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

## 2.1. Същност, цел и обхват на софтуерната архитектура

Думата „**архитектура**“ често се използва в контекста на нещо от високо ниво, което е отделено от детайлите на по-ниско ниво (Martin et al., 2017). Софтуерният продукт, разглеждан в настоящия труд, ще се състои от 2 клиентски приложения, които ще 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.



Фиг. 2.2. Диаграма от високо ниво на главните приложения (разработка на автора)

Крайните потребители ще управляват и проследяват поръчките и доставките в реално време с мобилно приложение. Целта му е да помага с планирането и логистиката, въздействие върху крайния резултат с информация и данни. Информацията на смартфона трябва винаги да е актуална, тъй като текущото състояние на поръчка и местоположението на доставките се проследяват на живо. Също така ще има и възможност за преглед на историята на доставките. Приложението ще се предлага безплатно чрез Google Play Store и Apple App Store.

ТОДО+

### 2.1.1. **Ключови бизнес процеси и дейности при системата за** управление на поръчките

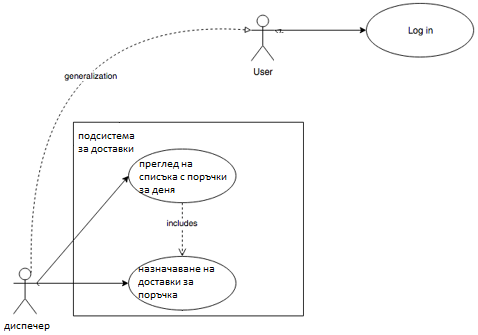
Тази подточка ще представи важни случаи на употреба, които са критични за бизнеса, част от основния домейн. Ще бъдат използвани диаграми на бизнес сценариите. Те ще идентифицират действия, които очакваме потребителите да направят.

Най-подходящ за взаимодействие с крайните потребители са мобилните приложения (). Също така поддържат функции като местоположение и камера и работят с уеб API. Обхватът на мобилното приложение включва екран за вход както за клиент, така и за шофьор, интерфейс за текущите поръчки и доставки към тях. Също така панел за създаване или промяна на поръчка. Фигура 2.1 представя процесите под формата на диаграма:



Фиг. 2.1. Диаграма на главен бизнес сценарий (разработка на автора)

- Обхват на уеб портала



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

### 2.1.2. Функционални и нефункционални изисквания към системата

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. Някои от основните изисквания са:

• Регистриране на акаунт;

• Вписване на потребител;

• Отписване на потребител;

• Преглеждане на текущите поръчки;

• Разглеждане на детайлите за определена поръчка и доставки;

• Филтриране на елементите по дата;

• Добавяне на нова поръчка;

• Промяна или премахване на съществуваща, неактивна поръчка;

Нефункционалните изисквания описват как една система трябва да се държи и установяват ограничения на нейната функционалност. Този тип изисквания са известни още като атрибути за качество като:

• Трябва да е високо-достъпно 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 за използване мониторинг на състоянието на системните единици и диагностични дневници, за да помогне при отстраняване на неизправности или други проблеми, които 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 уеб интерфейс;

• Трябва да поддържа междуплатформен хостинг и развитие;

## 2.2. Концептуален модел на системата

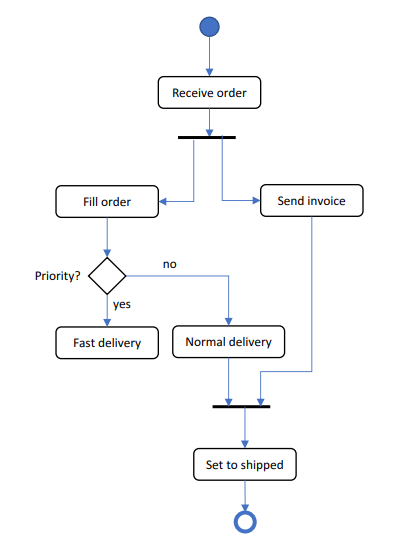
Концептуалните модели са абстрактни представяния за това как трябва да протича изпълнението на задачите. Те представят визуално концепция или операция.

### 2.2.1. Поведенчески диаграми

Поведенческите диаграми идентифицират как различните елементи взаимодействат помежду си, за разлика от структурните диаграми, които описват блоковете, които изграждат самата системата.

#### 2.2.1.1 Диаграми на дейностите UML

Диаграмите на дейността изглеждат много подобни на блок-схемите. Наличието на тези прилики улеснява комуникацията между технически и не-технически лица. Следната диаграма представя работни потоци и общи операции в мобилното приложение. ТОДО+



Фиг. 2.6. Диаграма на дейностите за мобилно приложение. (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.

Пример може да разгледаме в контекста на подсистемата за поръчки:



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

https://blog.openreplay.com/jwt-authentication-best-practices

### 2.2.2. Структурни диаграми

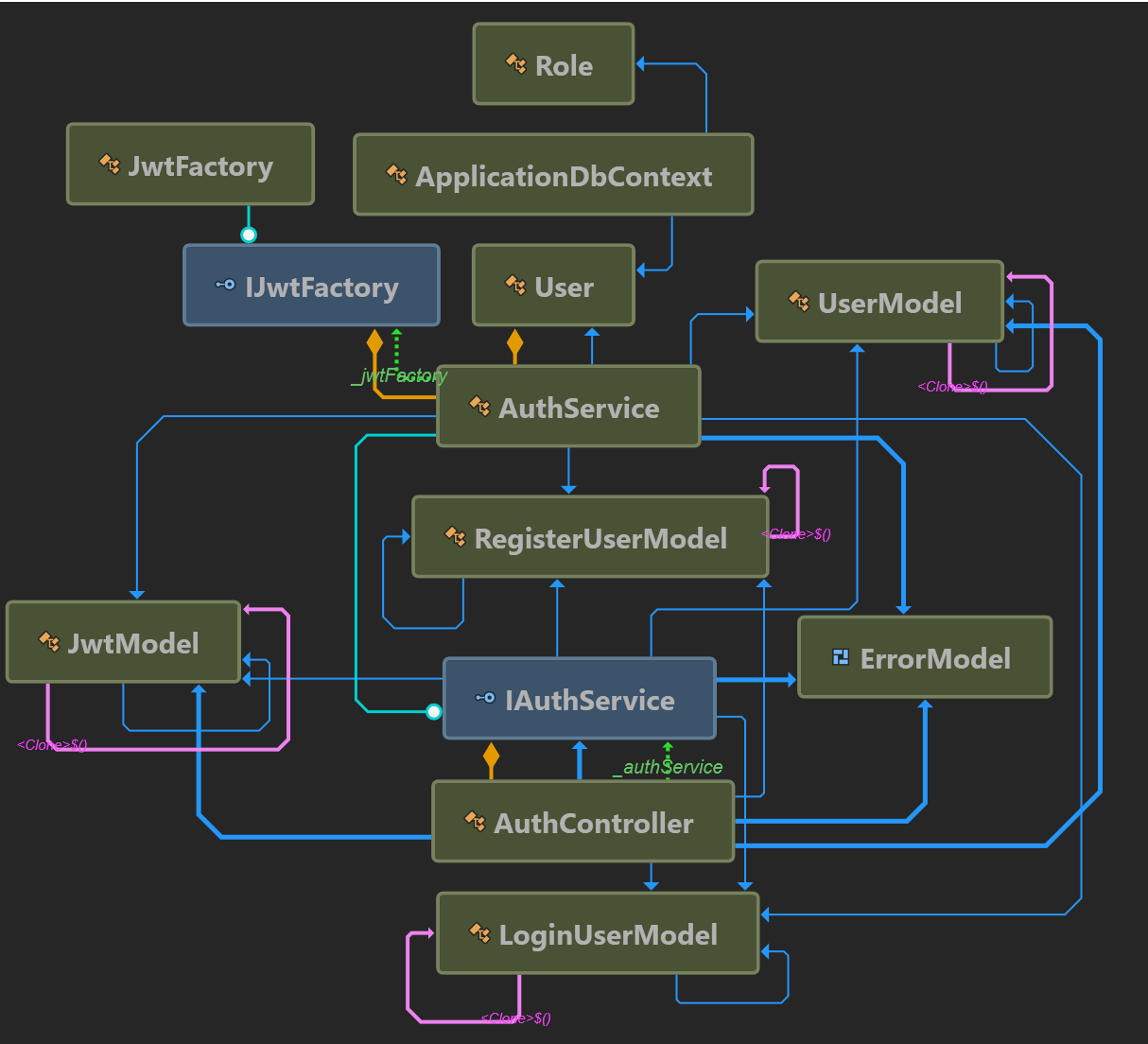
Структурните диаграми помагат за дефиниране цялостната структура системата, подобно на плана, който определя как изглежда една къща. Структурните диаграми моделират как ще изглежда системата в архитектурно отношение. Те ни помагат да дефинираме „речника“ на системата, гарантират съгласуваност от заинтересовани 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 ще 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 за БСВС „Разработка на продукт / Планиране на проект”. (разработка на автора)