# Глава 2. Архитектура на облачна система за управление на поръчки от клиенти

Тази глава разглежда решение от високо ниво, което да съсредоточава върху всички основни потребителски, бизнес и ИТ изисквания. Важна част от глава 2 са градивните елементи и интерфейси, изграждащи системата, както и комуникационните модели, които да ръководят композицията. Сложността на операциите стреми да бъде сведена до минимум. Представени са всички случаи на употреба и бизнес сценарии, съвместно с които се моделират приложенията за обслужване на клиенти. Освен това дизайнът обхваща функционалност, използваемост, устойчивост, производителност, икономически, технологични ограничения, компромиси и естетически проблеми на бекенд частта.

## 2.1. Ключови бизнес процеси и дейности свързани със системата за управление на поръчките

Думата „**архитектура**“ често се използва в контекста от високо ниво, което е отделено от детайлите на по-ниско ниво (Martin et al., 2017). Софтуерният продукт, разглеждан в настоящия труд, се състои от 2 клиентски приложения, които се свързват към разпределена бекенд система, базиранa на микроуслуги, работеща върху множество процеси и сървъри (хостове). Всяка услуга се изпълнява в отделен процес като контейнер, разположен в клъстер от виртуални машини. Това разделение на подсистеми и отделни нива и компоненти цели да постигне разбираемост и лесна поддръжка. Работните рамки са съвместими на всяко ниво, без да се дублират функционалности.

На фигура 2.1 са показани приложенията, които изграждат системата за управление на поръчките от клиенти.

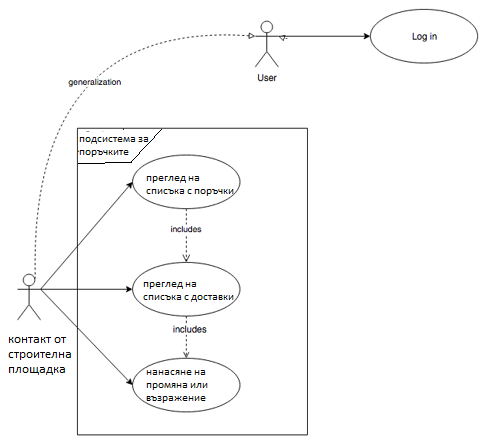


Фиг. 2.1. Диаграма от високо ниво на главните приложения. (разработка на автора)

Тази подточка представя важни случаи на употреба, които са критични за бизнеса и са част от основния домейн. Използвани са UML диаграми на бизнес сценариите. Те идентифицират действия, които очакваме потребителите да направят.

Най-подходящ за взаимодействие с крайните потребители са мобилните приложения. Важни техни характеристики са, че поддържат функции като местоположение, камера и работят с уеб API. Клиентите на фирмата, които се явяват крайните потребители, управляват и проследяват поръчките и доставките в реално време с мобилно приложение. Целта му е да помага с планирането и логистиката, да въздейства върху крайния резултат с информация и данни. Тази информация, на смартфона, трябва винаги да е актуална, тъй като текущото състояние на поръчка и местоположение на доставките се проследява на живо. Други възможности са преглед на история, създаване на нова, промяна или отказване на **не активна** съществуваща поръчка. Приложението може да се разпостранява безплатно чрез Google Play Store и Apple App Store.

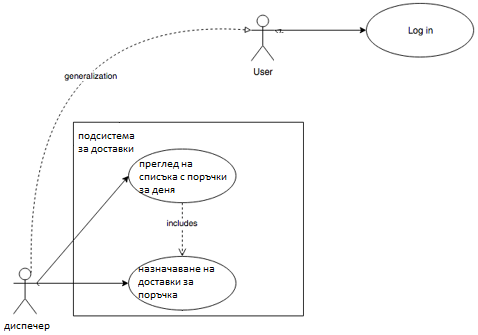
Обхватът на мобилното приложение, насочено към крайните клиенти, включва екран за вход, интерфейс за текущите поръчки и доставки към тях. Също така панел за създаване или промяна на поръчка. Фигура 2.2 представя процесите под формата на диаграма.



Фиг. 2.2. Диаграма на главен бизнес сценарий (разработка на автора)

Уеб порталът е софтуер, насочен към диспечерите, част от цялостната система за управление на транспорта (TMS). Чрез него могат да се създават поръчки и доставки, като същевременно се сравняват, за да се гарантира, че поръчките се доставят от най-подходящото превозно средство. Уеб порталът служи като инструмент за вземане на решения, с предварително зададени предложения, които могат да бъдат одобрени, или отхвърлени и променени, според гледната точка на диспечера на смяна. Вземайки под внимание текущите събития, подсистемите зад уеб портала насрочват за доставка това, което и когато клиентът е поръчал. Те разчитат на правилна информация за поръчка и актуализация на събития. Целта е да се минимизират разходите.

Обхват на уеб портала включва балансиране на работното натоварване на превозните средства, позволява проследяване и коригиране, както на поръчките, така и на доставките, осигурява предварително зададени решения, на база на които диспечерите могат да коригират и контролират броя на доставките. Диспечерите имат възможност да поправят грешни данни, като говорят с клиентите или шофьорите. Същевременно всички промени се отразяват в мобилното приложение.

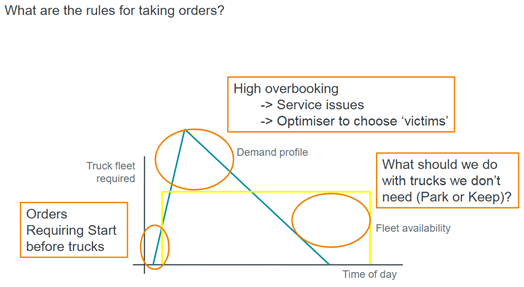


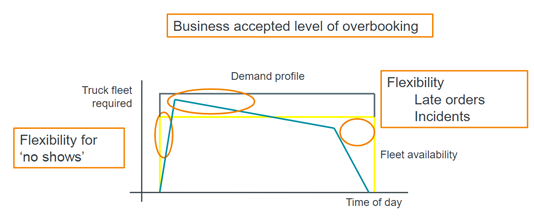
Фиг. 2.2. Диаграма на главен бизнес сценарий (разработка на автора)

Уеб портала прилага усъвършенствани техники като: оптимизация в реално време, оптимизация за оценка на поръчки, оптимизация на предварителното планиране, които се извършват постоянно във фонов режим. Входните данни, идващи от ЕРП, са записите за завод, клиентски местоположения, превозни средства, техните свойства и статуси, поръчки, параметри на оптимизатора и други.

….

Следна фигура представя правила за приемане на поръчки:





Фиг. 2.3. Правила за приемане на поръчки (разработка на автора)

Уеб портала поддържа доклад за късно зареждане е Той служи като обратна връзка към диспетерите. В случай, че клиентските поръчки закъснеят, клиентите могат да бъдат извикани проактивно. Целта е ако все пак има закъснение, по-добре е да клиентите да бъдат уведомени предварително.

Уеб портала предоставя ежедневно експортиране на пробега на всеки камион, въз основа на отчет от базата данни. Автоматизиран интерфейс е проектиран и внедрен, за да замени ръчното извличане на данни. ЕРП предоставя нов функционален модул за получаване на данни за пробега чрез RFC. Веднага след като камион приключи смяната си, всички изминати натоварени и разтоварени маршрути, записани в база данни се експортира със специален идентификатор към ЕРП. Ако това експортиране бива неуспешно, съобщение, съдържащо идентификатора, се появява в полето за съобщения.

## 2.2. Концептуален модел на системата

Концептуалните модели са абстрактни представяния за това как трябва да протича изпълнението на задачите. Те представят визуално концепция или операция. За визуализиране и конструиране на елементите е използван унифицираният език за моделиране (Unified Modeling Language).

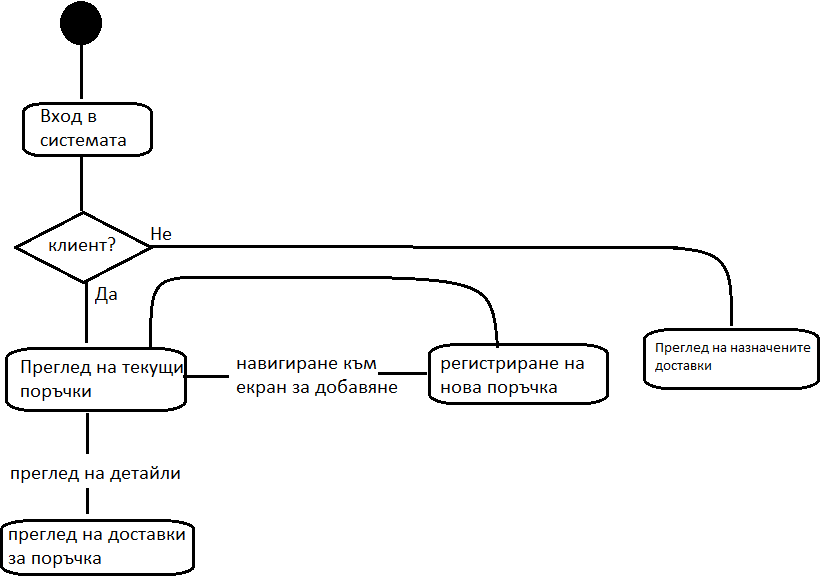
### 2.2.1. Поведенчески диаграми

Поведенческите диаграми идентифицират как различните елементи взаимодействат помежду си.

#### 2.2.1.1 Диаграми за активност UML

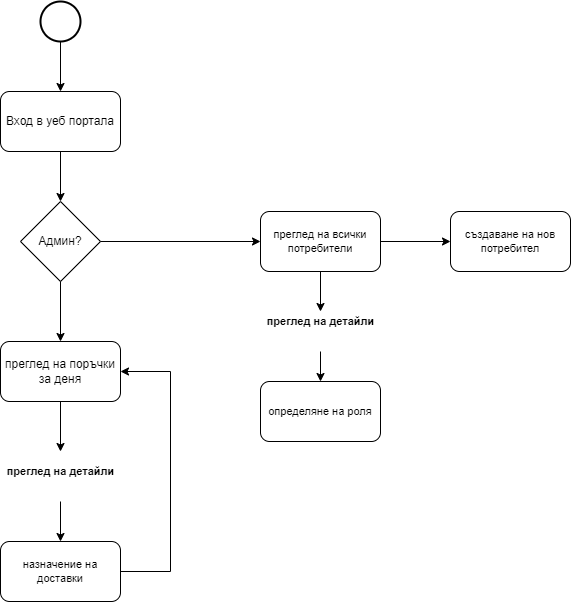
Диаграмите за активност изглеждат много подобни на блок-схемите. Наличието на тези прилики улеснява комуникацията между технически и не-технически лица (stakeholders).

Следната диаграма представя работни потоци и общи операции за мобилното приложение:



Фиг. 2.3. Диаграма на активноста за мобилно приложение. (разработка на автора)

Следната диаграма изобразява потока от операции в уеб портала:

 Фиг. 2.4. Диаграма на активност за уеб портал. (разработка на автора)



Фиг. 2.4. Диаграма на активност на поръчка. (разработка на автора)

#### 2.2.1.2. Диаграма на последователностите UML

Диаграмите на последователностите също са често използвани поведенчески диаграми в UML. Те идентифицират как обектите в система взаимодействат помежду си, за да реализират определена функционалност, като визуализират времевата линия и редът, в който се извършват операциите.



Фиг. 2.7. Диаграма на последователностите. (разработка на автора)

### 2.2.2. Структурни диаграми

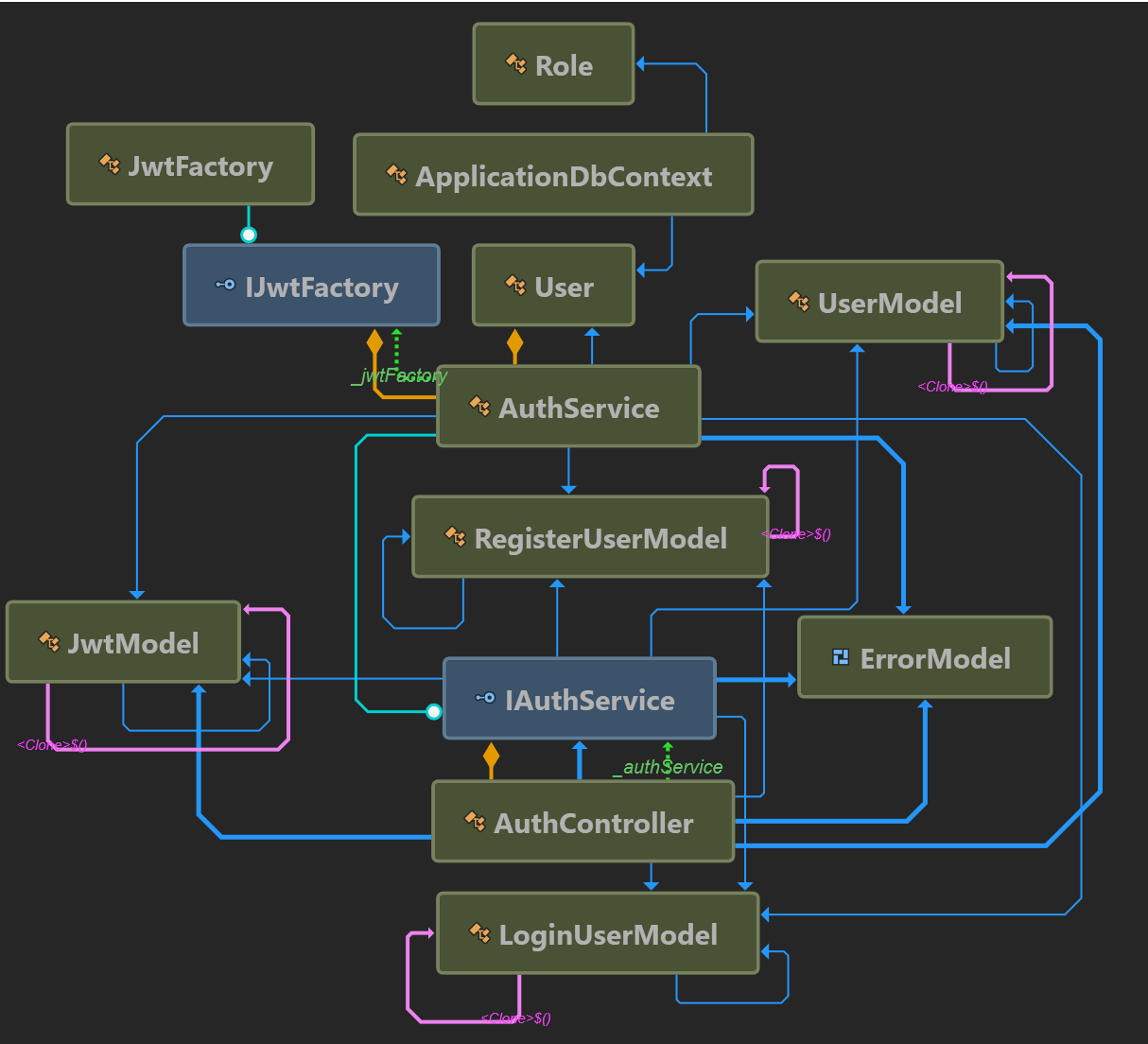
Структурните диаграми помагат за дефиниране цялостната структура системата, подобно на плана, който определя как изглежда една къща. Структурните диаграми моделират как изглежда системата в архитектурно отношение. Те ни помагат да дефинираме „речника“ на системата, гарантират съгласуваност от заинтересовани страни в проекта. Идентифицират различни връзки между различните частти.

Структурните UML диаграми изобразяват елементите на система, които са независими от времето и които предават концепциите и как те се свързват помежду си. Елементите в тези диаграми приличат на съществителните в естествения език.

#### 2.2.2.1. Диаграма на класовете UML

Диаграмите на класове са едни от най-често срещаните, когато става на въпрос за разработката на софтуер. Едно от основните неща, които тези диаграми правят е да идентифицира речника на системата. Например, те определят връзките между обектите, които съответстват на основните съществителни.

Следващата част представя диаграма на класовете, свързани с удостоверяване. Това е процесът на определяне кой има достъп до системата. Елементите от приложението и зависимости, които обслужват тази част са визуализирани на фиг. 2.2. **DbContext** и **ApplicationUser** представляват комбинация от класове, които оперират с базата от данни.  **AccountController** използва тези свойствa чрез **UsersService**, който капсулира логиката по безопасен за използване начин.



Фиг. 2.8. Интерактивна диаграма на класовете. (разработка на автора)

Структурата на папките на приложениео е добре оформена, по следния функционален, управляван от домейн дизайн:

├───**eShop**

│ ├───**src** – индикира път към основния код

│ │ ├───**eShop.Web** - логика за представяне

│ │ ├───**eShop.Business** – бизнес логиката на приложението

│ │ ├───**eShop.Core** – експонирани за клиенти модели

│ │ ├───**eShop.Domain** – вътрешни модели на приложението

│ │ └───**eShop.Persistence** – логика за достъпа до данните

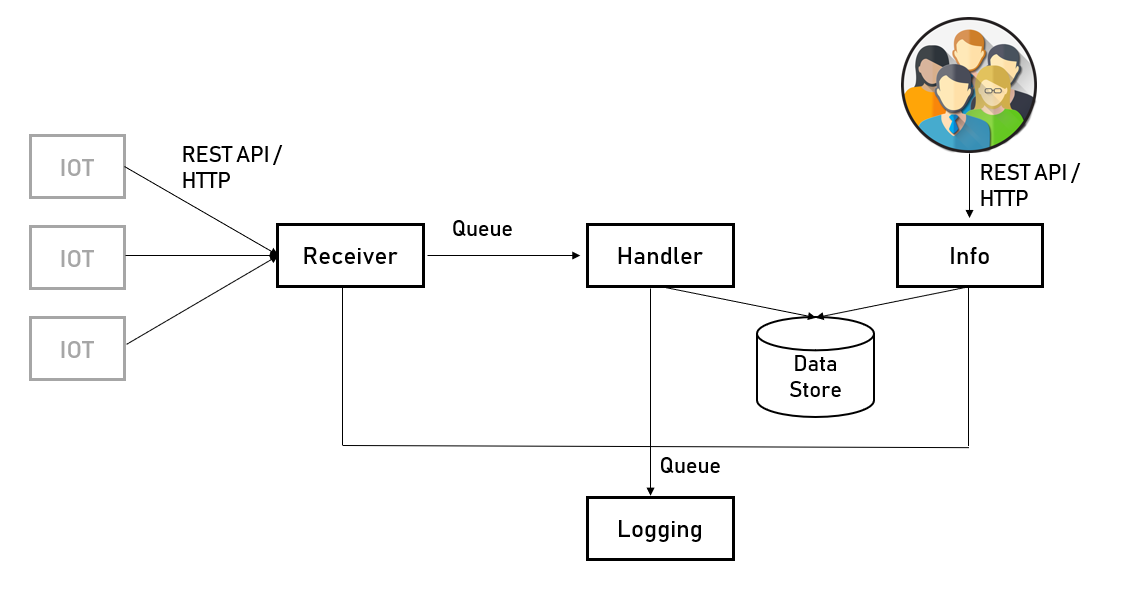
│ └───tests – индикира път към тестовете на кода

│ └───**eShop.Tests** – пълен набор от автоматизирани (интеграционни) тестове

### 2.2.3. Архитектурни диаграми



Фиг. 2.9. Процесен модел на веригата за доставки. (разработка на автора)



Фиг. 2.10. архитектурна диаграма на подсистемата за доставки. (разработка на автора)

Graphical user interface, diagram

Description automatically generated

Фиг. 2.11. Mикросървисната архитектурна диаграма. (разработка на автора)

## 2.3. Функционалност и потребителски интерфейс

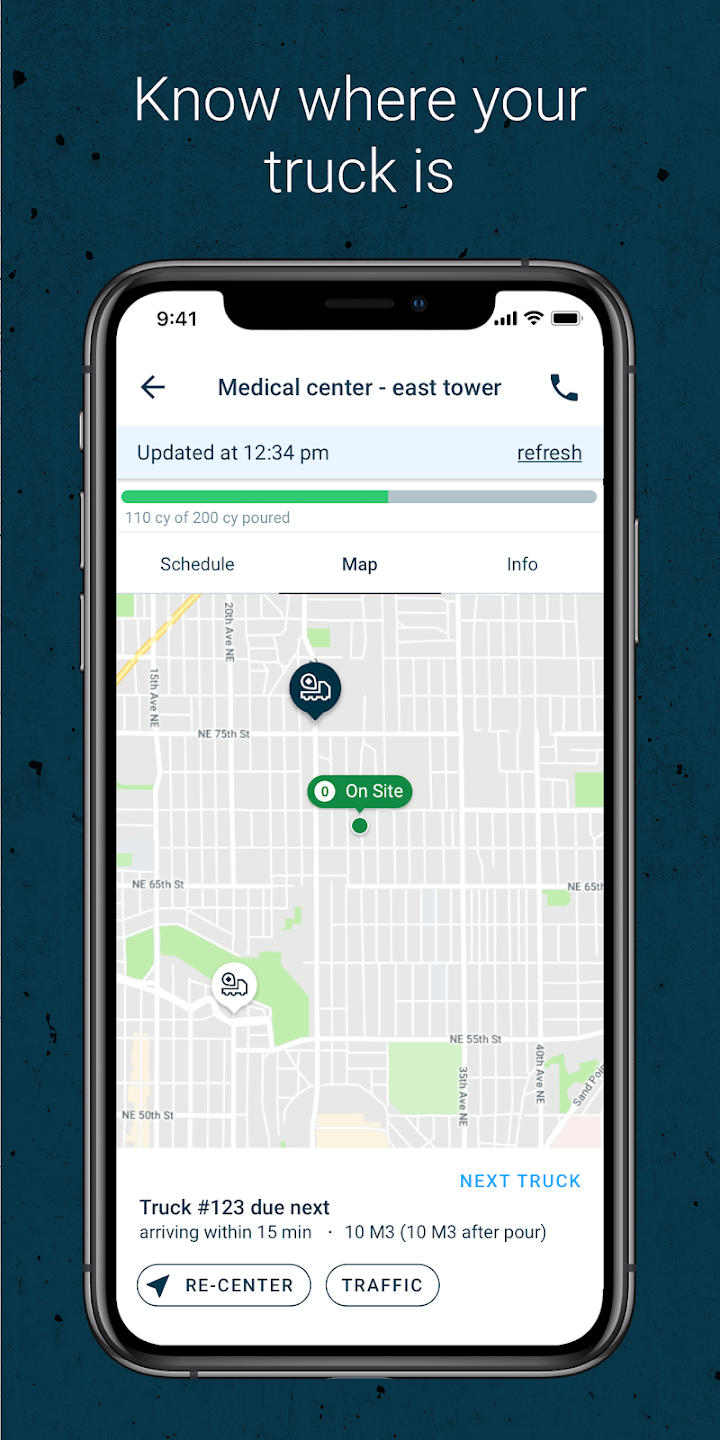
Ясно дефинираните изисквания са основата на успешен проект, тъй като включват набор от процеси като анализ, спецификация и валидиране. Функционалните изисквания са продуктови характеристики, които разработчиците трябва да внедрят, за да позволят на потребителите да изпълнят своите задачи. Като цяло функционалните изисквания описват поведението на системата при определени условия.

Нека започнем с преглед на характеристиките и изискванията на мобилното приложението. Както беше отбелязано, то представлява приложение за поръчки и проследяване на доставки онлайн. Примерна скица на интерфейса на началния екран, след вход, в на потребителското приложение е даден на фиг. 2.10. Представени са основни елементи на заглавна част – име на текущ потребител, инструмент за избор на дата и списък на текущите поръчки.



Фиг. 2.10. Скица на основен екран на приложението. (разработка на автора)

Фигура 2.11. представя детайлна информация за определена поръчка и доставките към нея.



Фиг. 2.11. Скица на екран за информация за поръчка. (разработка на автора)

Регистрирането на нови поръчки ще се осъществява чрез екран в главното меню.

Фигура 2.12 представя използвано на приложението от доставчика. Както беше споменато, мобилното приложение допринася за бързо изпълнение на процесите, сравнително лесно за използване и удобно за работа, чрез функционалностите за достъп до геолокация, навигация, съобщения, телефон. Това допринася за стандарт за сигурност на данни и връзки. Водача може да провери списъка, с предстоящи доставки, назначени към него. Приложението притежава функционалност за уведомяване, в случай че списъкът е експортиран, но няма насрочени задачи.



Фиг. 2.12. Скица на начален екран за доставчика. (разработка на автора)

Посоченият екран съдържа подробности за поръчка, включващ материал, количество, местоположение за товарене и разтоварване и планирани часове. Тъй като, доставката може да бъде анулирана или пренасочена към друга поръчка, остатъкът се докладва и след това бива върнат, използван повторно или отклонен. Също така шофьорът може да съобщи за повреда, като отписването е възможно след изпращане на лог файлове към диспечера.

За да подпомогне автоматизирането и рационализирането на документацията, приложението поддържа функционалност за електронно доказателство за доставка. Това е процес, който създава документацията, валидираща получаването на стоката от клиента. Традиционно POD се осъществява чрез подпис на клиента на физически документи. В случай на липса на подпис трябва да се посочи причина. Мобилното приложение „улавя“ съответните данни и снимки, като това бива последния етап от доставката. Следната фигура представя екрана за тази функционалност.



Фиг. 2.13. Скица на екран за доказателство за доставка (ePOD). (разработка на автора)

Когато материалът е доставен, от клиента се иска да потвърди получаването чрез подпис на мобилното устройство, който се предава заедно с допълнителни данни. ePOD документа се изпраща препраща към ERP.

Графичен интерфейс на уеб портала, представен на фигура 2.14., е предназначен за използване от диспечерите за разпределяне и планиране на работата. Той представя информация за поръчките, които трябва да бъдат доставени, като също така дава пълен контрол върху всички превозни средства. Целта към обслужването на клиенти е, да даде представа за организацията през работния ден, да се определи допустимо ниво на резервиране и да се съобши на поемащите поръчки.





Фиг. 2.13. Главен екран в уеб портала. (разработка на автора)

Изгледът предлага списък, в който всеки ред съдържа инфомрация за камион със съответните доставки, основни данни и данни за състоянието, всички получени поръчки и планирани доставки. Ширината на колоната може да се променя чрез плъзгане, в случай, че ширината е твърде малка. Сортирането е възможно по всяка колона (възходящо/низходящо). GUI актуализира на всеки 10 секунди и след всяка транзакция, като анкетира сървъра, който трябва да отговори на всяка заявка за актуализиране поотделно. Сглобяване на всички нови данни изискват изчислителна мощност, затова броят на заявки към сървър е ограничен. Почти пълната оперативна база данни е съхранена в паметта. В оперативната база данни се поддържа само малък набор от данни. Историческите данни не са необходими за оптимизиране и изпращане.

Уеб порталът служи като инструмент, използван за актуализации на състоянията на пристигане/напускане, натоварване и други за камионите, които към определен момент са без дистанционно предаване на данни.

Уеб порталът поддържа функционалност за времето, което отнема на превозното средство за да стигне от точка А до точка Б. По този начин, той служи като инструмент за разстояние и продължителност на пътуването, като например продължителността на пътуването от завода до мястото на клиента. Пътят обратно може да е различен, заради еднопосочни улици. От камиона може да бъде поискано да се върне някъде другаде. Уеб порталът поддържа вътрешна „Матрица за време и разстояние“ (често наричана „Разстояния и продължителности“ за да съхранява заявените времена и разстояния. На следващата фигура е визуализиран изгледат за маршрутизиране.



Фиг. 2.14. Екран за маршрутизиране. (разработка на автора)

Уеб портала комуникира с камиони и заводи, за текущите статуси на почивки, отчети за състоянието, начало и края на товарите. Информира за задания, анулации и т.н. Уеб портала и мобилното приложение предоставят телематична система, която дава обратна връзка към диспечърите. Уеб портала използва събития за местоположение, за да изгради прогнозната оставаща продължителност на пътуването, от мобилното приложение, което изпраща геокоординати. По подразбиране актуализациите на ETA за отчетено местоположение са на всяка минута, но това може да се промени.

Нефункционалните изисквания често се наричат ​​„атрибути за качество“ на системата. Те са критериите за оценка на това как една софтуерна система трябва да работи.

Следващите точки отбелязват някои от основните изисквания:

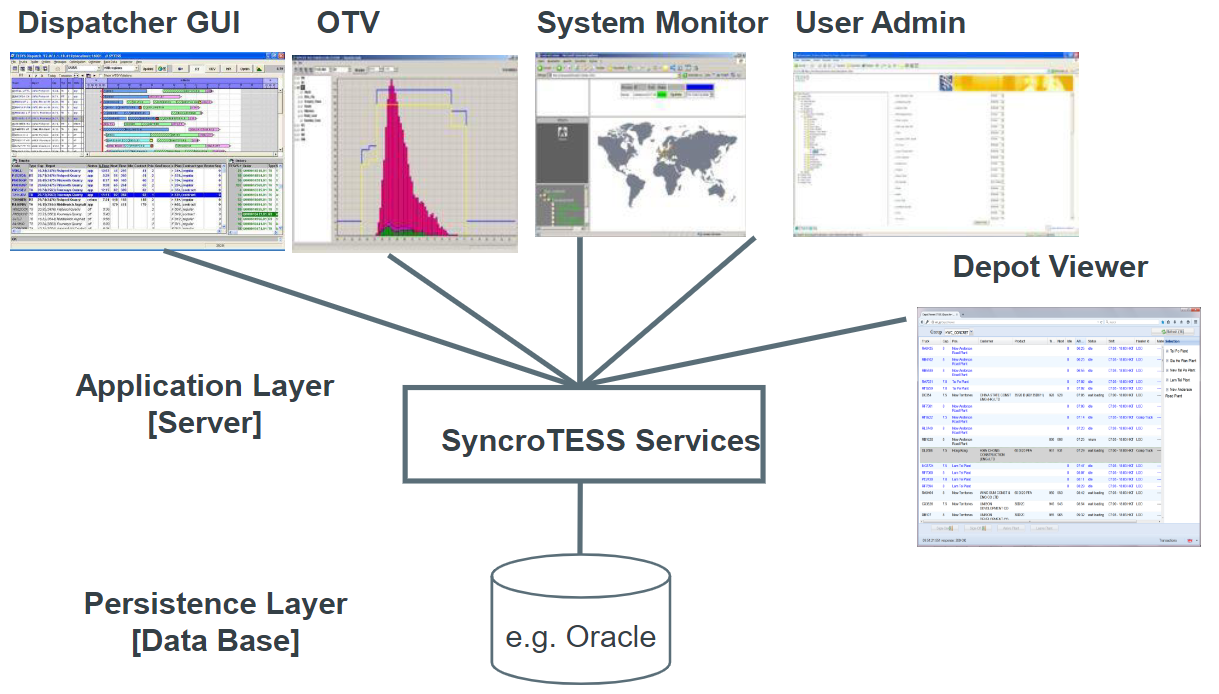
• Системата трябва да е високо-достъпна и да може автоматично да разширява мащаба, за да отговори на увеличаващия се трафик (също така да намалява мащаба, след като трафикът спадне);

• Трябва да осигурява лесен диагностични дневници, за да помогне при отстраняване на неизправности или други проблеми, които би могли да възникнат по време на работа;

• Трябва да поддържа гъвкав процес на развитие, включително подкрепа за непрекъсната интеграция и внедряване (Continuous integration / deployment);

• Трябва да поддържа междуплатформен хостинг и развитие;

TODO: ИЗИСКВАНИЯ ЗА бързодействие и изпълнение / натоварване / ИЗИСКВАНИЯ ЗА ОБЕМ ДАННИ / брой потребители /



Дава изглед на високо ниво какви са изискванията на камиона●Трябва да се използва като основа за Demand SmoothingView върху същите данни, които виждат всички останали●OTV изгледът се променя с промяната на поръчките и извършването на доставкиЕ неоптимизиран изглед на поръчките●Отразява времената от поръчките●По-малко изразителни за големи времеви прозорци за доставка и минимално разпределение на товара Също така позволява лесен достъп, за да видите кога са доставени поръчките●Възможно е да се отговори на въпроси на клиенти като●„Кога ще бъдете тук?“●„Колко вече сте доставено?'

The Material Replenishment(MRP)

За производството на бетон са необходими суровини и SyncroTESSплан за група бетони определя търсенето на бетон:●SyncroTESSекспортира търсенето на готов бетон ●на завод●на продукт●на интервал от половин час●SAP използва тази информация, за да изчисли търсенето на суровини●SAP генерира/актуализира автоматично обобщените поръчки

## 2.4. Kомуникационни модели между програмните интерфейси

Комуникационните технологии са от важно значение за много уеб приложения, в частност системи за електронна търговия или управление на поръчки. Те са част от Световната мрежа, която сама по себе си представлява разпределена система от взаимосвързани ресурси.

Актуалността на изследваната тема се обуславя от тенденцията облачните технологии да се превръщат в инструменти за стратегическа трансформация и дигитализация на бизнеса. Облачните платформи позволяват бърза реализация на иновативните идеи. Това предимство поставя компаниите една стъпка пред конкурентите.

Обект на изследване в настоящия реферат е разпределена информационна система, базиранa на микроуслуги, работеща върху множество процеси и сървъри (хостове). Всяка услуга се изпълнява в отделен процес като контейнер, разположен в клъстер от виртуални машини. Приложенията взаимодействат помежду си с помощта на протоколи за комуникация TCP, HTTP и AMQP в зависимост от естеството на работа. Разгърната в облачната платформа Azure, инфраструктурата се управлява от инструмент за оркестрация Kubernetes.

Целта на реферата е да представи информационното взаимодействие между подсистемите за управление, които осъществяват връзка чрез технологии за изпращане и получаване на данни.

Основни задачи, които са поставени:

- да се разгледат актуалните комуникационни модели, които интегрират отделните части на софтуерният продукт;

- да се предложат основни принципи и добри практики при изграждането на облачно базираните приложения;

Основната теза на изследването е свързана с внедряването на бизнес процеси „от край до край“, като същевременно се поддържа последователност и съгласуваност в компонентите на системата. Архитектурата трябва да поддържа важни нефункционални изисквания като: висока степен на достъпност , разширяване на мащаба при увеличаващ се потребителски трафик и други.

Клиентски и сървърни приложения могат да използват различни видове комуникация, насочени към постигането на различни цели. Може да разграничим два основни типа, които се използват между компонентите на системата: синхронна и асинхронна.

### 2.4.1. Синхронна комуникация

Уеб услугите са интерфейси, които са предназначени за комуникация между приложения, за разлика от уеб сайтовете, които са насочени към взаимодействието с хората и се достъпват през браузър (Бенджамин, 2012). Клиенти на уеб услуга могат да бъдат както мобилни или базирани в потребителския браузър приложения, така и други уеб услуги.

В синхрония подход клиентът изпраща HTTP заявка към услуга, която я обработва и връща обратно HTTP отговор. Клиентският код може да продължи своята задача само след получаване на отговор. Заявка и отговор имат обща структура:

- Начален ред, описващ текущия HTTP метод, адрес, статус и протокол;

- Заглавни редове (headers) – дават възможност на клиента и сървъра да предадат допълнителна информация;

- Метаданни за двата HTTP компонента;

- Тяло, съдържащо данни, свързани със заявката или отговора;

Фигура 1.1 илюстрира примерна HTTP заявка/отговор.



Фигура 1.1. Примерна HTTP заявка/отговор

Източник: Бенджамин Е, 2012

#### 2.4.1.1. Механизъм за трансфер на репрезентативно състояние

Representational State Transfer (REST) представлява софтуерна архитектура за проектиране на уеб услуги. Представена е през 2000г. като част от дисертацията на Рой Т. Филдинг. Продуктите, използващи REST, са базирани на хипермедия. REST е независим от протоколите на приложния слой, като концептуалните идеи зад него са взети от HTTP и WWW (Прайс, 2022).

Основно предимство на REST е, че той използва отворени стандарти и не обвързва внедряването на API или клиентските приложения с конкретна реализация.

REST API са проектирани около ресурси/бизнес обекти. В системата за управление това са потребители и поръчки. Създаването на поръчка може да се постигне чрез изпращане на HTTP POST заявка, която съдържа определена информацията. HTTP отговорът показва дали поръчката е създадена успешно или не.

REST е архитектура за моделиране на обекти и операции, които приложението изпълнява. Ресурсите често се групират в колекции. Колекцията е отделен ресурс и притежава собствен идентификатор (Petersen, 2022). Добра практика е URI да се организират в йерархия, като например: https://manager.com/orders връща колекцията от поръчки. Всеки елемент в колекцията има свой собствен уникален идентификатор, като например този за конкретна клиентска поръчка може да бъде https://manager.com/orders/eu.123123.231. Друг примерен идентификатор: https://manager.com/orders/eu.123123.231/deliveries представя доставки за поръчка. Важно да отбележим, че това ниво на сложност може да бъде трудно за поддържане, ако връзките между ресурсите се променят в бъдеще.

HTTP протоколът дефинира редица методи, показани в таблица 1.1, които осигуряват различна семантика, когато се прилагат към ресурс:

***Таблица 1.1****: Методи на протокола HTTP 1.1.*

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| **GET** | Най-разпространеният метод. Използва се за извличане на репрезентации на ресурси. Информацията се съдържа в отговора. |
| **HEAD** | Подобен на GET, но без обект в отговора. |
| **PUT** | Заменя ресурса в посочения URI. Тялото на заявката описва ресурса, който трябва да бъде актуализиран. |
| **DELETE** | Премахва ресурса в посочения URI. |
| **POST** | Създава нов ресурс. Тялото на заявката предоставя подробности за новия ресурс. |
| **OPTIONS** | Предоставя мета данни за ресурс. |
| **PATCH** | Извършва частична актуализация на ресурс. |

Резултат на конкретна заявка зависи от това дали ресурсът е колекция или отделен елемент. Следващата таблица описва общите конвенции, приети от повечето RESTful реализации, използвайки примера за управление на поръчки.

The following table displays the API actions:

|  |  |
| --- | --- |
| **Functionality** | **Path** |
| Get all the updates for a specific house’s devices for a given time range | GET  /api/house/*houseId*/devices/updates?from=*from*&to=*to* |
| Get the updates for a specific device for a given time range | GET /api/device/*deviceId*/updates?from=*from*&to=*to* |
| Get the current status of all the devices in a specific house | GET /api/house/*houseId*/devices/status/current |
| Get the current status of a specific device | GET /api/device/*deviceId*/status/current |

***Таблица 1.2***: *Общи REST конвенции.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ресурс** | **GET** | **POST** | **PUT** | **DELETE** |
| **/orders** | Извлича всички поръчки. | Създава нова поръчка. | Общо актуализиране на поръчките. | Премахва всички поръчки. |
| **/orders/1** | Извлича детайли за поръчка 1. |  | Актуализира данните за поръчка 1, ако съществува. | Премахва поръчка 1. |
| **/orders/1/ deliveries** | Извлича доставките за поръчка 1. | Създава нова доставка за поръчка 1. | Общо актуализиране на доставките за поръчка 1. | Премахва всички доставки за поръчка 1. |

В HTTP протокола форматите се определят чрез използване на типове медии, наричани още MIME. За недвоични данни, повечето уеб API поддържат JSON (application/json) или XML (application/xml) като формат за обмен. Те се използват за представяне на структурирани данни. Например, заявка към посочения по-горе URI за детайли на поръчкa, ще върне следния отговор във формат JSON:

{

"orderId":eu.123123.231,

"orderValue":99.90,

"productId":1

}

Сървърът информира клиента за резултата от заявка чрез използване на предварително зададени “кодове на състоянието”, представени в следната таблица.

***Таблица 1.3****. Таблица с диапазоните на HTTP кодовете.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Статус код** | **Тип** | **Описание** | **Пример** |
| **1xx** | Информационен | Изпраща се с подготвителна цел. | 100 Continue |
| **2xx** | Успех | Заявката обработена успешно. | 200 OK |
| **3xx** | Пренасочване | Клиентът трябва да изпрати  допълнителни заявка. | 301 Redirect |
| **4xx** | Грешки в клиента | Резултат от грешна заявка, причинена от клиента. | 404 Not Found |
| **5xx** | Грешка в сървъра | Грешка от страна на сървъра. | 503 Service Unavailable |

REST е подходящ за CRUD-базирани операции. Клиентите взаимодействат със сървъра през HTTP. REST е широко разпространен и е поддържан от повечето работни рамки като ASP.NET, Symfony, Spring, Node.js и други.

#### 2.4.1.2. Механизъм за заявки към отдалечени процедури

gRPC е високопроизводителна рамка, която позволява дистанционно извикване на процедури (RPC). На ниво приложение, gRPC структурира съобщенията между клиенти и бек-енд услуги. Произхождащ от Google, това е проект с отворен код и част от Cloud Native Computing Foundation.

Kлиентско gRPC приложение създава локална функция в уеб услуга, която реализира бизнес операция. Тази локална функция извиква друга функция на отдалечена машина (Vettor, 2022).

В приложенията, базирани на облак, разработчиците често работят на различни езици за програмиране, рамки и технологии. gRPC осигурява

„хоризонтален слой“, който помага за съвместимостта между компонентите.

gRPC използва HTTP/2 като транспортен протокол, който разполага с разширени възможности (Vettor, 2022):

- Двоичен протокол за транспортиране на данни, за разлика от HTTP1.1, който е текстов;

- Поддържа изпращане на множество паралелни заявки през една и съща връзка;

- Компресира съдържанието на съобщенията, което намалява натоварването на мрежата;

Механизмът е базиран на технология с отворен код, наречена Protocol Buffers. Файловете .proto осигуряват висока ефективност и платформено-неутрален формат за структуриране на съобщения. Използвайки междуплатформен език за дефиниране на интерфейс (IDL), разработчиците дефинират “договор” за всяка микроуслуга. Договорът, реализиран като текстов .proto файл, описва методи, входове и изходи. Използвайки прото файла, компилаторът може да генерира както клиентски, така и сървърен код за целевата платформа.

Кодът включва следните компоненти:

• Строго обособени обекти, споделени от клиента и услугата, които представляват елементи от съобщението;

• Базов клас, който отдалечената gRPC услуга може да наследява;

Може да разгледаме следния пример за order\_delivery.proto.

syntax = "proto3"; // версия на синтаксиса

option csharp\_namespace = "Manager.Protos";

package order\_delivery; // идентификатор на пакета

import "google/protobuf/wrappers.proto";

service OrdersDeliveries {

rpc GetOrder (GetOrderRequest) returns (NullableOrder); // метод за извикване

}

message GetOrderRequest { // формат на съобщението

google.protobuf.StringValue order\_nr = 1;

}

message Order { // формат на отговора

google.protobuf.StringValue order\_nr = 1;

}

**Фигура 1.2.** Protobuf файл за интегриране на микроуслугата за поръчки.

*Източник: Разработка на автора*

По време на изпълнение всяко съобщение се сериализира като стандартно представяне на Protobuf и се обменя между клиента и отдалечената услуга. Единицата за обем на информация се сериализират в двоичен вид (Smith, 2022).

### 2.4.2. Асинхронна комуникация

Съществен момент при изграждането на ориентираната към услуги архитектура е интеграцията помежду им. Комуникацията между подсистемите трябва да бъде сведена до минимум. Целта на всеки микросървис е да той бъде автономен и достъпен за потребителя, дори ако другите, които са част от приложението, не работят. Силна зависимост в архитектурата може да опишем в случай, когато трябва да се извърши повикване от една микроуслуга към друга (като изпълнение на HTTP заявка за получаване на данни), за да може да се обработи отговор. Програмният продукт няма да бъде устойчив ако някоя от частите се срине. Освен това, създаването на вериги от заявки/отговори, намалява значително производителността на цялото приложение.

#### 2.1 Базирана на съобщения комуникация

Асинхронните съобщения и управляваната от събития комуникация са от критично значение при разпространението на промените в множество микроуслуги и свързаните с тях домейн модели. Когато настъпят промени, системата се нуждае от начин за съгласуване в различните модели. Решението: евентуална последователност и комуникация, управлявана от събития, базирана на асинхронни съобщения.

Клиент прави заявка към услуга, като й изпраща съобщение. Тъй като това е асинхронна комуникация, клиентът приема, че отговорът няма да бъде получен веднага или че няма да има отговор. Съобщението се състои от заглавие (метаданни като информация за идентификация) и тяло. Съобщенията се изпращат чрез асинхронни протоколи като AMQP. Част от предпочитаната инфраструктура за този тип е брокер на съобщения, който е различен от тези, използвани в SOA. В опростен вариант, той действа като шина.

Налични са различни брокери на съобщения като всеки един има низ за връзка и потребителски достъп. Реалната част на процеса се състои в крайните точки, които публикуват и получават съобщения, тоест в микроуслугите. Важно правило, което системата за управление се опитва да спазва, доколкото е възможно, е да използва само асинхронни съобщения между вътрешните услуги, а синхронна комуникация: само от клиентските приложения към API Gateway.

Когато се използва асинхронна комуникация, управлявана от събития, микроуслуга публикува интеграционно съобщение, задействащо се когато нещо се случи в нейния домейн и друга микроуслуга трябва да получи информация за това. Пример е промяна на цената в микроуслуга от продуктов каталог. Микроуслугите се абонират за събитията, за да могат да ги получават асинхронно. Когато това се случи, получателите могат да актуализират собствените си обекти. Тази система за публикуване/абониране се реализира чрез по-горе споменатия брокер.

#### 2.2 Съгласуваност между услугите

Високата степен на наличност и толерантност към частични проблеми не могат да бъдат гарантирани на 100% в разпределените системи. От гледна точка на широкомащабните уеб архитектури, това е от важно значение за съхранението в услугите, тъй като тези компоненти запазват текущите състояния на приложението.

Условието при внедряване на бизнес процеси от край до край, е същевременно да се поддържа последователност в услугите.

Важни изисквания за последователността са:

- Нито една услуга не трябва да включва таблици/хранилища за данни от друга и не трябва да извиква директни заявки към тях;

- Предизвикателство при възникване на частична повреда – колкото по-свързани са частите на системата, толкова по-голям проблем.

### 2.4.3.3. Комуникационни модели за достъп до бекенда

За да бъдат нещата опростени, клиент от „предния край“ може да комуникира директно с микроуслугите. Следната фигура показва този вариант. Този подход се използва, когато различни части от страницата на клиента изискват различни микроуслуги. Всяка микроуслуга има публична крайна точка, която е достъпна от клиентските приложения. Макар и лесна за изпълнение, директната комуникация с клиента би била приемлива само за прости микросервизни приложения. Този модел свързва тясно клиентите от предния край с основните бек-енд услуги, което води до редица проблеми, включително:

- Микроуслугите трябва да бъдат изложени на „външния свят“;

- Междусекторни проблеми като удостоверяване и оторизация;

- Сложен клиентски код - клиентите трябва да следят множество крайни точки и да се справят с възможни неуспехи;

Като надграждане на първия модел за проектиране, облачната инфраструктура позволява да се внедри API Gateway. Тя предоставя единична точка за група микроуслуги. Наподобява модела за дизайн: „фасадата“. Известен е също като „backend for frontend“. Изгражда се за конкретните нужди на клиента. Действа като пълномощник между клиентите и микроуслугите. Може да осигури удостоверяване, кеширане или други. Azure предоставя няколко готови продукта:

- Azure Application Gateway - насочен към .NET, работещ с архитектура, ориентирана към микро услуги, предоставящ унифицирана входна точка към системата. Услугата поддържа възможности за балансиране на натоварването;

- Azure API Management - шлюз, който позволява контролиран достъп до бек-енд услуги, базиран на конфигурируеми правила. Предоставя уеб портал на разработчиците, които могат да го използват за инспектиране на услугите и анализиране на тяхното натоварване.

Шаблонът е показан на следната фигура:



**Фигура 2.2**. Комуникация чрез шлюз за приложните програмни интерфейси.

*Източник: Smith, S., 2022*

API шлюзът може да се превърне в “анти-модел“ като монолитно приложение, съдържащо твърде много крайни точки, обединяващо всички микроуслуги. API шлюзовете трябва да бъдат разделени от логически групи въз основа на бизнес ограничения. Протоколи за пренос на данни могат да бъдат HTTP или gRPC.