

**ФАКУЛТЕТ ЗА ИНФОРМАТИКА**

**Tема:**

**” РАЗВОЈ НА ВЕБ АПЛИКАЦИЈА ЗА ПРОДАВАЊЕ НА СУПЛЕМЕНТИ ВО ИСХРАНА”**

**Дипломска работа**

Ментор: Кандидат:

проф. др. Ѓорѓи Какашевски Леон Чобанов (15333)

Јуни, 2021 година

Скопје

# Апстракт

Оваа дипломска работа го опфаќа развојот на мојата веб-апликација за продавање на суплементи во исхрана. Пред се таа претставува веб продавница каде што корисникот може да си направи свој акаунт и да набави производ по негов избор. Предноста на оваа апликација се состои во тоа што купувачот ќе може да си ги комбинира производите кои се неопходни во подобрување на неговата физичка кондиција. Исто така во зависност од можностите производите ќе бидат понудени со пониска маржа со цел да бидат поконкурентни во однос на физичките пунктови каде што физички се продаваат истите производи. Понатаму исто така како општествено одговорна апликација ќе им бидат понудени на клиентите низа на објаснувања за предностите но и за негативните страни од прекумерна консумирање на тие производи. Таргетираната група на оваа апликација се спортисти кои се занимаваат со фитнес и бодибилдинг но може да се прошири на сите кои се занимаваат со спортски активности.

**Содржина**

[Апстракт 2](#_Toc76717462)

[**Содржина** 3](#_Toc76717463)

[Интродукција 5](#_Toc76717464)

[1 Корисничко сценарио(Use Case) 5](#_Toc76717465)

[- Авторизација и автентикација 7](#_Toc76717466)

[- Кошничка 10](#_Toc76717467)

[**-** Купување на продукти 11](#_Toc76717468)

[Административен панел: 12](#_Toc76717469)

[2. Моделирање и имплементирање на базата на податоци. 15](#_Toc76717470)

[Генерален податочен модел за креирање на логичкиот модел на оваа веб апликација. 15](#_Toc76717471)

[ДФД диаграм 16](#_Toc76717472)

[3. Креирање на REST API 17](#_Toc76717473)

[3.1 Authentication 20](#_Toc76717474)

[3.2 Слоева архитектура. 24](#_Toc76717475)

[3.3 Kiss and dry 27](#_Toc76717476)

[4. Frontend 28](#_Toc76717477)

[5. Stripe 33](#_Toc76717478)

[6. Elastic search 35](#_Toc76717479)

[Заклучок 38](#_Toc76717480)

[Литература 39](#_Toc76717481)



# Интродукција

За практичната реализација на оваа веб апликација се искористени следните технологии:

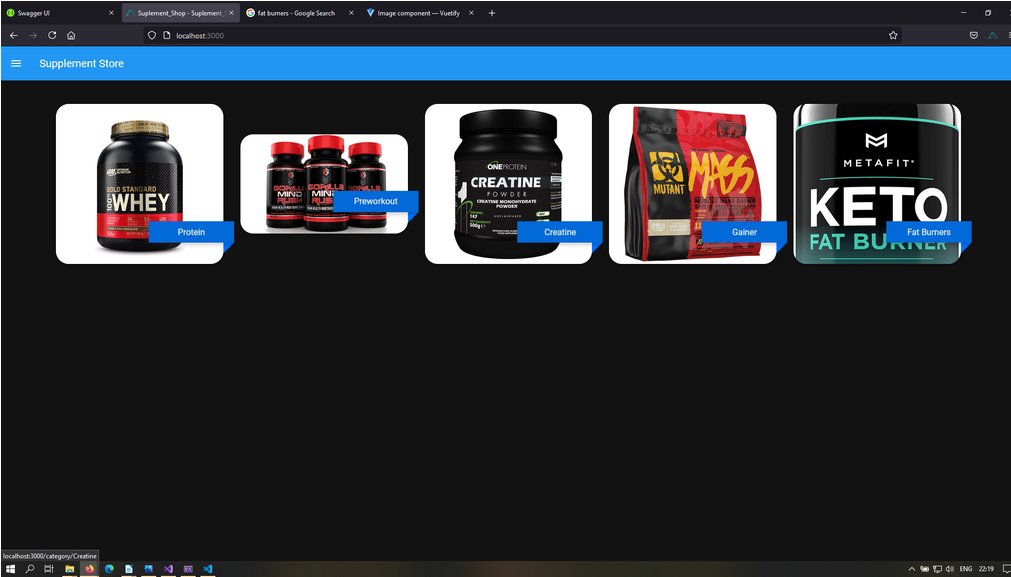
* Microsoft ASP.NET Core - веб фрамеворк
* C# - Програмски јазик за backend
* EntityFramework - Persistence
* Vue JS – Фронтенд java script базиран framework
* Nuxt – веб апликациски framework базиран на Vue.js
* Stripe – Процесирање на наплата
* *Ms*SQL- Датабаза

Додека за целосна реализација на овој проект пред се треба да се отвори мала компанија која би била регистрирана со некои почетен влог од пет илјади евра.

# 1 Корисничко сценарио(Use Case)

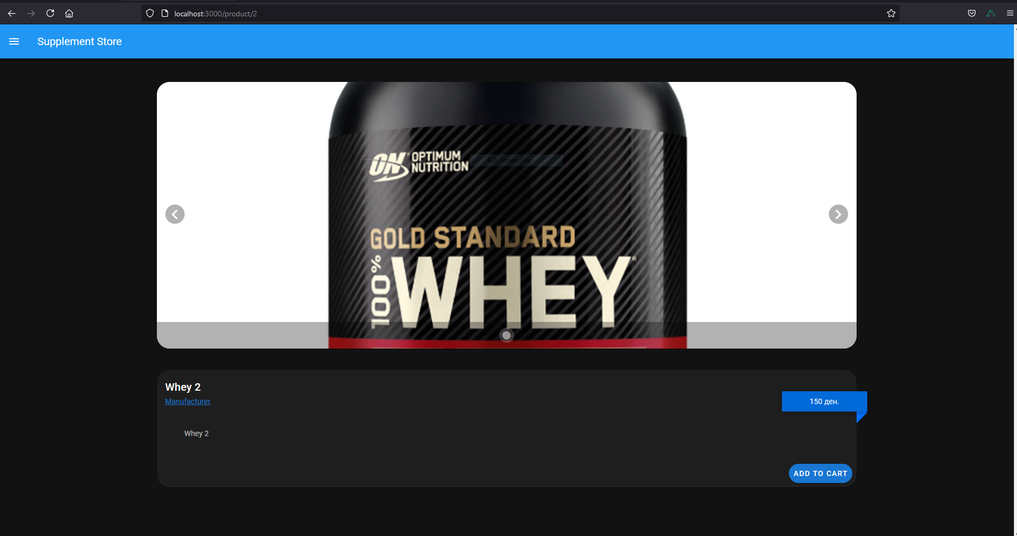
**- Разгледување на продавницата**

Корисникот може да ја разгледа продавницата.Навигира низ категориите,да кликне на производите и да види дискрипција за тој производ.Во прегледот на деталите, тој може да ги види акциите и цената на тој производ.



**Слика : Генералниот изглед на веб продавницата.**

За време на процесот на дизајнирање на оваа веб страна ставено е големо нагласување на симплистички дизајн.Темната позадина е избрана за да му олесни на корисникоти тој да моње лесно да навигира низ продуктите без непотребен замор на очите додека продуктот е со бела позадина која создава цврст контраст.

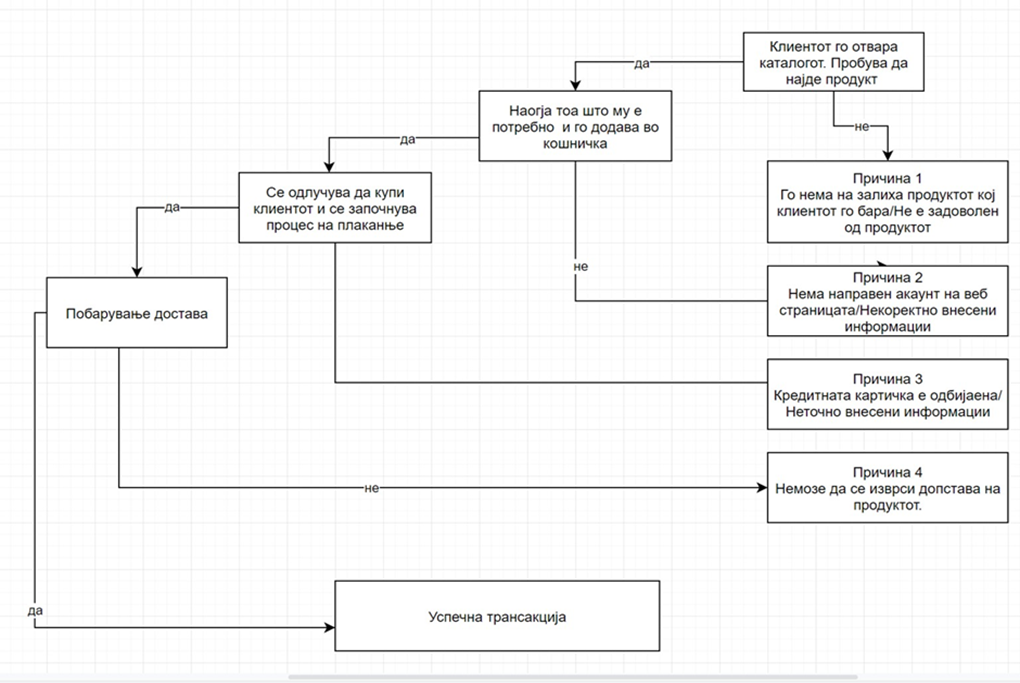


**Слика : Продукти во продавницата.**

Како што можеме да забележиме од горната слика сите информации за продуктот се листирани заедно со можноста да се додаде во кошничка.

**-Процес на одлуки**

Процес за одлука се ефикасен начин за графички приказ на голем број опции и обезбедуваат механизам за испитување на можните исходи и придобивки од изборот на тие опции. Тие исто така можат да му помогнат на аналитичарот да формира избалансирана слика за ризиците и придобивките поврзани со секој можен тек на дејствување. Тие се близок братучед на табелата за одлуки, но имаат придобивка дека се графички.Во нашиот случај овој диаграм ни е потребен за да моњеме детално да предвидиме секој чекор на типичниот корисник.



**Слика: УМЛ диаграм за процес на одлуки.**

## - Авторизација и автентикација

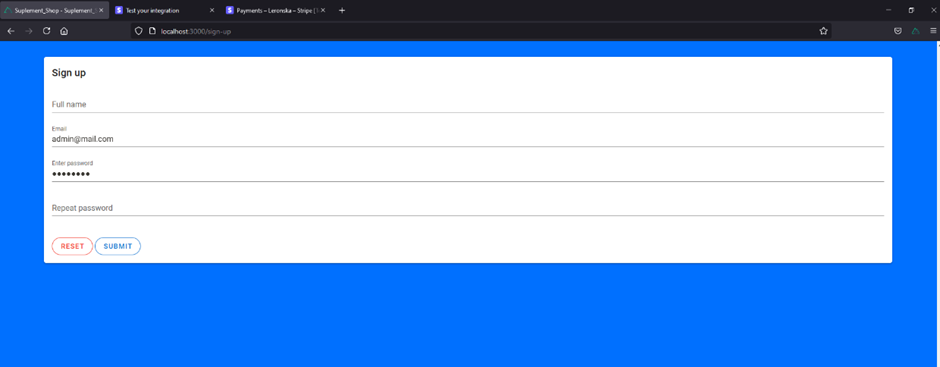
Автентикацијата е процес на проверка на идентитетот. Уникатен идентификатор е поврзан со корисник што е корисничко име или корисничка идентификација. Традиционално, ние користиме комбинација на корисничко име и лозинка за автентикација на корисник. Логиката за проверка треба да се одржува локално, така што ќе ја наречеме локална автентикација.

Најчестата техника за проверка е користење на корисничко име и лозинка

при неговото спроведување:

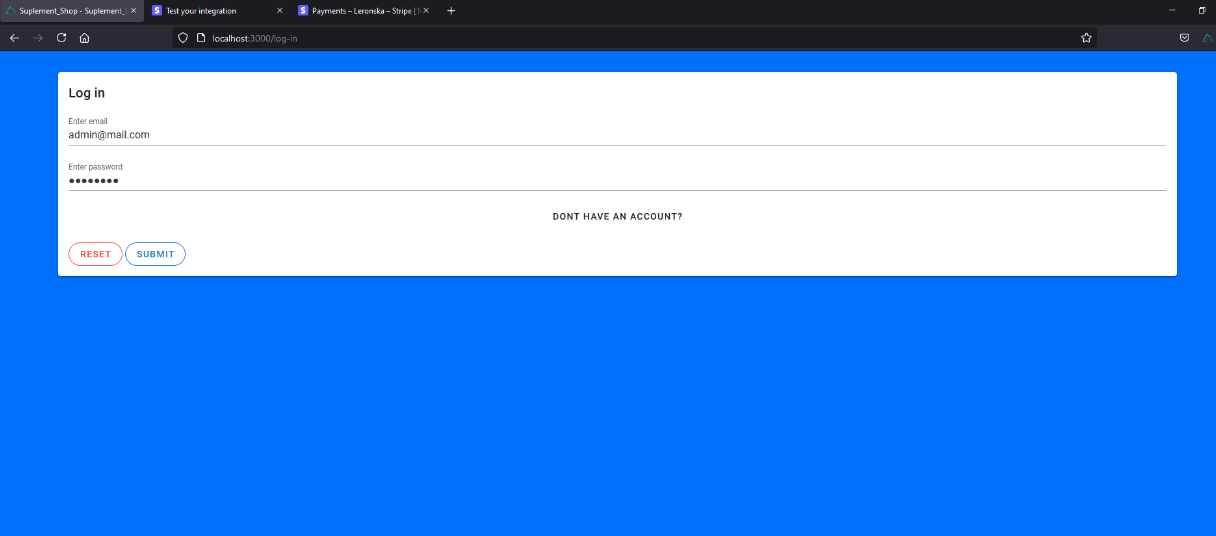
* Корисникот се регистрира користејќи идентификатор како корисничко име / е-пошта / мобилен;
* Апликацијата ги зачувува корисничките квалификации во базата на податоци;
* Апликацијата испраќа е-пошта / порака за верификација за да ја потврди регистрацијата;
* Објави успешна регистрација, корисникот ги внесува ингеренциите за најавување;
* При успешна автентикација, на корисникот му е дозволен пристап до специфични ресурси;
* Корисничката состојба се одржува преку Сесии или JWT.

Креирањето на акунт е истото така лесен процес кој бара само име ,емајл адреса и пасворд од корисникот.



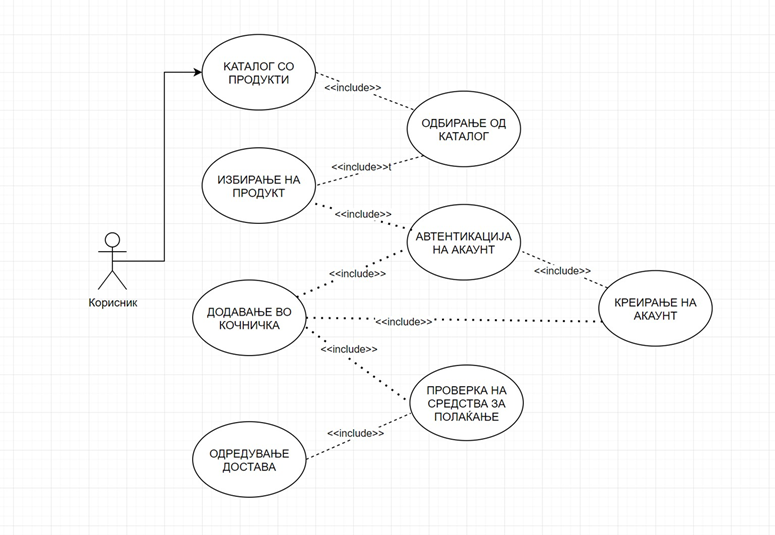
**Слика : Креирање на нов акаунт.**

Од како ќе се верификуваат информациите дадени од страна на корисникот тој треба само да се најавува во својот акаунт.

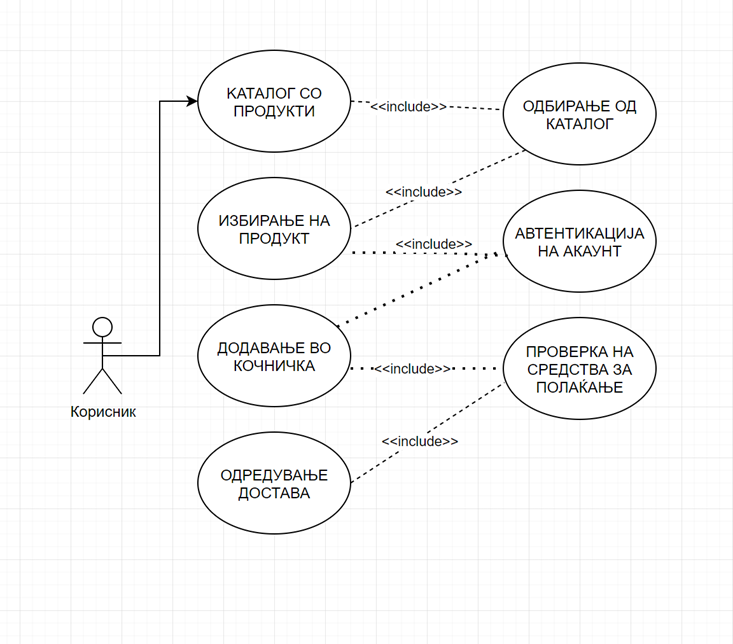
****

**Слика : Логирање во акаунт.**

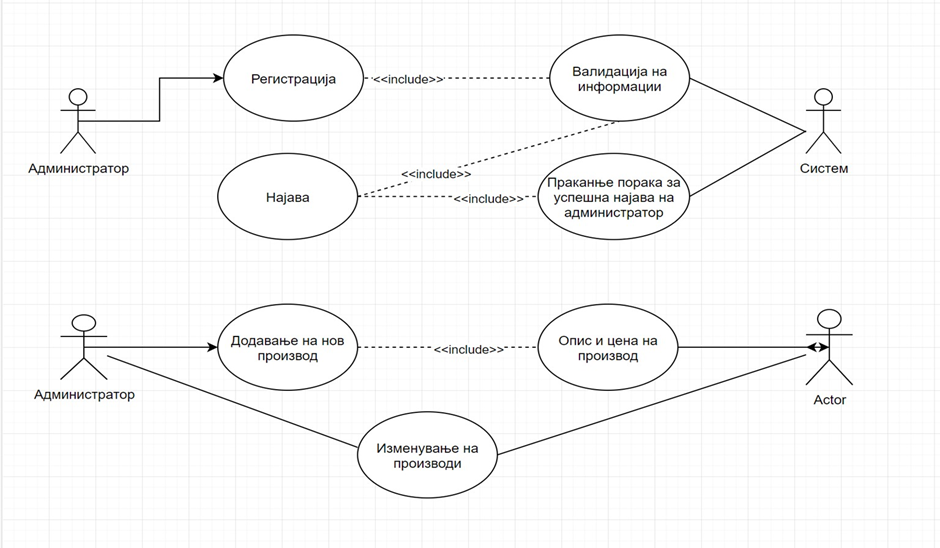
Корисничко сценарио (Use Case) диаграм претставуваат дејствија што се потребни за да се овозможи или да се напушти целта. Случај за употреба има повеќе „патеки“ што може да ги преземе секој корисник во исто време.Сценариото за употреба е единствена патека низ случајот за употреба.



**Слика Use Case диаграм за не најавен корисник.**



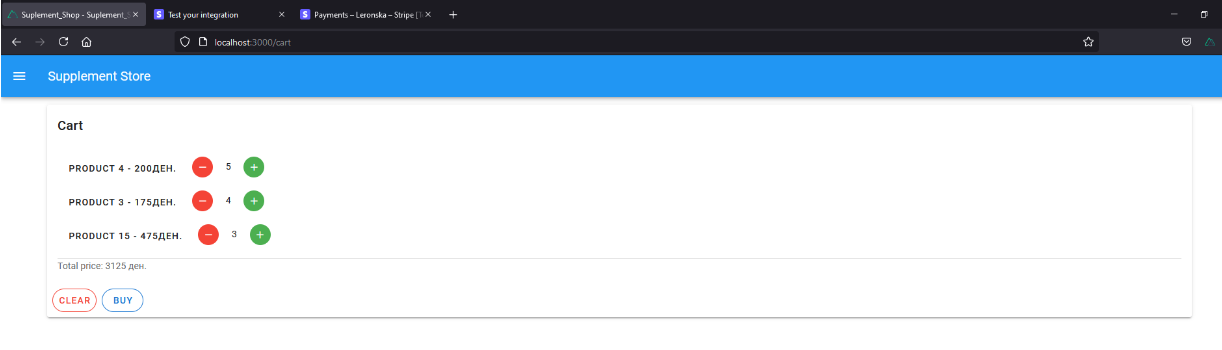
**Слика Корисничко сценарио за најавен корисник.**



**Слика: Корисничко сценарио за управител на веб продавница**

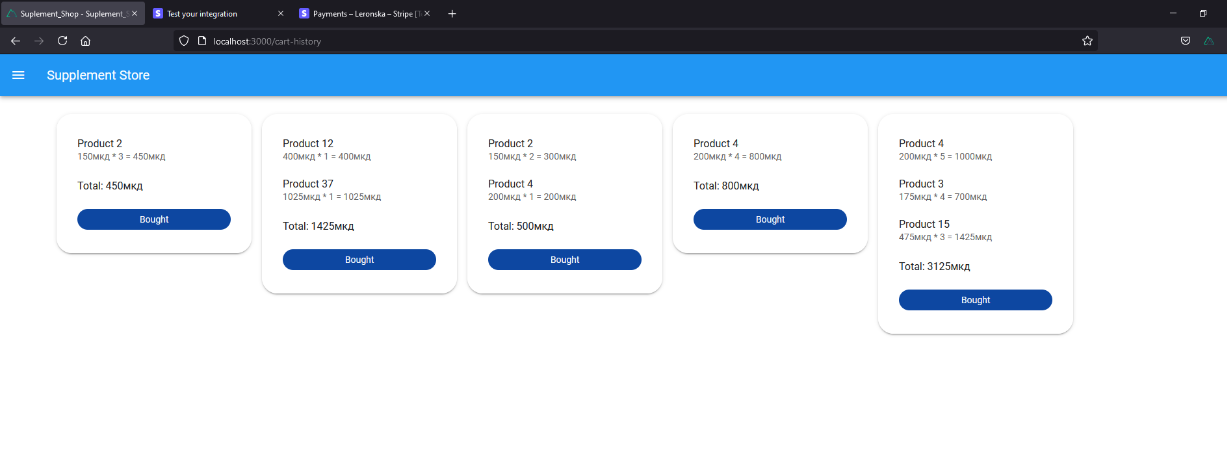
## - Кошничка

Кошничка е дел од софтвер кој одржува евиденција за производите што ги купува еден купувач од онлајн продавницата. Тоа е работа на каталог на продавници што им помага на купувачите на Интернет во изборот на производи што сакаат да ги купат, преглед на нивните избрани производи, модифицирање на нивните избори, додавање или отстранување производи и купување на производи. Кошничка за трговија делуваат како мост помеѓу купувањето преку интернет и купувањето. Важноста на количката во онлајн продавница не е ни дискутабилна тема. Постојат бесконечни причини за тоа. Леснотијата на користење во мрежните објекти за купување дејствува како двоглав меч; со лесни процеси, клиентите можат брзо да купуваат производи, како и да ги напуштат.



**Слика: Кошничката на нашата веб продавница.**

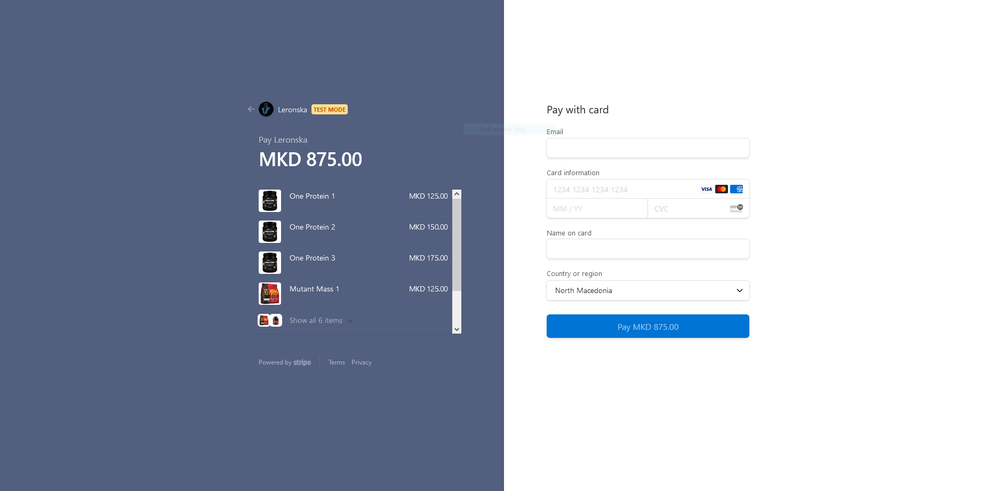
Веб апликацијата исто така има секција која се вика cart history и тука корисникот може да си ја провери историјата на кошничката т.е. кои продукти ги купил во минатото.



**Слика : Историја на кошничка.**

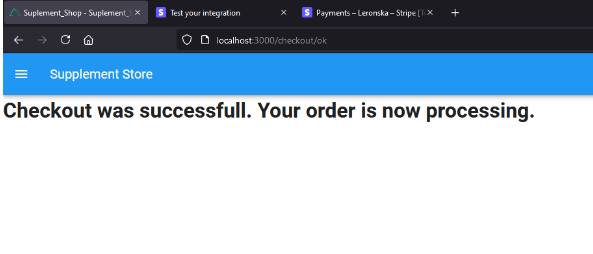
## **-** Купување на продукти

Финалниот чекор кон успешна трансакција е направен со помош на Stripe. Stripe обезбедува API што веб девелоперите можат да ги користат за интегрирање на процесирање на плаќањето во нивните веб-страници и мобилни апликации.



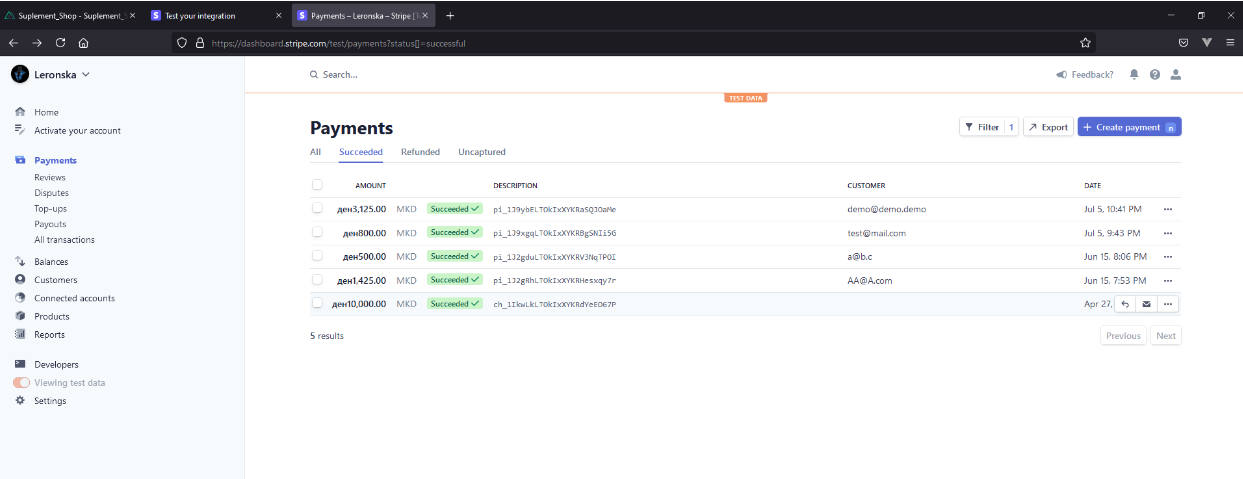
**Слика: Купување на продукрти.**

Потврдна порака која се појавува после секоја успешна трансакција иму дава дознание на корисникот дека се е во ред.



**Слика: Успешна трансакција.**

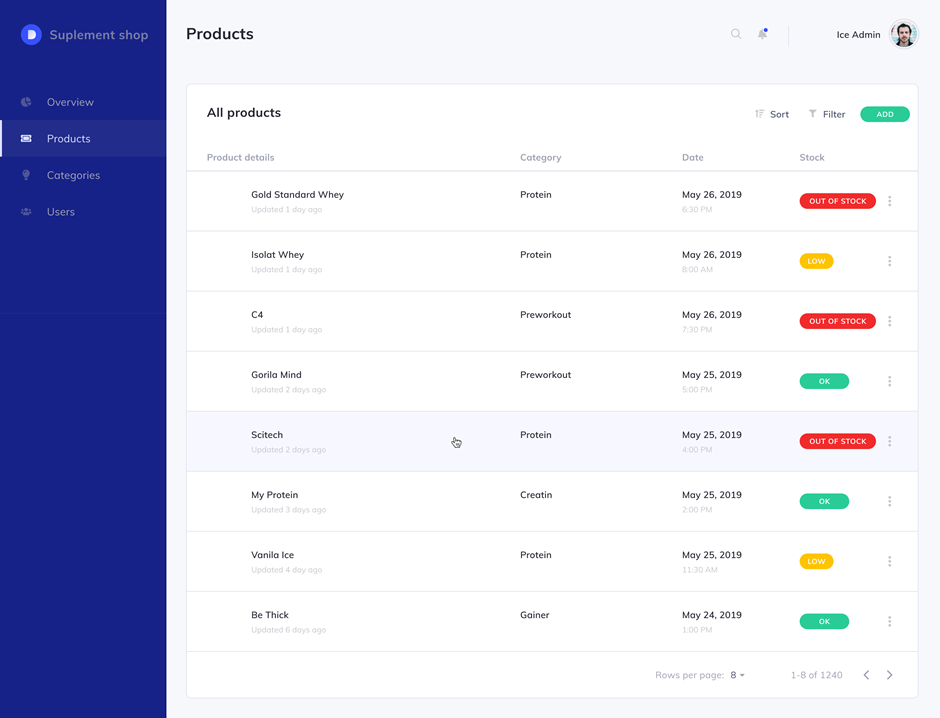
Од другатта стртана на трансакцијата Stripe им овозможува на администраторите да гледаат историја на наплати.

**Слика: Историја на трансакции.**

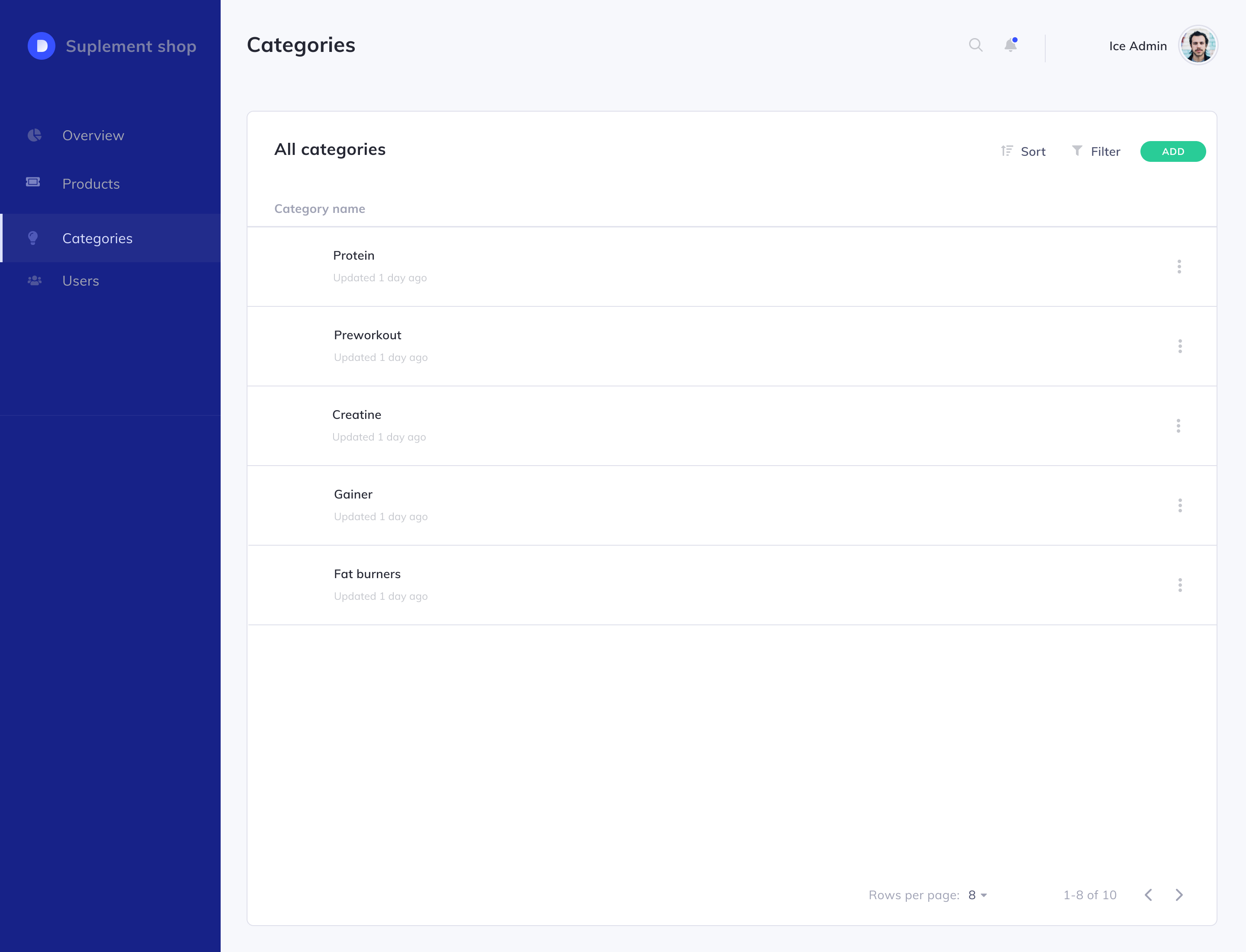
## Административен панел:

Административниот панел (обично најавен од / wp-admin) е местото каде што се создаваат нови објави, категории, ознаки, страници, врски и прилагодени типови на објава. Исто така, тука се менуваат датотеките со теми, се додаваат додатоци, се активираат или ажурираат приклучоци и се менуваат читањето / пишувањето / општите поставки.На кратко, административниот панел е местото каде што се создава содржината и се управува со веб-страницата. Ова е клучот за тоа како функционира системот за управување со содржини (CMS).

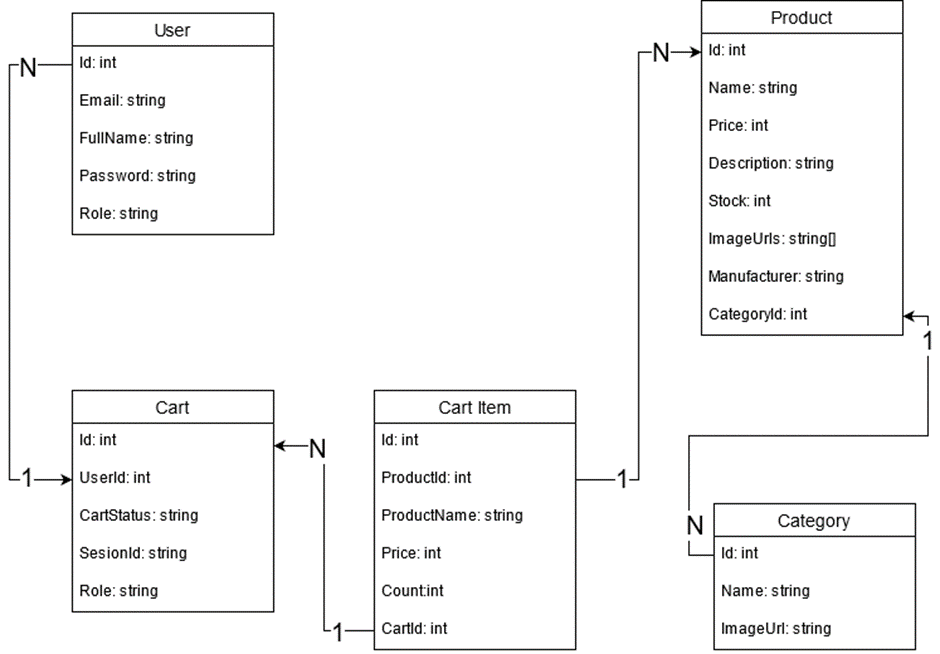
Cистем за управување со содржина, често скратено како CMS, е софтвер кој им помага на корисниците да создаваат, управуваат и модифицираат содржина на веб-страница без потреба од специјализирано техничко знаење.На поедноставен јазик, системот за управување со содржини е алатка која ви помага да градите веб-страница без да имате потреба да го пишувате целиот код од нула (или дури и воопшто да знаете како да кодирате).Наместо да градите ваш сопствен систем за создавање веб-страници, складирање слики и други функции, системот за управување со содржини се справува со сите основни работи за вас, така што ќе можете да се фокусирате на повеќе делови на вашата веб-страница.Надвор од веб-страниците, можете да најдете и системи за управување со содржини за други функции - како управување со документи.



**Слика: Мокап на администраторски панел.**

**Слика: Moкап едитирање на категории**

# 2. Моделирање и имплементирање на базата на податоци.



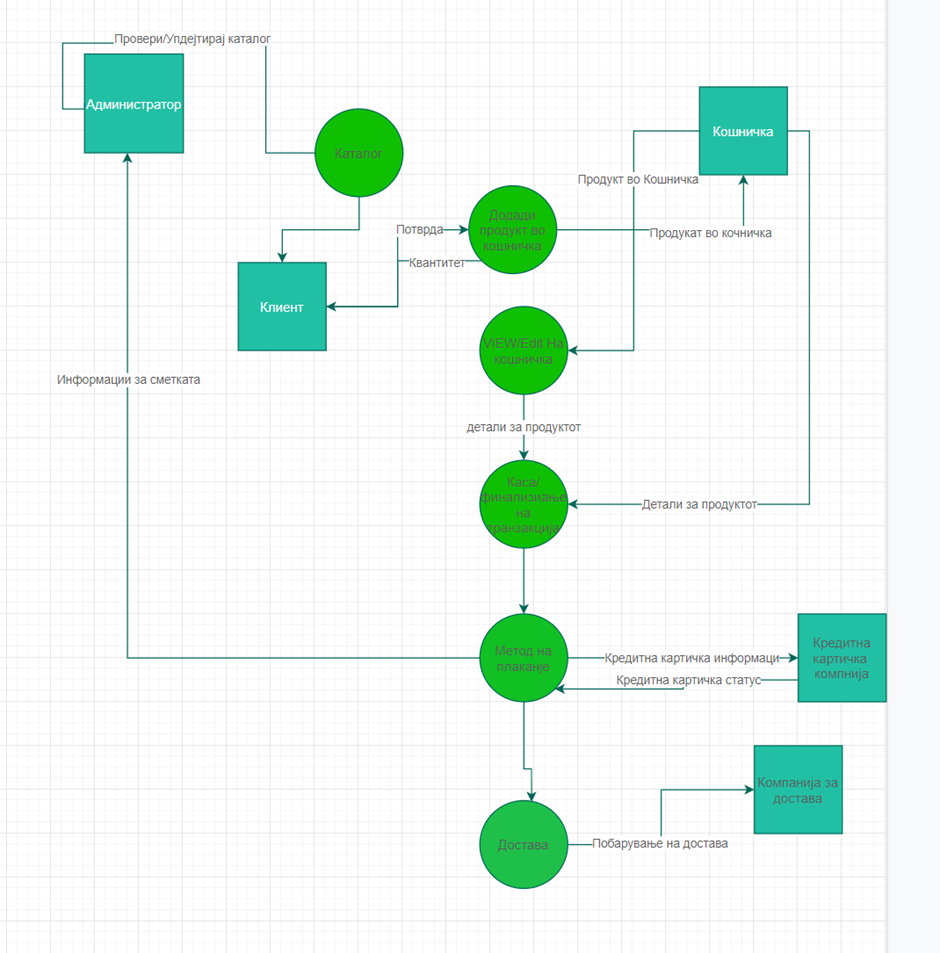
**Слика УМЛ диаграм на датабазата.**

## Генерален податочен модел за креирање на логичкиот модел на оваа веб апликација.

* -Администратор ( Упдејтирање на Каталог, Сметка за секоја успешна трансакција )
* -Клиент (Избор на продукт, персонални информации, промоционални кодови)
* -Кредитна Компанија (иформации за компанијата,статус на кредитната кратичка,Код картичка)-Клиент и Кредитна Компанија (Број на картичка, Код картичка,Статус)
* -Компанија за достава(Адреса на продуктот,Дестинациска адреса,Статус на достав)
* -Клиент и Компанија за достава (Адреса на клиент, Очекувано време на достигнување)

## ДФД диаграм

Дијаграмите за текови на податоци (Data Flow Diagrams – DFD) служат за графичко претставување на текот на податоци што се случуваат во некој апликативен систем. Тие не ја претставуваат контролата на тие процеси. Процесите претставени со DFD можат да се одвиваат паралелно, но и во сосема различно време. Моделот е прилично едноставен затоа што постојат само четири симболи: надворешен ентитет (external entity), процес (process), податочен склад (data store), податочен тек (dataflow).



# 3. Креирање на REST API

REST е акронимот што се залага за: Претставнички државен трансфер. REST е архитектонски стил на дистрибуиран систем. Се заснова на множеството принципи што опишуваат како да се дефинираат и адресираат мрежните ресурси. Овој пакет принципи за првпат беше опишан од „Рој Филдинг“ во 2000 година. РЕСТ е поголем од веб-услугите.

RESTful услугите користат HTTP (протокол за хипер пренос на текст) за комуникација. REST е системски интерфејс со надворешни системи како веб-ресурси идентификувани од URI (униформни идентификатори на ресурси).

* **Microsoft ASP.NET Core - веб фрамеворк**

Како што споменавме погоре, ASP.NET Core е нов, со отворен извор, модуларен, крос-платформа, растеглив, асинхрон и многу послаба рамка. Сега е единствената рамка што работи покрај времето на траење на .NET Core 5 (Core-CLR) или времето на траење .NET Framework (CLR) и има многу предности, од кои повеќето ќе ги разгледаме подолу.

NuGet

За разлика од монолитната .NET рамка, .NET Core платформата е збир на пакети NuGet кои обезбедуваат мало, дискретно парче функционалност. Тоа ви овозможува да ги оптимизирате апликациите, да ги направите полесни и многу слаби. Можете исто така да испратите приватна верзија на .NET Core Framework за вашата специфична апликација без ризик дека другите верзии на апликацијата можат да го променат однесувањето на вашата апликација.

Тоа е клучна вредност на ASP.NET Core, која повеќе не се заснова на System.Web.dll. Може да работи на повеќе верзии на .NET Core на истата машина. NuGet им овозможува на ASP.NET тимот да обезбеди нова функционалност и да се поправат многу полесно и побрзо. На овој начин, ако Мајкрософт обезбеди надградба на кој било од пакетите, можете само да ја надградите и тоа е тоа. Со ASP.NET Core 2.1, Microsoft воведе метапакет наречен Microsoft.AspNetCore.App, кој ги содржи сите библиотеки испратени од тимовите .NET и ASP.NET и обезбедува директна поддршка без да ве заклучуваат во специфични верзии на трети лица зависности. Сето ова ги подобрува перформансите и приспособливоста на апликациите.

Отворен извор

.NET Core и ASP.NET Core сега се со отворен извор. Тоа е одличен чекор напред за целата .NET заедница затоа што го прави процесот на развој транспарентен и чист, овозможувајќи им на програмерите да бидат вклучени во преглед на кодови, поправање грешки, обезбедување нови карактеристики и можност детално да ги проучуваат користените библиотеки. Едното го подразбира другото и отворениот извор му овозможува на Мајкрософт да го прошири обединувањето .NET и на развој преку платформа. .NET Core има единствена база на кодови што може да се искористи за градење и поддршка на сите платформи, вклучително и Windows, Linux и Mac OS. Очигледно, некои одделни компоненти, како што е специфичен оперативен систем за датотеки, бараат посебна имплементација. Моделот за испорака преку NuGet овозможува да се апстрахираат тие разлики далеку.

Единствен API

Важниот дел од ова за програмерите е дека тоа е единствен API кој работи на различни платформи. Тие не треба да се грижат за тоа бидејќи пакетот веќе содржи различни имплементации за секоја од опкружувањата. Функциите за конфигурација станаа и попријателски расположени преку платформата, заменувајќи ги старите со датотеки JSON или INI и променливи на околината. Крос-платформа значи и дека имате повеќе

широк спектар на оперативни системи што можете да ги користите за распоредување бидејќи Azure ќе поддржува ASP.NET Core и во Linux и во Виндоус ВМ. Од вас зависи што да изберете.

Визуелен код на студио

Како одличен плус за оние кои не сакаат да го инсталираат Visual Studio, сега можете да изберете и Visual Studio Code, Atom, Emacs или дури и да работите со командна линија. На пример, Visual Studio Code е крос-платформски уредник на кодови за Linux, MacOS и Windows. Т ви овозможува да градите, дебагирате и извршувате веб-апликации на ASP.NET Core, веб-апликација на јазол или апликации на конзола .NET Core со можност за користење на IntelliSense, комплетирање на код и интеграција на Git. Ако сте партизан со алатки за командна линија, ќе ми биде драго да слушнете дека сè што е вклучено во градењето, составувањето или водењето на .NET Core апликации може да се направи со командна линија.

Предности на перформансите

Поради помалиот отпечаток на ASP.NET Core, има и некои придобивки од перформансите кои се специфични за .NET Core, но повеќето од нив се применуваат и на .NET Framework и .NET Core.

Што се однесува до лигаментите ASP.NET и IIS, сега ASP.NET Core всушност не бара IIS да работи. Се оди со следниве имплементации на серверот:

• Кестрел е стандарден, меѓу-платформен HTTP сервер за ASP.NET Core;

• IIS HTTP сервер (IISHttpServer) е IIS-процесна имплементација на серверот што се користи со ASP.NET основниот модул;

• HTTP.sys е само HTTP-сервер за Windows, заснован на двигателот на јадрото HTTP.sys и HTI-серверот API.

Најпопуларниот меѓу нив е Кестрел и исто така е стандарден во ASP.NET Core шаблоните. Кога креирате нов проект во Visual Studio, тој автоматски се конфигурира да работи во Кестрел. Вториот всушност се заснова на libuv, кој се користи за I / O работа и поддржува водење повеќе јамки за настани. Сепак, и покрај тоа што е брз, крос-платформски и е оптимизиран за перформанси на проток, тој не ги обезбедува сите функционалности, бидејќи веб-услугата со целосна опција (како IIS) не обезбедува споделување порти, лесна конфигурација на SSL и презапишување URL. Некои од овие карактеристики може да се појават во иднина, но денес Кестрел обично се користи со обратен прокси-сервер, како што се IIS, Nginx или Apache. Тоа е плус и минус во исто време. Кестрелот е навистина брз, всушност, шест пати побрз од Node.js за операции со статички и обичен текст, и обезбедува подобри перформанси за обработка на барањата, но сето ова се постигнува затоа што не е веб-сервер богат со одлики. Па, зошто беше потребно да се создаде таков веб-сервер ако сепак треба да користите прокси-сервер? Бидејќи без него, не сте во можност да ја стартувате вашата апликација на различни платформи без промени. Без Кестрел вклучен

други крос-платформски веб-сервери, кодот за апликација ASP.NET Core ќе се промени за секој од овие веб-сервери да одговара на нивните критериуми за стартување.

Како што можете да видите, ASP.NET Core е голема промена. Сè уште можете да користите рамки како MVC, Web API, SignalR и нивната синтакса нема да се промени екстремно, како и деловната логика напишана со Рамка на субјектот. Тој е сè уште ист код C #, F # итн., А .NET Framework стои зад сите овие. На крајот, крајната цел не беше да се замени .NET Framework со .NET Core туку да се додаде алтернатива за програмерите, да се вклучи новата заедница, да се обезбеди нов оџак за да се вклопат нови промени во развојот и да се прошири платформата во иднината.

* **C # - Програмски јазик за backend**

C # има неколку предности, поради што е толку заеднички јазик во повеќето проекти за развој на Windows. Прво, тој добро се интегрира со Windows. Не ви требаат посебни конфигурации за да добиете програма C # да работи во вашата околина за Windows. Без разлика дали станува збор за веб-апликација, услуга за Виндоус или апликација за работна површина, програмите C # лесно се распоредуваат на мрежата. Сè додека целниот сервер или работната станица поддржува .NET, распоредувањето на вашата програма C # треба да биде непречена транзиција од развој во производство.

Следно, C # е лесно да се најдат дополнителни програмери без разлика дали е за договор или за полно работно време. Ако вашиот бизнис расте и ви требаат дополнителни програмери, C # е еден од вообичаените јазици што го учат програмерите. Исто така, има многу блиска синтакса со Java, така што обично може да најдете развивач кој разбира Java (за мобилно програмирање) и C # (за backend услуги) истовремено.

Ако имате тим на развивачи, Мајкрософт му олесни на тимот да работи заедно на еден проект. Visual Studio има интегрирани алатки за контрола на промените и спојување на кодови. Microsoft нуди верзии на Team Foundation Server на својот софтвер и Enterprise опции за големи развојни тимови.

Конечно, C # е составен јазик, што значи дека кодот складиран на сервер со кој се соочува јавноста е во бинарна форма. Ако вашиот сервер се пробие, хакерот нема автоматски пристап до вашиот изворен код. Со други заеднички јазици како што е PHP, хакерот добива пристап до изворниот код, што потоа може да му овозможи пристап до лозинките на базата на податоци. Со C #, хакерот мора да го декомпајлира или „пробие“ вашиот софтвер пред да може да ги види критичните компоненти.

## 3.1 Authentication

Автентикацијата е процес на утврдување на идентитетот на корисникот. Овластување е процес на утврдување дали корисникот има пристап до ресурс. Во ASP.NET Core, автентикацијата се врши од страна на IAuthenticationService, што се користи од средниот софтвер за автентикација. Услугата за автентикација користи регистрирани ракувачи со автентикација за да завршат активности поврзани со автентикацијата. Примери за активности поврзани со автентикација вклучуваат:

**Автентикација на корисник.**

Одговарање кога неовластен корисник се обидува да пристапи до ограничен ресурс.

Регистрираните управувачи за автентикација и нивните опции за конфигурација се нарекуваат „шеми“.

Шемите за автентикација се специфицирани со регистрирање на услуги за проверка при Стартување. Конфигурирај ги услугите:

Повикувајќи метод за проширување што е специфичен за шемата по повик до сервисите.AddAuthentication (како што се AddJwtBearer или AddCookie, на пример). Овие методи за проширување користат Authentication Builder.AddScheme за да регистрираат шеми со соодветни поставки.

Поретко, се користи директно повикување на AuthenticationBuilder.AddScheme.

Параметарот AddAuthentication JwtBearerDefaults.AuthenticationScheme е името на шемата што ќе се користи стандардно кога одредена шема не е побарана

Ако се користат повеќе шеми, политиките за овластување (или атрибутите за овластување) можат да ја одредат шемата за автентикација (или шемите) од кои зависи проверката на корисникот. Во примерот погоре, шемата за автентикација на колачињата може да се користи со наведување на неговото име (CookieAuthenticationDefaults.AuthenticationScheme шема по дифолт, иако може да се обезбеди друго име при повик на AddCookie).

Во некои случаи, повикот кон AddAuthentication автоматски се извршува со други методи на продолжување. На пример, кога користите ASP.NET основен идентитет, AddAuthentication се нарекува внатрешна проверка.

Средниот софтвер за автентикација е додаден во Стартување. Конфигурирајте со повикување на методот за проширување UseAuthentication на IApplication Builder на апликацијата. Повикувањето на Употреба за автентикација го регистрира средниот софтвер што ги користи претходно регистрираните шеми за проверка. Повикајте UseAuthentication пред кој било среден софтвер што зависи од автентикација на корисниците. Кога користите рутирање на крајната точка, повикот до UseAuthentication мора да оди:

По UseRout, така што информациите за маршрутата се достапни за одлуки за автентикација.

Пред UseEndpoints, така што корисниците ќе бидат автентицирани пред да пристапат до крајните точки.

**Концепти за автентикација**

Автентикацијата е одговорна за обезбедување на Главниот директориум на побарувања за овластување за донесување одлуки и за дозвола против. Постојат повеќе пристапи кон шемата за проверка за да избере кој управувач со автентикација е одговорен за генерирање на точен пакет на тврдења:

• Шема за автентикација

• Стандардната шема за проверка, дискутирана во следниот дел.

• Директно поставете HttpContext.User.

Нема автоматско испитување на шеми. Ако стандардната шема не е одредена, шемата мора да биде наведена во атрибутот за овластување, во спротивно, се случува следнава грешка:

InvalidOperationException: Не е наведена автоматска шема и не е пронајден DefaultAuthenticateScheme. Стандардните шеми може да се постават со помош на AddAuthentication (string defaultScheme) или AddAuthentication (Action <AuthenticationOptions> configureOptions).

**Шема за проверка**

Шемата за проверка може да избере кој управувач со автентикација е одговорен за генерирање на точниот пакет на побарувања. За повеќе информации, видете во “Овластете со одредена шема“.

Шема за проверка е име што одговара на:

• Ракувач со автентикација.

• Опции за конфигурирање на таа специфична инстанца на управувачот.

Шемите се корисни како механизам за упатување на автентикација, предизвик и забрана на однесување на поврзаниот управувач. На пример, политиката за овластување може да користи имиња на шеми за да одреди која шема за автентикација (или шеми) треба да се користи за автентикација на корисникот. Кога ја конфигурирате автентикацијата, вообичаено е да се наведе стандардната шема за проверка. Стандардната шема се користи освен ако ресурс не побара специфична шема. Исто така е можно да:

• Наведете различни стандардни шеми што ќе ги користите за автентикација, предизвик и забрана на дејствија.

• Комбинирајте повеќе шеми во една користејќи шеми на политики.

**Ракувач со автентикацијата**

Управувач со автентикација:

• Дали е тип што го спроведува однесувањето на шемата.

• Излезе од IAuthenticationHandler или AuthenticationHandler <Опции>.

• Ја има примарната одговорност за автентикација на корисниците.

Врз основа на конфигурацијата на шемата за проверка и контекстот на дојдовното барање, ракувачи со автентикација:

• Конструирајте предмети за автентикација кои го претставуваат идентитетот на корисникот доколку автентикацијата е успешна.

• Вратете „без резултат“ или „неуспех“ ако проверката е неуспешна.

• Имајте методи за предизвик и забрана за дејствија за кога корисниците се обидуваат да пристапат до ресурсите:

o Тие не се овластени за пристап (забрануваат).

o Кога тие не се автентицирани (предизвик).

RemoteAuthenticationHandler<TOptions> vs AuthenticationHandler<TOptions>

* RemoteAuthenticationHandler <TOptions> е класа за автентикација која бара чекор оддалечена автентикација. Кога е завршен чекор за автентикација на далечинскиот управувач, управувачот повикува назад кон CallbackPath поставен од управувачот. Управувачот го завршува чекорот за автентикација користејќи ги информациите пренесени на патеката за повратен повик HandleRemoteAuthenticateAsync. OAuth 2.0 и OIDC ја користат оваа шема. JWT и колачињата не прават бидејќи тие можат директно да ги користат заглавието и колачето на носителот за да се проверат. Давател од далечина е хостиран во овој случај:
* Дали е давател на автентикација.
* Примерите вклучуваат Фејсбук, Твитер, Гугл, Мајкрософт и кој било друг давател на ОИДЦ кој се справува со автентикација на корисниците користејќи механизам за ракувачи.

**Автентикација**

Дејството за автентикација на шемата за автентикација е одговорно за конструирање на идентитетот на корисникот врз основа на контекстот на барањето. Враќа AuthenticateResult означувајќи дали автентикацијата била успешна и, доколку е така, идентитетот на корисникот во билетот за автентикација. Погледнете ја AuthenticateAsync. Примери за автентикација вклучуваат:

• Шема за автентификација на колачиња што го конструира идентитетот на корисникот од колачињата.

• Шема на носители на JWT десерилизирање и валидирање на токен на носители на JWT за конструирање на идентитетот на корисникот.

**Предизвик**

Овластување се повикува на предизвик за проверка кога неовластен корисник бара крајна точка за која е потребна автентикација. Предизвик за автентикација се издава, на пример, кога анонимен корисник бара ограничен ресурс или кликне на врската за најавување. Овластувањето предизвикува предизвик користејќи ги наведените шеми за автентикација или стандардните ако не е одредено Погледнете ChallengeAsync. Примерите за предизвици за автентикација вклучуваат:

• Шема за проверка на колачиња што го пренасочува корисникот на страната за најавување.

• Шема на носители на JWT враќање на резултат 401 со заглавие на автентикација www:

Акција за предизвик треба да му каже на корисникот кој механизам за проверка треба да го користи за пристап до бараниот ресурс.

**Забрани**

Акцијата за забрана на шемата за автентикација се повикува со Овластување кога автентициран корисник се обидува да пристапи до ресурс до кој не им е дозволен пристап. Погледнете ForbidAsync. Примерите забранети за проверка вклучуваат:

• Шема за проверка на колачиња што го пренасочува корисникот на страница што укажува на пристап е забранета.

• Шема на носители на JWT со резултат 403.

• Прилагодена шема за проверка пренасочувајќи се на страница каде што корисникот може да побара пристап до ресурсот.

Забрането дејство може да го извести корисникот:

• Тие се проверени.

• Не им е дозволен пристап до бараниот ресурс.

Погледнете ги следниве врски за разлики помеѓу предизвик и забрана:

• Предизвик и забрана со управувач на оперативни ресурси.

• Разлики помеѓу предизвик и забрана.

Authentication providers per tenant

ASP.NET Core рамката нема вградено решение за автентикација на повеќе станари. Иако секако е можно за клиентите да напишат една, користејќи ги вградените карактеристики, им препорачуваме на клиентите да ја разгледаат јадрото на овоштарникот за оваа намена.

Оrchard core е:

• Модуларна и рамка на апликации со повеќе станари со отворен извор, изградена со ASP.NET Core.

• Систем за управување со содржини (CMS) изграден над таа рамка на апликации.

## 3.2 Слоева архитектура.

Слоевата архитектура, како што јас разбирам, е организација на структурата на проектот во четири главни категории: презентација, апликација, домен и инфраструктура. Секој од слоевите содржи предмети поврзани со посебната загриженост што ја претставува.

Презентацискиот слој ги содржи сите класи одговорни за презентирање на корисничкиот интерфејс до крајниот корисник или испраќање на одговор на клиентот (во случај да работиме длабоко во задниот крај).

Апликацискиот слој ја содржи целата логика што ја бара апликацијата за да ги исполни своите функционални барања и, во исто време, не е дел од правилата на доменот. Во повеќето системи со кои сум работел, слојот за апликација се состоел од услуги што ги оркестрираат објектите на доменот за да се исполни сценариото за употреба.

Слојот на доменот го претставува основниот домен, претежно се состои од субјекти на домен и, во некои случаи, услуги. Деловните правила, како што се инваријантите и алгоритмите, сите треба да останат во овој слој.

Инфраструктурниот слој (познат и како постојан слој) ги содржи сите класи одговорни за вршење технички работи, како што се перзистирање на податоците во базата на податоци, како DAO, складишта или што било друго што користите

Постојат две важни правила за класична слоевита архитектура правилно да се спроведе:

Сите зависности одат во една насока, од презентација до инфраструктура. Па, управувањето со упорноста и доменот е малку незгодно затоа што инфраструктурниот слој честопати ги зачувува објектите на доменот, така што всушност знае за класи во доменот

Ниту една логика поврзана со загриженоста на еден слој не треба да се става во друг слој. На пример, не треба да се прават никакви логики за домен или пребарувања за базата на податоци во корисничкиот интерфејс.

**Суштината на слоевитата архитектура**

Архитектурата е вид на преоптоварен термин, па веројатно треба да истражуваме подлабоко во она што овој термин навистина значи во контекст на слоевите. Главната идеја зад слоевитата архитектура е поделба на загриженоста - како што веќе рековме, сакаме да избегнеме мешање на кодот на доменот или базата на податоци со нештата за интерфејс, итн. Вистинската идеја за одвојување на проект во слоеви сугерира дека ова одделување на проблеми треба да постигнете со организација на изворниот код. Ова значи дека освен некои упатства за тоа кои проблеми треба да ги одделиме, Архитектурата со слоеви не ни кажува ништо друго за дизајнот и спроведувањето на проектот. Ова подразбира дека треба да го надополниме со некои други архитектонски процеси, како што се некои однапред дизајнирани, дневни сесии за дизајн, па дури и целосен дизајн на управување со домен. Која опција и да ја избереме, не е важна, барем заради слоевитоста, но треба да запомниме: архитектурата со слоеви не ни дава ништо друго освен упатство за тоа како да го организираме изворниот код.

**Спроведување на слоевита архитектура**

Опремени со знаење за слоевите што треба да се создадат, односите меѓу нив и суштината на архитектурата, ние сме подготвени да ја спроведеме. Како што веројатно повеќето од вас очекуваат, ќе го исечеме системот во слоеви со создавање на посебен пакет за секој од нив. Кога станува збор за примена на правилата за зависност и поделба, работите не се толку очигледни. Може да се обиде да го стави секој слој во посебен моден модул, но тогаш фаќањето на чудната врска помеѓу доменот и упорноста не би било лесно. Јас обично се држам со пакети и користам здрав разум заедно со прегледи за кодови за да се осигурам дека ниту едно од правилата не е прекршено.

**Предности на слоевита архитектура**

Иако можеби некои од вас сè уште не веруваат, архитектурата со слоеви има некои придобивки, вклучително и:

Едноставност - концептот е многу лесен за учење и видлив во проектот на прво разбирање.

Доследно во различни проекти - слоевите и така целокупната организација на кодови е прилично иста во секој слоевит проект.

Загарантирано раздвојување на грижите - само загрижености што имаат слој и до тој степен што ќе се држите до правилата на слоевита архитектура, но тоа е многу лесно со кодната организација што ја подразбира

Прегледливост од техничка гледна точка - кога сакате да промените нешто во некои / сите предмети од даден вид, тие се наоѓаат многу лесно и се чуваат заедно.

Недостатоци на слоевитата архитектура

И, се разбира, слоевитата архитектура не е совршена. Некои од неговите недостатоци вклучуваат:

Нема вградена приспособливост - кога проектот расте премногу, треба сами да пронајдете клуч за негово понатамошно организирање. Принципите на слоевита архитектура нема да ви помогнат.

Случаи на скриена употреба - не можете навистина да кажете што прави еден проект со едноставно гледање во организацијата за кодови. Треба барем да ги прочитате имињата на часовите и, за жал, понекогаш дури и спроведувањето.

Ниска кохезија - часовите што придонесуваат за вообичаени сценарија и деловни концепти се далеку едни од други, бидејќи проектот е организиран околу раздвојување на техничките проблеми.

Без инверзија на зависност - во класичната слоевита архитектура, зависностите се директни и, концептуално, промените од ниско ниво на инфраструктурен слој можат да се шират во поважни повисоки слоеви.

Кога да се примени слоевита архитектура

Отпрвин, сакав да напишам неколку случаи кога јас ја користам слоевитата архитектура, но мислам дека можеме да користиме повеќе методички пристап заснован на неговите добри и лоши страни:

Едноставност: тоа е очигледно важно за секого. Непостоечката крива на учење го прави особено одржлив за тимови со ниско искуство и леснотијата на аплицирање за проекти без буџет за инвестирање во понапредна архитектура.

Конзистентност: повеќе мали проекти управувани од иста група луѓе, како што е внатрешна архитектура за микросервиси

Поделба на загриженоста: тимови со слабо искуство.

Техничка прегледливост: до одреден степен, им помага на сите. Тоа зависи од тоа што се обидувате да најдете во даден момент.

Без вградена приспособливост: проекти што би требало да останат мали, како што се микросервиси или поголеми проекти кои можат лесно да се факторизираат во помали делови.

Случаи на скриена употреба и мала кохезија: Проекти без или неколку сложени деловни сценарија, како што се CRUD операции или едноставни REST услуги.

Без инверзија на зависност: проекти со малку зависности од инфраструктурата, во кои нема сериозни промени на ниско ниво за реално да се шират повисоко.

## 3.3 Kiss and dry

KISS е термин создаден во 1960-тите, кој требаше да го одразува пристапот на создавање едноставни трајни авиони што можат лесно да се поправат во итни случаи. Оваа идеја оттогаш е усвоена од многу гранки на науката и инженерството.

KISS е термин исто така користен при развој на решенија за програмски проблеми. Буквално преведен, КИСС значи „држи го едноставно, глупаво“ или „чувај го глупаво едноставно“.

Сепак, се претпоставува, изјавата не требаше да звучи негативно, туку да се сугерира едноставен дизајн што ја прави услугата и одржувањето на ниво на детската игра.

Употребата на KISS во програмирањето може да се разгледа во три аспекти:

1. Поедноставување за сегашен и иден развој на соработка на вашиот код. Од моја перспектива, ова не е ништо друго освен вешта примена на солидни принципи.

Да се ​​напише само доволно код за да се помине единичен тест, кој се однесува на TDD и првиот солиден принцип - единствена одговорност.

Интуитивни пристапи кон пишување на софтвер, достапен и едноставно разбран за секого. Само програмирањето KISS не поставува цврсти и брзи правила во врска со спроведувањето на кодот, како што е случајот со повеќето SOLID принципи, кои се општи упатства за тоа како да се создаде SOLID код.

Не повторувајте се (Сува, или понекогаш не повторувајте се) е принцип на развој на софтвер насочен кон намалување на повторувањето на софтверските обрасци, [1] негово заменување со апстракции или користење на нормализација на податоците за да се избегне вишок.

Принципот Сува е наведено како „Секое знаење мора да има единствена, недвосмислена, авторитетна застапеност во рамките на системот“. Принципот е формулиран од Енди Хант и Дејв Томас во нивната книга „Прагматичен програмер“. Тие го применуваат доста широко за да вклучуваат „шеми на бази на податоци, планови за тестирање, систем за градење, па дури и документација“. Кога сувиот принцип ќе се примени успешно, модификација на кој било единствен елемент на системот не бара промена во другите логички неповрзани елементи. Дополнително, елементите што се логички поврзани, сите се менуваат предвидливо и подеднакво, и така се чуваат во синхронизација. Освен што користат методи и потпрограми во нивниот код, Томас и Хант се потпираат на генератори на кодови, системи за автоматско градење и јазици за скрипти за да го набљудуваат принципот СУВ низ слоевите.

Како што можете да видите, слоевитата архитектура има свои светли и темни страни. За мене, неговата едноставност и конзистентност го прават добро погоден за микросервиси без премногу сериозна деловна логика. Може да се постави прашање дали таквите микросервиси треба да постојат на прво место, но реалноста на факторирање на големи монолити често ги прави помало зло. Најважната лекција што треба да ја одземете од овој напис е: слоевата архитектура е организирање код за добро раздвојување на проблемите и ништо друго.

# 4. Frontend

Vue.js е JavaScript библиотека за градење веб-интерфејси со користење на моделот на архитектура MVVM (Model-View-ViewModel). Vue.js е целосно дизајниран од нула за да биде постепено усвоен. Значи, ако сте запознати со HTML, CSS и JavaScript, тогаш Vue.js е многу лесен за почеток,

Во многу случаи, Vue.js ги заменува Angular, Ember и React - ова е добра работа и го прави различно работењето со него. Покрај ова, Vue се комбинира со други модерни библиотеки со алатки и поддршка за да овозможи совршена апликација на една страница.

**1. Многу мала големина**

Успехот на рамката JavaScript зависи од нејзината големина. Колку е помала големината, толку повеќе ќе се користи. Една од најголемите предности на Vue.js е неговата мала големина. Големината на оваа рамка е 18–21 KB и не му треба време на корисникот да ја преземе и користи. Ги победи сите обемни рамки како React.js, Angular.js и Ember.js.

**2. Лесно се разбираат и развиваат апликации**

Едноставноста е дел од ДНК на Vue. Една од причините за популарноста на оваа рамка е тоа што е прилично лесно да се разбере. Корисникот може лесно да додаде Vue.js на својот веб-проект поради неговата едноставна структура. И малите и големите шаблони за скали може да се развијат преку оваа рамка.

**3. Познат по својата флексибилност**

Vue.js доаѓа со совршен баланс за брзо пишување и извршување директно од прелистувачот. Може да изградите комплексна апликација користејќи ES6, JSX, компоненти, рутирање, збирки и сл. Ова го прави претпочитан избор да се користат Vue.js за развој на апликации преку платформа. Ви овозможува пристап до широк спектар на различни средини.

**4. Добра детална документација**

Исто така, нуди предност на детална документација, што го прави поедноставно за развивачите да ја напишат и извршат првата апликација. Потребно е само да имате основно познавање на HTML или JavaScript за развој на веб-страница или апликација. Ова се должи на фактот дека карактеристиките на документацијата се доволно сеопфатни.

**5. Се осврнува на недостатоците на AngularJS и ReactJS**

Еван Вие, творецот на Vue.js ја воведе оваа популарна рамка JavaScript. Тој го создаде Vue.js со цел да се осврне на недостатоците на двата најпопуларни рамки на JavaScript - AngularJS и ReactJS.

Така, тој разви лесен раствор преку екстракција на делови од AngularJS без додавање на кој било друг концепт. Тој сакаше да ги комбинира најдобрите карактеристики и на рамките и на тој начин се приближи до новиот Vue.js.

Vue.js го прифати пристапот базиран на компоненти, проток на податоци на еден начин за хиерархија на компонентите, способност за виртуелно прикажување, високи перформанси, реквизити и улогата што ја игра државното управување со апликациите во веб апликации итн. Vue е многу моќна алатка за градење искуства за збогатување на мрежата, а рамката е многу зрела и добро одржувана од заедницата.

Nuxt.js е рамка на високо ниво која е изградена со употреба на Vue. Тоа го поедноставува развојот на веб апликации на апликации со една страница или универзални Vue.js. Оваа рамка на Vue.js ги апстрахира деталите за дистрибуцијата на кодот на серверот и клиентот, така што ќе можете да се фокусирате само на развој на веб апликации.

Целта на оваа рамка е таа да биде доволно флексибилна за развивачите на веб-страници да ја користат како главна база на проекти.

Бидејќи поголемиот дел од она што го прави Nuxt.js се случува за време на фазата на развој на веб-апликации, добивате многу функции со само неколку дополнителни KBs додадени на вашите JS-датотеки.

Дозволете ни да ги истражиме причините зошто можеби ќе ви треба Nuxt за вашиот следен проект за развој на веб.

**Како работи Nuxt.js**

Кога корисник посетува веб-апликација Nuxt.js или се движи на една од нејзините веб-страници преку <nuxt-link>, еве што се случува:

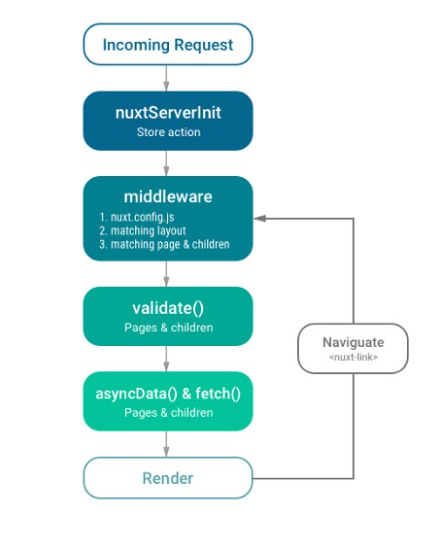
1. Кога корисникот првично ја посетува веб-апликацијата, ако акцијата nuxtServerInit е јасно дефинирана во продавницата, Nuxt.js ќе го повика ова и ќе ја ажурира продавницата.

2. Nuxt.js извршува кој било постоечки среден софтвер за посетената веб-страница. Ја проверува датотеката nuxt.config.js на прво место за глобалниот среден софтвер, потоа ја проверува соодветната датотека за изгледот за таа барана страница и, конечно, ја проверува веб-страницата и нејзините **деца** за среден софтвер. Средниот софтвер е приоритет по тој одреден редослед.

3. Ако таа одредена маршрута што е посетена е динамична маршрута и постои валиден () метод за тоа, маршрутата е јасно потврдена.

4. Потоа, Nuxt.js ги повикува методите за преземање () и asyncData () за вчитување на податоци пред да ја пренесете страницата. Методот asyncData () е широко користен за преземање на податоци и нивно прикажување од страната на серверот, додека методот за преземање () е широко користен за пополнување на продавницата пред рендерирање на веб-страницата.

5. На последниот чекор, се дава страницата што содржи соодветни податоци.



**Слика Kако функционира Nuxt.**

Предности на Nuxt.js

1. Создадете универзални веб-апликации без **кавга**

Една од најголемите предности на Nuxt.js е тоа што помага полесно да се создаваат универзални апликации.

Што е универзална апликација?

Универзална веб-апликација се користи за опишување на JS-кодот што може да се изврши и од страната на серверот и од страната на клиентот.

Многу напредни рамки на JS, како Vue, се насочени кон создавање апликации на една страница, исто така наречени SPA. Има низа придобивки од имањето СПА преку традиционална веб-страница.

На пример, можете да изградите многу пријатни кориснички интерфејси кои се ажурираат доста брзо. Но, СПА имаат и некои недостатоци, како долго време на вчитување, и Google се бори со овие страници затоа што првично на страницата нема содржина за индексирање заради оптимизација на машините за пребарување. Целата содржина е генерирана со ЈС по фактот.

Универзална веб-апликација е да имате апликација на една страница, но наместо да имате празна страница со index.html, претходно ја вчитувате веб-апликацијата на сервер и испраќате изречена HTML како одговор на барање на прелистувач за секоја рута да се забрза, вчитувајте време и подобрете ја оптимизацијата на пребарувачот со тоа што на Google ќе му биде полесно да ја използи веб-страницата.

Nuxt.js неизмерно ви помага поедноставно да напишете универзални веб апликации.

Создавањето универзални веб-апликации може да биде мачна задача затоа што треба да направите плејада на конфигурации и од страната на клиентот и од серверот.

Ова е проблемот што Nuxt.js има за цел да го реши за апликациите Vue. Nuxt.js го прави прилично едноставно споделување на истиот код помеѓу серверот и клиентот за да можете да се фокусирате на логиката на вашата веб-апликација.

Nuxt.js исто така ви овозможува пристап до разни својства како што е isClient и isServer на вашите компоненти за лесно да одлучите дали рендерирате нешто на серверот или на клиентот.

Исто така, постојат специјални компоненти како компонентата no-SSR, која е широко користена за намерно спречување на рендерирање на компонентата од страната на серверот.

Во последниот, Nuxt.js ви дава пристап до методот asyncData во внатрешноста на разни компоненти што можете да ги користите за да земете податоци и да ги направите на програмирање од страна на серверот.

Сега, тоа е врвот на ледениот брег за тоа како Nuxt.js ви помага при креирање универзални веб-апликации. Можете да дознаете повеќе за тоа што нуди Nuxt за рендерирање на универзални веб-апликации од неговиот упатство за корисник.

**2. Статички прикажете ги вашите апликации Vue и искористете ги сите придобивки на универзалната апликација без сервер**

Една од најголемите иновации на Nuxt.js доаѓа со неговата наредба генерира команда. Оваа команда генерира статичка верзија на веб-апликацијата. Generaе генерира HTML за секоја од рутите и ќе ја стави во својата датотека.

На пример, ако ги имате подолу споменатите страници (термин за правци на Nuxt):



Nuxt will generate the following folder structure:



Придобивките од тоа се многу слични на предностите на универзалните веб-апликации. Таму има обележување за да се изврши побрзо вчитување на страницата и да им се помогне на роботите и пребарувачите на социјалните медиуми да ја ползат веб-страницата.

Разликата е во тоа што повеќе не ви треба сервер. Сè е генерирано за време на фазата на развој на веб.

Тој е моќен додека ги добивате придобивките од универзалното рендерирање без потреба од сервер. Можете само да ја хостирате вашата веб-апликација на Amazon S3 и GitHub Pages.

Можете да прочитате повеќе за ова во статичкиот генериран дел од документите на Nuxt.js.

**3. Добијте автоматско разделување на кодови**

Nuxt.js лесно може да генерира стабилна верзија на вашата веб-страница со специјална конфигурација на веб-пакет.

За секоја рута (страница) што е генерирана статистички, рутата добива и своја JavaScript-датотека, само со потребниот код за извршување на таа одредена рута. Ова може да помогне при брзина затоа што ја одржува големината на JS-датотеката мала, во однос на големината на целата ваша веб-апликација.

**4. Поставување преку командната линија со почетниот образец**

Nuxt.js исто така обезбедува почетен образец кој се нарекува почетен образец кој ви ги дава сите скелиња потребни за да започнете со проект, со добра структура на папки за организација.

Осигурете се дека имате инсталирано Vue CLI и извршете ја следнава команда:



Оттука, само cd во веб-апликацијата и стартувајте npm инсталирајте и треба да го носите ова понатаму.

**5. Добијте одлична структура на проектот по дифолт**

Во многу мали веб-апликации на Vue.js, на крајот управувате со структурата на кодот најдобро што можете во различни датотеки.

Стандардната структура на веб-апликации Nuxt ви овозможува одлична почетна точка за организирање на вашата веб-апликација на разбирлив начин.

Еве неколку директориуми со кои ве поставува:

• Компоненти - Папка за да можете да ги организирате вашите компоненти Vue.js поединечно.

• Распоред - Папка за да ги содржи вашите главни распореди на веб апликации.

• Pages - Папка за да ги содржи маршрутите на вашата веб-апликација. Nuxt ги чита сите датотеки Vue во овој директориум и го гради рутерот за веб-апликации.

• Продавница - Папка за да ги содржи сите датотеки за чување Vuex на вашата апликација.

**6. Поставете сервер за автоматско ажурирање за лесен развој**

Во споредба со самото поставување на овој процес или процесот на промена-освежување на кој се навикнати развивачите на веб-апликации, креирањето со Nuxt е лесно. Се поставува со сервер за развој со автоматско ажурирање.

Додека развивате и работите на тие неколку датотеки .vue, Nuxt.js користи конфигурација на веб-пакет за да провери за сите промени и собира сè за вас.

Покрај ова, можете да ја извршите командата npm run dev во рамките на проектот Nuxt и тој ќе го постави серверот за развој на веб.

# 5. Stripe

Порта за плаќање е апликација за е-трговија која овластува обработка на кредитни картички и директна дебитна преку вашата веб-страница. Нудењето безбеден портал за плаќање е витален дел од веб-страница за е-трговија, бидејќи веројатно ќе влијае на тоа дали клиентот ќе направи или нема да направи трансакција.

Има толку многу достапни апликации за е-трговија со различни понуди на услуги, одлики и поволности, но ние ќе ви кажеме за Stripe и зошто тоа е нашиот претпочитан портал за плаќање.

Stripe е новото дете на блокот во извесна смисла и иако започна во 2011 година, портата за плаќање стана достапна само во Австралија минатата година. Таа е една од единствените платежни компании во Австралија која поддржува мулти-валута и може да обработува кредитни картички од над 110 различни земји, без потреба за поставување посебни трговски објекти. Подолу, одблизу ги разгледуваме карактеристиките на Stripe.

Карактеристики на ленти за веб-страници за е-трговија

Една од најголемите придобивки на Stripe е што му овозможува на корисникот да остане на вашата страница кога плаќа, наместо да се пренасочува на друга страница. Нема поставени трошоци или месечни надоместоци и поставувањето е моментално. Другите придобивки вклучуваат:

Лесно се поставуваат периодични плаќања

Пробни периоди со задоцнета исплата

Може да им понуди на вашите клиенти попусти и купонски кодови

Ви овозможува да поставите плаќања за претплата

Ако клиентот ја смени претплатата на средно наплата, Stripe ви дава можност да про-рата наплаќате

Способност за лесно полнење на клиент без зачувување детали за кредитни картички

Одличен систем за потсетување

Без надоместоци за поврат

Информативен интерфејс на графикони и графикони, ги прави податоците разбирливи и јасни

Беспрекорно искуство со исплата значи зголемени конверзии

Перспектива на веб-развивач

Користевме многу портали за плаќање како веб-развивачи и сметаме дека некои се многу необични и одземаат многу време. Откривме дека Stripe е еден од најлесните за поставување, интеграција и одржување, бидејќи има одличен API. Исто така, му дава на инвеститорот целосна контрола врз изгледот и чувството на целото искуство со плаќањето.

Придобивките од перспектива на клиентот

Stripe е многу помоќна алтернатива на Paypal, што ги вади вашите корисници надвор од страницата кога вршат плаќање. Второ, Stripe не е врзана за одредена банка и е една од единствените платежни компании во Австралија која поддржува мулти-валута, без потреба да аплицира за сметка за трговец на Интернет. Stripe исто така нуди безбедна интеракција со своите сервери за да ги зачува безбедно податоците на вашите клиенти. Сè на сè, без поставени такси, без месечни такси, без минимални трошоци и без такси за чување картички, Stripe прави сè лесно.

# 6. Elastic search

Elasticsearch е високо скалабилен мотор за пребарување и анализатор со целосен текст со отворен извор. Тоа ви овозможува да зачувувате, пребарувате и анализирате големи количини на податоци брзо и во блиско реално време. Општо се користи како основен мотор / технологија што ги напојува апликациите што имаат сложени карактеристики и барања за пребарување. Elasticsearch обезбедува дистрибуиран систем на врвот на Lucene StandardAnalyzer за индексирање и автоматско погодување на типот и користи JSON REST API за да се повика на карактеристиките на Lucene.

Лесно е да се постави надвор од кутијата бидејќи се испраќа со разумни стандардни вредности и ја крие комплексноста од почетниците. Има кратка крива на учење за да ги сфати основите, така што секој што има малку напор може да стане продуктивен многу брзо. Таа е без шема, користејќи некои стандардни вредности за да ги индексира податоците.

Во случај на потрошувачи кои бараат информации за производи од каталозите на веб-страниците за трговија, се соочуваат со проблеми како што е долго време во барањето информации за производите. Ова доведува до слабо корисничко искуство, а за возврат го пропушта потенцијалниот клиент. Денес бизнисот бара алтернативни начини каде што се зачувува голема количина на податоци на таков начин што враќањето е брзо.

Ова може да се постигне со усвојување на NOSQL наместо RDBMS (систем за управување со релациони бази на податоци) за складирање на податоци.

Elasticsearch стои како NOSQL DB затоа што:

• лесен за употреба

• Има одлична заедница

• Договорност со JSON

• Случаи со широка употреба

**Компоненти на заднината**

За подобро разбирање на Elasticsearch и неговата употреба е добро да имате општо разбирање за главните компоненти на задниот дел.

**Јазол**

Јазол е единствен сервер кој е дел од кластер, ги зачувува нашите податоци и учествува во индексирањето и можностите за пребарување на кластерот. Исто како кластерот, јазолот се идентификува со име кое по дифолт е случаен универзално уникатен идентификатор (UUID) што му е доделен на јазолот при стартување. Можеме да ги уредиме стандардните имиња на јазли во случај да сакаме.

**Cluster**

Кластерот е збирка на еден или повеќе јазли што заедно ги собира целите ваши податоци и обезбедува федерални можности за индексирање и пребарување. Може да има N јазли со исто име на кластерот.

Еластичното пребарување работи во дистрибуирана средина: со репликација на меѓу-кластери, секундарниот кластер може да започне да дејствува како топла резервна копија.

**Индекс**

Индексот е збир на документи кои имаат слични карактеристики. На пример, можеме да имаме индекс за одреден клиент, друг за информација за производ и друг за поинаква типологија на податоци. Индексот се идентификува со единствено име кое се однесува на индексот при извршување операции за пребарување, ажурирање и бришење индексирање. Во еден кластер, можеме да дефинираме индекси колку што сакаме. Индексот е сличен на базата на податоци во RDBMS.

**Документ**

Документот е основна единица на информации што може да се индексира. На пример, можете да имате индекс за вашиот производ, а потоа и документ за еден клиент. Овој документ е изразен во JSON (JavaScript Object Notation) што е сеприсутен формат за размена на податоци на Интернет. Аналогија на единечна суровина во ДБ.

Во рамките на индекс, можете да зачувате онолку документи колку што сакате, така што во истиот индекс да имате документ за еден производ, а друг друг за една нарачка.

**Шард и реплики**

Еластичното пребарување обезбедува можност да го поделите вашиот индекс на повеќе парчиња наречени парчиња. Кога креирате индекс, можете едноставно да го дефинирате бројот на парчиња што сакате. Секоја фрагмент сама по себе е целосно функционален и независен „индекс“ што може да биде хостиран на кој било јазол во кластерот.

Раздвојувањата се важни причини што ви овозможуваат хоризонтално да го поделите обемот на вашите податоци, потенцијално исто така и во повеќе јазли, паралелизирајќи ги операциите со што се зголемуваат перформансите. Исто така, парчињата може да се користат со правење повеќе копии на вашиот индекс во копии, што во облак опкружувања може да бидат корисни за да се обезбеди висока достапност.

**Еластичен оџак**

Иако пребарувачот е во неговото јадро, корисниците започнаа да го користат Elasticsearch за дневници и сакаа лесно да ги проголтаат и визуелизираат. Еластичното пребарување, Логсташ, Кибана се главните компоненти на еластичниот оџак и се познати како ELK.

**Кибана**

Кибана ви овозможува да ги визуелизирате вашите податоци за Еластично пребарување и да се движите по Еластичен оџак. Можете да изберете начин да им дадете облик на вашите податоци со започнување со едно прашање за да откриете каде ќе ве води интерактивната визуелизација. Можете да започнете со класичните графикони (хистограми, графикони со линии, графикони со пити, сончања и така натаму) или да дизајнирате сопствена визуелизација и да додадете гео податоци на која било мапа.

Можете исто така да извршите напредна анализа на временски серии, да пронајдете визуелен однос во вашите податоци и да истражувате аномалии со одликите за машинско учење.

За повеќе детали, погледнете на официјалната страница.

Случаи со употреба на еластично пребарување

Еластичното пребарување може да се користи на толку различни начини што ми е тешко да ги забележам сите најинтересни случаи на употреба. За едноставност, избрав три главни категории и три главни компании користат случаи, ако сакате да нуркате повеќе, погледнете на нивната страница за случаи за употреба.

Главна продавница за податоци: креирајте систем за пребарување каталог, зачувување документи и најавување.

Комплементарна технологија: додадете способности за визуелизација на SQL, mongoDB, индексирање на фрлија и пребарување во Hadoop или додајте обработка и складирање на kafka.

Адитивна технологија: во случај да имате веќе најавено Elasticsearch, можеби ќе сакате да додадете мерила, мониторинг и аналитички способности.

Напомена: целиот код, заедно со сите позадински слики и материјали, се наоѓаат на следниот линк:

**https://github.com/DarkPheonixJW24/SuplementShop.git**

# Заклучок

Веб страницата го отсликува идентитетот на една институција и често е прва точка за контакт за идните консумери.Администраторите треба да ја гледаат својата веб страница како алатка за продажба на институцијата и соодветно да ја дизајнираат нејзината содржина и карактеристики. Да се има соодветна содржина, јазик, корисничко искуство, пристапност и ефективност за посетителите е особено важно.

Битен дел во дизајнирањето се респонзивните веб продавници, кои денес се во најголем тренд. Сами по себе имаат предности на прилагодување за сите екрани, тоа исто така е голема предност за корисниците.Сите посетители на овие страници ќе добијат квалитетно корисничко искуство благодарение на респонзивноста од страницата. Овој дизајн овозможува лесно одржување,се е во една страница.

# Литература

1. https://www.uml-diagrams.org/examples/online-shopping-use-case-diagram-example.html
2. https://www.uml-diagrams.org/examples/online-shopping-credit-cards-use-cases-example.html
3. https://docs.microsoft.com/en-us/ef/core/
4. https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-5.0&tabs=visual-studio
5. https://docs.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-5.0
6. https://en.wikipedia.org/wiki/KISS\_principle
7. https://en.wikipedia.org/wiki/Don%27t\_repeat\_yourself
8. https://nuxtjs.org/docs/2.x/get-started/installation
9. https://vuejs.org/v2/guide/
10. https://stripe.com/docs/payments/accept-a-payment