се съдържат всички полета, които са свързани с данни за клиента. Описанието на всяко поле е представено в табл 3.2.

|  |  |
| --- | --- |
| Текстово поле | Обяснение |
| Name | Двете имена на клиента |
| Address | Адреса на клиента |
| Phone number | Телефонът на клиента |
| Going to address | Дали ще има посещение на адрес |
| Defect by customer | Описание на проблема според клиента |
| Add equipment | Каква е комплектацията на уреда |
| Other information | Забележки по ремонта |
| Other images | Допълнителни снимки (гаранционна карта, касов бон, снимки на дефекта) |

табл 3.2.

В секцията Unit data се съдържат всички данни за уреда. Обяснение за всяко поле е представено в табл. 3.3.

|  |  |
| --- | --- |
| Текстово поле | Обяснение |
| Brand | Марката на уреда |
| Type | Типът на уреда |
| Model | Моделът на уреда |
| S/N | Серийният номер на уреда |
| Product code/IMEI | Продуктовият код на уреда (IMEI за мобилни телефони) |
| Bought at | Датата на покупка на уреда |
| WC number | Номер на гаранционната карта на уреда |
| Warranty period | Продължителност на гаранцията |
| Dealer/Shop | Откъде е закупен уредът |
| S/N image | Снимката на стикера на уреда |

табл. 3.3

При отварянето на страницата се взима най-новодобавеното Id на ремонтната карта и се увеличава с 1, за да няма дублиращи се идентификационни номера на ремонтните карти. След това се инициализират следните променливи:

* bool showResultFromOcr – използва се за отваряне на прозореца с резултатите от четенето на информация от снимка (първоначална стойност = false);
* string imagePath – пътеката до снимката на стикера на уреда (първоначална стойност = string.Empty (“”));
* AddRepairBindingModel repair – моделът, чрез който ще бъдат предадени данните на модела, който ще комуникира с базата данни. Съдържа следните полета:
  + CustomerName – съдържа информация за името на клиента.
  + CustomerAddress – съдържа информация за адреса на клиента.
  + CustomerPhoneNumber – съдържа телефонния номер на клиента
  + DefectByCustomer – какво е проблемът според клиента
  + GoingToAddress – дали ще има посещение на адрес
  + InWarranty – дали уредът е в гаранция
  + ApplianceBrand – марка на уреда
  + ApplianceType – тип на уреда (печка, пералня, хладилник, телевизор и т.н.)
  + ApplianceModel – съдържа данни за модела на уреда. Например модел на LG TV съдържа следните данни:
    - Първите две цифри – размера на екрана в inches
    - Трети знак (буква) – резалюция на дисплея
    - Четвърти знак (буква) – година на производство
    - Пети знак (цифра) – серия
    - Шести знак (цифра) – модел в серията
    - Седми знак (цифра) – иноформация за дизайн
    - Осми знак (цифра или буква) – екстри или цвят

Пример: 55UK750V е 55 inches TV, с UHD резолюция, произведен през 2018, горна серия, 5 модел в серията, черен цвят, с тунер DVB -T/T2/C/S/S2

OLED TV имат различна абревиатура.

* + ApplianceSerialNumber – серийния номер на уреда. Например серийния номер на уред LG съдържа следните данни:
    - Първи знак (цифра) – година на производство
    - Втори и трети знак (цифри) – месец на производство
    - Четвърти и пети знак (букви) – завод производител
    - Шести и седми знак (букви) – регионално предназначение
    - Последни пет (букви или цифри) – пореден производствен номер

Пример: 712WRVA2Х615 е произведен през 2017 декември, в Полша, кодовете на заводите се знаят само от производителя, пореден производствен номер(2X615).

* + ApplianceProductCodeOrImei:
    - Product code – съдържа данни за модела, региона, за който е предназначен уреда (Европа, Азия и т.н.), цветови и други характеристики.
    - IMEI (International Mobile Equipment Identity) – уникален номер за идентифициране на мобилни телефони, който се изписва на екрана като се въведе \* # 06 # на клавиатурата или заедно с друга системна информация в менюто за настройки на операционните системи. В него се съдържа информация за модел, модел код и сериен номер. Тази информация може да бъде достъпена само през официалния сайт на производител само ако има оторизиран достъп.
  + ApplianceEquipment – каква е окомплектацията на уреда. Например за телевизор – дистанционно, стойка, самият телевизор, user manual.
  + BoughtFrom – магазин или дилър, от който е купен уредът.
  + WarrantyCardNumber – номер на гаранционна карта на уреда.
  + WarrantyPeriod – продължителност на гаранцията на уреда, даваща се в месеци.
  + BoughtAt – дата на покупка на уреда.
  + AdditionalInformation – забележки към ремонта.
  + ModelSNImage – полето, което съдържа информация за снимката на стикера на уреда.
  + OtherImages – масив, който съдържа информация за всички останали качени снимки.
* ResultFromOCRBindingModel result – моделът, чрез който ще бъдат предавани данните от взимането на информация от снимка. Съдържа следните полета:
  + ApplianceBrand – марката на уреда.
  + ApplianceType – типът на уреда.
  + И други полета.

Когато бъде засечено, че е качена снимката на стикера на уреда, се прави заявка към API контролера, за да бъдат взети необходимите данни, като това се прави като се пуска нова нишка и се взима резултата от изпълнението ѝ. Това се прави така с цел бързодействие. Методът Task.Run създава нишка, която се добавя в опашката на чакащите задачи, минава през thread pool и се записва в опашката на изпълнените задачи. В метода, който се извиква при пращане на заявка към контролера, в който се случва следното:

* Създава се обект от типа ResultFromOCRBindingModel, който ще бъде върнат след изпълнението на целия метод.
* Взимат се regex-ите, което се прави като се предостави марката на уреда и типа на уреда.
* Създава се обект от типа ImageAnnotatorClient чрез ImageAnnotatorClient.Create(), защото класът е абстрактен, който е предоставен от библиотеката Google.Cloud.Vision.V1, която е използвана за взимане на информация от снимка.
* Взима се пътеката на снимката на стикера чрез метода GetImagePath(string, int), който от посочената директория взима пътеката на файла на посочения индекс.
* Създава се обект от типа Google.Cloud.Vision.V1.Image чрез Google.Cloud.Vision.V1.Image.FromFile, който очаква като аргумент пътеката до файла.
* Създава се обект от типа IReadOnlyList<EntityAnnotation> чрез client.DetectText, очакващ снимката, от която ще взима резултатите, а те се записват в новосъздадения обект.
* Итерира се през колекцията от предната точка и се гледа дали има съвпадение с по-горе посочените regex-и.
* Резултатният обект се връща.

След като приключи изпълнението на нишката, се променя стойността на showResultFromOcr на true. Показва се диалог, на който се проверява за грешки като например заместването на 0 с О. След като се провери от човешка гледна точка дали има грешки, се натиска бутона Оk, който затваря диалога. След това се извиква метода StateHasChanged, който е наследен от ComponentBase класът, който е наследен от всеки новосъздаден Razor Component. Това се прави с цел да бъде real-time.

Ако бъде засечено, че се на полето Other images има промяна, всички файлове се прехвърлят на полето OtherImages в класа AddRepairBindingModel.

Когато се натисне бутона Save, се прави Http Post заявка към контролера, при което се отива на метода UploadRepair в контролера. В този метод се извиква метода от сървиса за ремонтите, на който се подава два аргумента – първият е името на сервиза, а вторият – обект от типа AddRepairBindingModel, който съдържа цялата информация за поръчката. Методът в сървиса изпълнява следното:

* Създава се обект от типа FirestoreDb, който ще бъде използван за връзка и комуникация с базата данни.
* Взима се Id на документа, на който който ще името на сервиза, съвпада с името на името от подадения аргурент.
* Създава се обект от типа CollectionReference, който ще посочи къде ще се качат данните.
* Създава се обект от типа Repair, който е модела, който ще комуникира с базата данни, като в конструктора се подава втория аргумент на метода.
* Стойността на Id на ремонта се променя на Id на ремонта на подадения ремонт.
* След това се започва асинхронна транзакция, в която се обновява най-новото Id на ремонта за кой сервиз и на съответната колекция се качва ремонта.
  + - 1. Взимане на всички ремонти

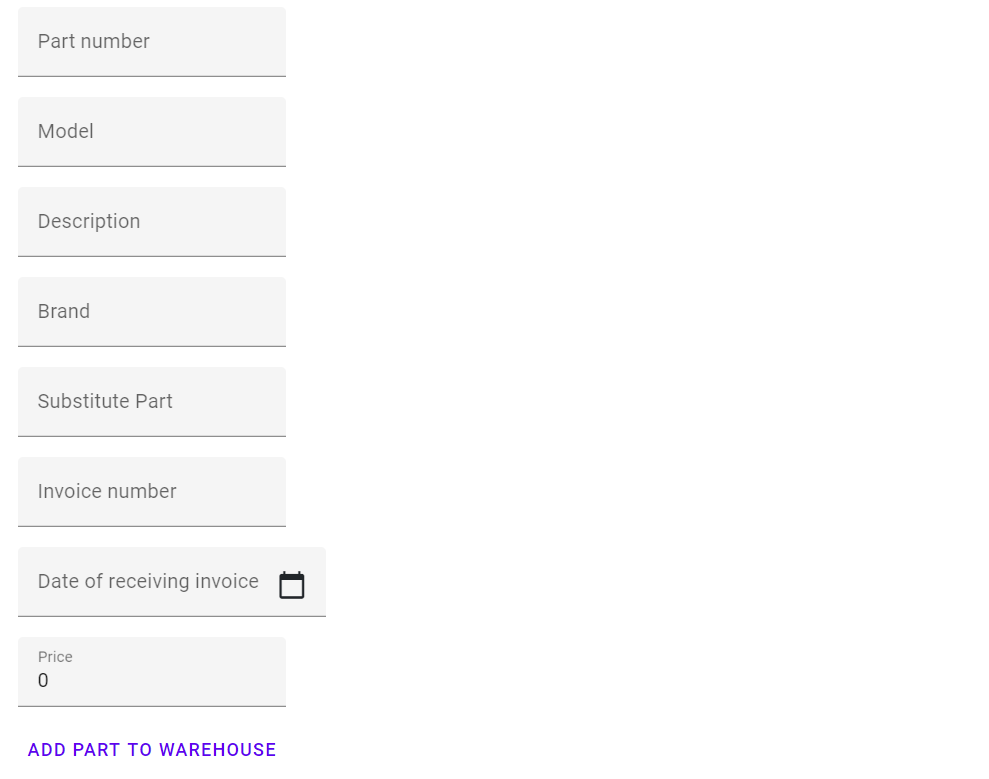
Когато се отвори страницата, която визуализира всички качени ремонти, се взимат всички ремонти, чието име е равно на подадения path variable serviceName. Методът прави следните действия:

* Създава се обект от типа FirestoreDb чрез config.GetFirestoreDb, в който се настройва и се взима инстанция на FirestoreDb. Взима се Id на документа чрез метода GetDocumentId.Result, защото методът е асинхронен и връща Task<T>, където Т е типът, който връща.
* Създава се List<RepairViewModel>, който ще бъде върнат и ще съдържа всички резултати.
* След това се създава Query, което взима всички документи в колекцията за ремонтите от базата данни и след това се изпълнява.
* Итерира се през резултатите и се проверява дали съществува и ако да се конвертира към RepairViewModel и се добаву в списъка.
* Списъкът се връща подреден по Id на ремонта във възходящ ред.

След като се вземат, се подреждат в таблица, на която бройката на ремонтни карти може да се променя. При двоен клик върху който и да е ред, се отваря цялата ремонтна карта. Във визуализацията има и падащо меню за статусите, бутон за добавяне на част като при натискането се отваря диалог, който съдържа поле за part number на часттар описание на часста и снимка на частта, понеже за един телевизор например, има няколко основни платки, които са съвместими с този телевизор. Ще съдържа и два бутона – Save и Cancel, които съответно запазват и качват промените и не запазват промените.

* + 1. Дейности, свързани с склада на сервиз
       1. Добавяне на част в склада

Формата за добавяне е показана на фиг. 3.9.



фиг. 3.9

Обяснението на всяко поле е представено в табл. 3.4.

|  |  |
| --- | --- |
| Поле | Обяснение |
| Part number | Идентификационния номер на частта |
| Model | За кой модел става часста |
| Description | Описание на частта |
| Brand | Марка на частта |
| Substitute part | Част заместител |
| Invoice number | Номер на фактурата на частта |
| Date of receiving invoice | Дата на получаване на фактурата |
| Price | Цена на частта |

При отваряне на страницата се създават следните променливи:

* Обектът part от типа AddWarehousePartBindingModel – използва се за пренасяне на данни от формата към back-end. Обяснение за всяко поле е представено в табл.3.4.

табл. 3.4

|  |  |
| --- | --- |
| Поле | Обяснение |
| PartNumber | Уникален идентификационен номер за части. Всеки производител си има негова идентификация. |
| Availability | Наличност в склада |
| Model | Списък, съдържащ за кои модели става частта. |
| Description | Описание на частта |
| Brand | Марка на частта |
| SubstituteParts | Списък, съдържащ всички нейни заместители |
| Price | Цена на частта |
| Invoice | Номер на фактура, с която е пристигнала |
| InvoiceDate | Дата на издаване на фактура |
| ReceivedDate | Дата на получаване на частта |

* substitutePart от типа string – използва се за взимане на част заместител

При натискане на бутона Add part to warehouse се изпълнява метода AddPartToWarehouse, който наличността на частта се променя на 1, датата на завеждане на частта в склада се променя на DateTime.UtcNow, защото например ако частта е поръчана на 20.12.2019г., има време на транспорт, което отнема две седмици например. part.InvoiceDate се променя на същата дата, но в UTC формат. Списъкът на частите заместители се инициализира на new List<string>, както и part.Model и в part.Model се добавя въведения модел. Проверява се дали е въведена част заместител и ако да се добавя в списъка с частите заместители. Последно се прави Http Post заявка към контролера за частите в склада.

След като се засече, че е направена заявка, се извиква метода AddPart, който приема два параметъра - path variable serviceName, която определя за кой сервиз се отнася и AddWarehousePartBindingModel, който съдържа цялата информация за частта. В него се извиква метода от сървиса за складовите части за добавяне на част, който приема двата по – горе посочените параметъра. В метода в сървиса се изпълняват следните действия:

* Създава се инстанция на FirestoreDb, която ще служи за връзка с базата данни
* Създава се инстанция на CollectionReference, която ще се качват в базата данни
* Създава се инстанция на WarehousePart, която ще бъде качена в базата данни
* Създава се инстанция на QuerySnapshot, където се пише
  + - 1. Я
      2. Я
      3. Я
      4. Я
      5. Я
    1. Процеси, свързани със справки в сервиза

Основната функция на справките е с цел преглед на финансите на сервиза. За целта се създава клас BaseCheckUp, който ще съдържа следните полета:

* ServiceName – име на сервиза
* ApplianceBrand – марка на уреда
* ApplianceModel – моделът на уреда

За всички справки резултатът ще зависи дали даден user има права за централен сервиз или не. Ако е с права за централен сервиз, резултатът , който ще върне справката ще бъде за всички централни сервизи и всички останали сервизи в структурата. Ако е с права за нормален сервиз – резултатът ще бъде само за конкретния сервиз. За всяка справка ще бъдат създадени два класа – за централни сервизи и за нормални сервизи.

* + - 1. Справка за труда на техниците

За да бъде реализирана тази справка, се създава клас TechniciаnCheckUpViewModel, който наследява от BaseCheckUp, и който съдържа следните полета:

* TechnicianName – име на техника
* ApplianceType – тип на уреда
* Labor – цена на труда на техник
* PartPrice – цена на частта
* PartNumber – идентификационния номер на частта

След това се създава класът TechniciаnCheckUpCentralViewModel, който ще наследява от TechniciаnCheckUpViewModel, като ще има едно допълнително поле:

* ServiceName – име на сервиза

Методът, който ще взима резултатът от тази справка, ще прави следното:

* Създава се списък от TechniciаnCheckUpCentralViewModel, в който ще бъдат добавени всички резултати.
* Проверява се дали сегашният user има права за централен сервиз. Ако има, ще се вземат всички техници от всички сервизи в структурата на сервиза. След това се извиква методът, който ще вземе трудът на всички техници като се подава списък от
* За всеки от техниците проверява в колекцията, в която се записват направените промени по ремонтите.
* Създава се
  + - 1. Справка по статус

Целта на тази справка е да се проверят например на колко адреса трубва да се ходи в рамките на даден ден. Създават се класовете RepairStatusCheckUpViewModel, който ще съдържа полетата RepairId (Id на ремонта) RepairStatus (статус на ремонта), и RepairStatusCheckUpViewModel, който ще наследява RepairStatusCheckUpCentralViewModel и ще има едно допълнително поле ServiceName (име на сервиза). Методът ще работи по посочения начин:

* Създава се списък от RepairStatusCheckUpCentralViewModel, в който ще бъдат добавени всички резултати.
* Взимат се всички ремонти, чийто статус е равен на подадения статус.
* В зависимост от правата ще се визмат Id и RepairStatus или Id, RepairStatus и ServiceName.
* Списъкът се връща.
  + - 1. Справка за вложени части в ремонт

Създават се класа UsedPartsInRepairCheckUpViewModel, който съдържа полетата ServiceName (име на сервиза), RepairId (номер на ремонта), ApplianceBrand (име на марката), ApplianceModel (модел на уреда) и Parts (списък с частите). Методът ще изпълнява следните действия:

* Създава се списък, в който ще бъдат добавени всички резултати.
* Взимат се всички ремонти за сервиза.
* За всяко RepairId се взимат всички вложени части от колекцията repairs-parts и се записват в списъка с частите на всеки ремонт.
* Резултатният списък се връща.
  + - 1. Справка за поръчките на сервизи

Създава се класа ServiceRepairsCheckUpViewModel, който наследява от BaseCheckUp, съдържащ полето RepairId (номер на поръчката). В методът се извършват следните действия:

* Създава се списък от ServiceRepairsCheckUpViewModel, в който се бъдат добавени всички резултати.
* Според правата се взимат за всички сервизи или само за един сервиз.
* Всеки от резултатите се конвертира към ServiceRepairsCheckUpViewModel.
* Списъкът се връща.
  + - 1. Справка за труд на техниците по извънгаранционни ремонти

Създава се клас от типа TechnicianOOWCheckUpViewModel, който ще съдържа полетата RepairId (номер на ремонта), Labor (цена труд на техник), PartPrice (цена на частта). В методът ще протичат следните процеси:

* Създава се списък от типа TechnicianOOWCheckUpViewModel, в който ще бъдат записани всички резултати.
* Според правата се взимат ремонтите за всички сервизи или само за един сервиз, чийто статус съдържа OOW(Out Of Warranty).
* Всеки от резултатите се конвертира към TechnicianOOWCheckUpViewModel и се добавя към списъка.
* Списъкът се връща.
  + 1. Логин система

В приложението ще има два типа вход:

* Вход за клиенти – въвеждат се името на клиента и номер сервизна поръчка. След като бъдат въведени тези данни приложението ще връща текущия статус на ремонта.
* Вход за оторизиран персонал – въвеждат се username, password и verification code. Verification code ще се генерира от security модула на приложението, който няма да бъде описан в настоящата дипломна работа, и който ще бъде пращан на имейла на user-a. След влизането с първончалната парола, в рамките на няколко минути паролата трябва да бъде сменена. След това user-a автоматично излиза, за да може да влезе с новата парола.
  + 1. Я

ЧЕТВЪРТА ГЛАВА

Ръководство на потребителя

СЪДЪРЖАНИЕ

Използвана литература

* Описание на ASP.NET Web API – <https://www.tutorialsteacher.com/webapi/what-is-web-api>
* Описание на SignalR – <https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-3.1>
* Описание на Blazor – <https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-3.1>
* Опиcание на Firebase – <https://searchmobilecomputing.techtarget.com/definition/Google-Firebase>
* Описание на Cloud Firestore – <https://firebase.google.com/docs/firestore>
* Описание на Google Cloud Platfrom – <https://intellipaat.com/blog/what-is-google-cloud/#cloud-services>
* Причини избиране Firebase
  + <https://www.altexsoft.com/blog/firebase-review-pros-cons-alternatives/>
  + <https://www.cleveroad.com/blog/a-story-of-firebase-or-your-next-favourite-cloud-based-service>
* Причини избиране Firestore
  + <https://medium.com/@abhinavkorpal/scaling-horizontally-and-vertically-for-databases-a2aef778610c>
  + <https://github.com/vaquarkhan/vaquarkhan/wiki/Difference-between-scaling-horizontally-and-vertically>
* Причини избиране GCP - <https://kinsta.com/blog/google-cloud-hosting/>