

# PulseBuy

Изработиле:

- Андреј Тодоровски 201084
- Дарко Сасански 201065
- Бојан Трпески 201091

Ментор: Проф. Бобан Јоксимовски

PulseBuy претставува напредна E-Commerce апликација која овозможува корисниците да купуваат технологија како мобилни телефони, лаптопи, тастатури, глувчиња, компјутери и други поврзани производи. Апликацијата вклучува ажурирање на залихите на производите во реално време, што значи дека корисниците секогаш имаат точни информации за достапноста на производите. Дополнително, апликацијата нуди комуникација со корисничка поддршка во реално време, овозможувајќи им на корисниците да добијат брзи и точни одговори на нивните прашања и проблеми.

PulseBuy исто така нуди и напредни функции како што се управување со кориснички профили, креирање на листи на желби, оставање и читање на рецензии за производи и слично. Администраторите имаат пристап до контролен панел преку кој можат да управуваат со производите, нарачките, корисниците и распродажбите.

Со својата лесна за користење и интуитивен интерфејс, PulseBuy овозможува одлично корисничко искуство, апликацијата е дизајнирана да биде брза, безбедна и адаптирана за сите потреби на современите купувачи.

Апликацијата е изработена во:

NestJS - Како основа на backend-от, NestJS овозможува структуриран и модуларен развој на серверската страна. Со својот моќен Dependency Injection систем и поддршката за TypeScript, овозможува лесно создавање на скалабилни и одржливи апликации.

SvelteKit - На клиентската страна, SvelteKit се користи за изградба на брзи и динамични кориснички интерфејси. Со својот уникатен пристап кон реактивност и минимален bundle size, SvelteKit обезбедува одлични перформанси и корисничко искуство.

PostgreSQL - За складирање на податоците, се користи PostgreSQL, моќна и сигурна релациона база на податоци која овозможува сложени операции и интегритет на податоците.

# Стартување на апликацијата

PulseBuy се состои од три дела (база на податоци, серверска апликација и клиентска апликација) кои што потребно е соодветно посебно да се стартуваат за да апликацијата функционира правилно.

Овие три дела можат да се стартуваат на два начини и тоа со користење само на docker или рачно.

Доколку избереме апликацијата да ја стартуваме само со docker, трите дела ќе се стартуваат во три посебни docker контејнери кои би комуницирале помеѓу себе со цел правилна работа на апликацијата. Овој начин на стартување на апликацијата не е погоден за време на развој на апликацијата, поради тоа што не е возможно да се применат новите промени во реално време.

Чекори за стартување на апликацијата со користење на docker:

- Со помош на CLI поставете ја вашата локација да е иста со локацијата на проектот во вашиот фајлов систем
- cd src
- docker-compose up -d
- Апликацијата ќе е достапна на localhost:5000

Другиот начин за стартување на апликацијата е тоа да го направиме рачно. Овој начин е попогоден за користење за време на развој на апликацијата, поради тоа што е возможно да се применат новите промени во реално време.

При стартување на апликацијата на овој начин имаме можност повторно да искористиме docker доколку би сакале базата на податоци да ја стартуваме од docker контејнер.

Дополнително потребно е да поставиме и .env датотеки за да ги поставиме соодтвено околинските варијабли. Ова е потребно да се направи и за серверската и за клиентската апликација.

Пример за .env датотека за серверската апликација:

```
DATABASE_HOST=localhost
DATABASE_PORT=5432
DATABASE_USER=postgres
DATABASE_PASSWORD=admin
DATABASE_NAME=pulsebuy
MAIL_USER=noreply.pulsebuy@gmail.com
PASSWORD=xxxxxxxxxxxx
CLIENT_URL=http://localhost:5173
```

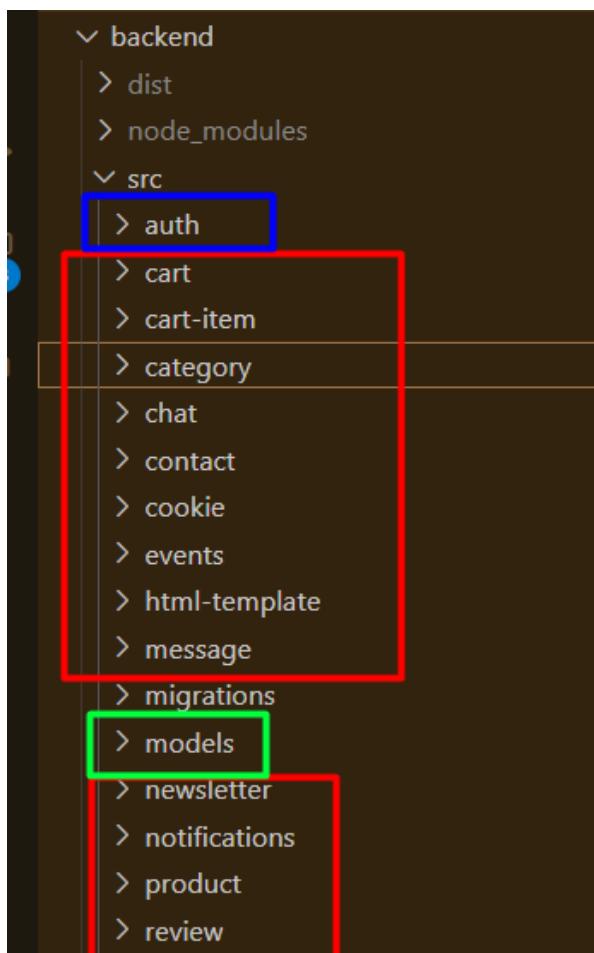
Пример за .env датотека за клиентската апликација:

```
VITE_API_URL=http://localhost:3000
```

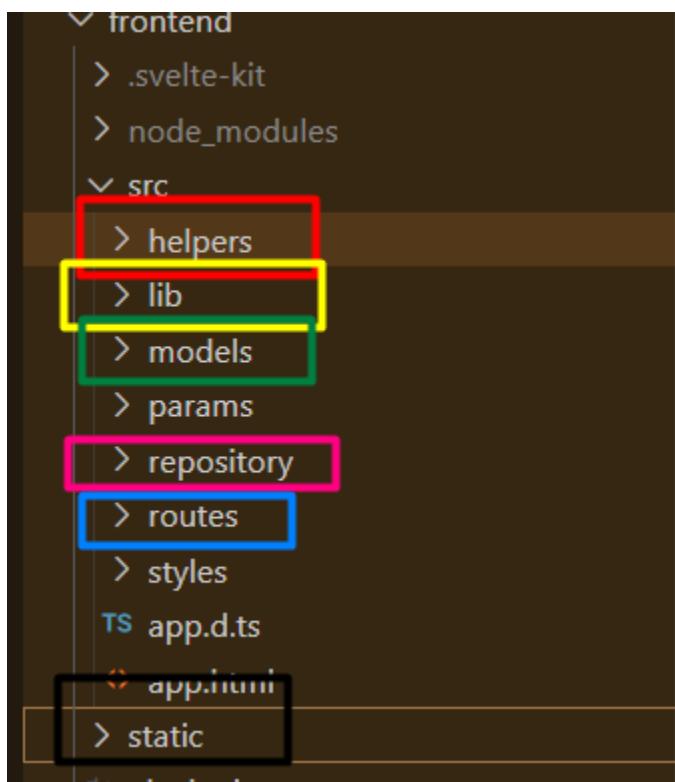
Чекори за рачно стартување на апликацијата:

- Со помош на CLI поставете ја вашата локација да е иста со локацијата на проектот во вашиот фајлов систем
- Стартување на базата на податоци, за овој чекор постојат два избори:
  - Стартување на PostgreSQL сервисот
  - Стартување на базата на податоци во docker контејнер:
    - cd src
    - docker-compose up -d pulsebuy\_db
    - cd ..
- Стартување на серверската апликација:
  - cd src/backend
  - Npm run start:dev
  - cd ..
- Стартување на клиентската апликација:
  - cd src/backend
  - Npm run dev
  - cd ..
- Апликацијата ќе е достапна на localhost:5173

## Структура на проектот



- Сервис потребен за автентикација, JWT токен стратегија за автентикација
- Модели за соодветните ентитети во базата
  - Репозиторија и сервиси за секој модел соодветно
    - Gateway за WebSocket комуникацијата, како и соодветен сервис



- Helper функции ко ги користиме низ проектот како на пример `isUserAdmin()`
- Стандандизирани изгледи низ проектот пример `footer`
- Модел на frontend за соодветните ентитети
- Репозиторија на frontend за соодветните контролери од backend
- Соодветни патеки во проектот, пропратени со изгледот, логиката.....
- Static фајлови за проектот, пример сликата на почетната страна

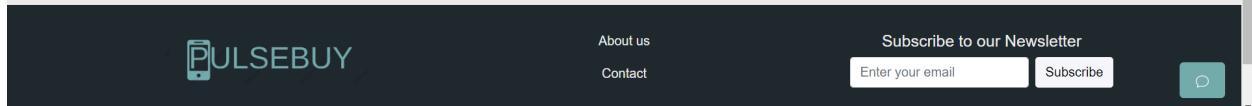
# Кориснички сценарија:

## 1. Регистрација и најава на системот

The screenshot shows a web browser window with the URL `localhost:5173/register`. The page has a dark header with the **PULSEBUY** logo. Below the header is a registration form titled "Register". The form fields are: First Name, Last Name, Email, Password, and Address. Each field has a placeholder text: "Enter your first name", "Enter your last name", "Enter your email", "Enter your password", and "Enter your address". A blue "Register" button is at the bottom, and a link "Already have an account? Login here" is below it.

The screenshot shows a web browser window with the URL `localhost:5173/login`. The page has a dark header with the **PULSEBUY** logo. Below the header is a login form titled "Login". The form fields are: Username and Your password. Each field has a placeholder text: "Enter your username" and "Enter your password". A blue "Log in" button is at the bottom, and a link "Don't have an account? Register here" is below it.

## 2. Пребарување и сортирање на производи и гледање на детали за производ

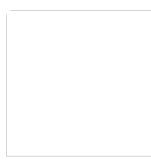


Login    localhost:5173/products

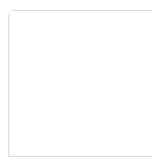
Addons Store Booking.com Facebook YouTube AliExpress

# PULSEBUY

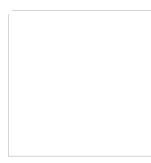
Home About Us Products



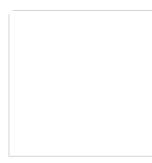
Телефони  
Телефони



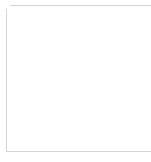
Лаптопи  
Лаптопи



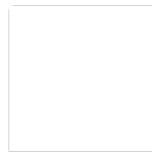
Галантерија  
Галантерија



Персонални компјутери  
Персонални компјутери



Тастатури  
Тастатури



Глувчиња  
Глувчиња



Сите производи  
Сите производи

Login    localhost:5173/products/category/1

Addons Store Booking.com Facebook YouTube AliExpress

# PULSEBUY

Home About Us Products

Пребарувај по име..

Најновото прво



Iphone 12

Iphone 12  
899.99MKD  
Out of stock



Iphone 13

Iphone 13  
949.99MKD  
Out of stock



Iphone 14

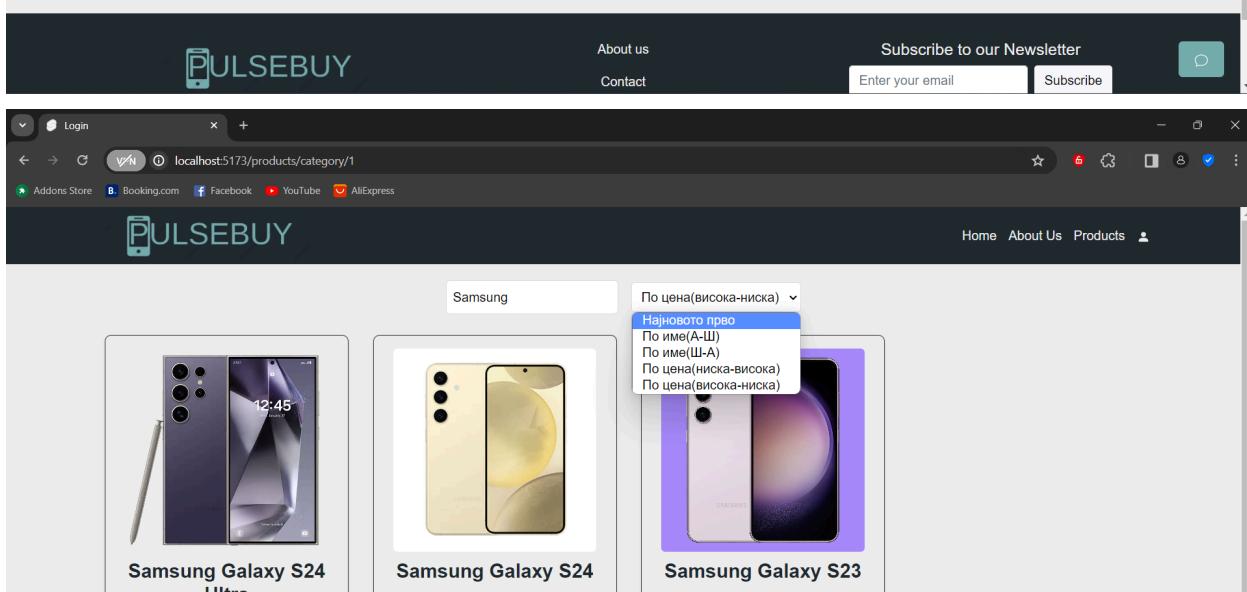
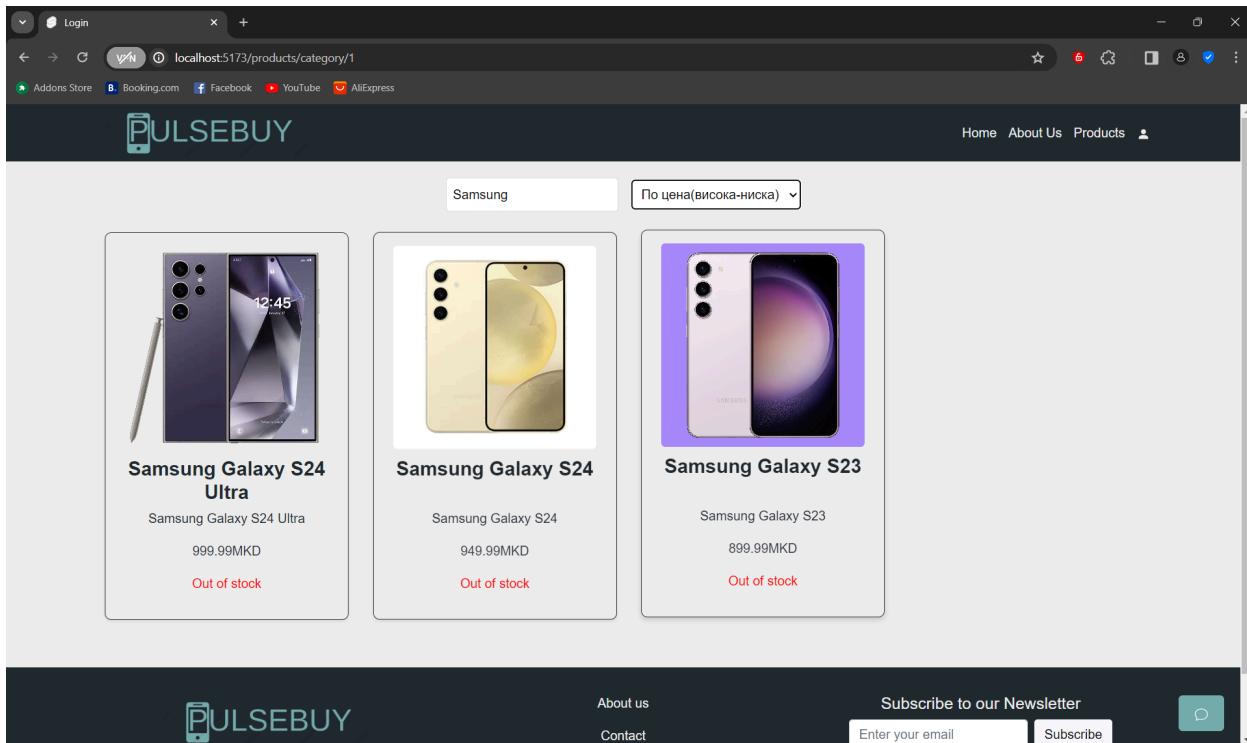
Iphone 14  
999.99MKD  
Out of stock



Samsung Galaxy S23

Samsung Galaxy S23  
899.99MKD  
Out of stock





3. Додавање на производи во кошничка и во листа на желби

localhost:5173/products/1

PULSEBUY

Iphone X

Category: Телефони

Iphone X

Price: \$799.99

2

Add to Cart

Reviews

What do you think of this product? Share your experience to help others.

What do you think of this product? 1 Add review

localhost:5173/cart

PULSEBUY

Your Cart

Product	Description	Quantity	Price	Total	Remove
Iphone X	Iphone X	2	\$799.99	\$1599.98	
Total Price \$1599.98					

Enter your address

Enter your city

Enter your postal code

Enter your country

Enter your phone number

Order Products

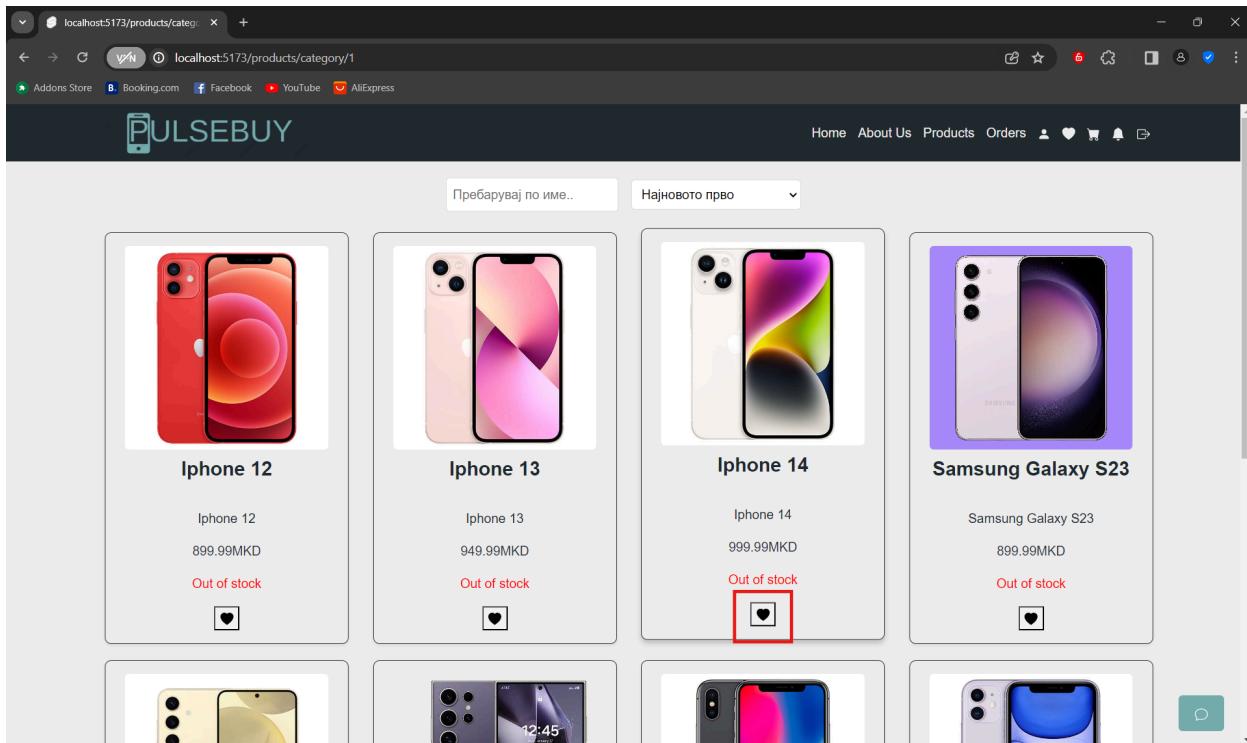
About us

Contact

Subscribe to our Newsletter

Enter your email

Subscribe



The screenshot shows the user's wishlist page. The page title is 'My Wishlist' and features a heart icon. The table lists one item: an iPhone 14, which is marked as 'Out of stock'. The 'Out of stock' status is highlighted with a red border around the heart icon.

4. Комплетирање на нарачка

The screenshots show a web application interface for a store named PULSEBUY. The top two screenshots illustrate the cart and order placement process, while the bottom screenshot shows the confirmation page for Order #1.

**Screenshot 1: Your Cart**

This screenshot shows the user's shopping cart. It includes a table with columns for Product, Description, Quantity, Price, Total, and Remove. A single item, an iPhone X, is listed with a quantity of 2, a price of \$799.99, and a total of \$1599.98. Below the table, there are input fields for address, city, zip code, country, and phone number. A blue "Order Products" button is at the bottom.

Product	Description	Quantity	Price	Total	Remove
Iphone X	Iphone X	2	\$799.99	\$1599.98	

**Screenshot 2: Order Products**

This screenshot shows the user's order summary for Order #1. It lists the same iPhone X item with a quantity of 2 and a total of \$1599.98.

Product	Description	Quantity	Price	Total
Iphone X	Iphone X	2	\$799.99	\$1599.98

**Screenshot 3: Order Confirmation**

This screenshot shows the confirmation page for Order #1. It displays the order details and a message indicating the order has been placed.

## 5. Добавување нотификации за работи во системот

The screenshot shows a web browser window with the URL [localhost:5173/orders](http://localhost:5173/orders). The page title is "PULSEBUY". The main content displays an order summary for "Order #1". The table includes columns for Product, Description, Quantity, and Price. One item listed is "Iphone X" at \$799.99 per unit, quantity 2, totaling \$1599.98. A success message box is overlaid on the page, stating: "Your order was successfully processed. Your total is 1599.98" on Jun 22, 2024, 11:30 AM, with a link to "Mark as read". The footer features the PulseBuy logo, links to "About us" and "Contact", and a newsletter sign-up form.

Product	Description	Quantity	Price
 Iphone X	Iphone X	2	\$799.99
			\$1599.98

Your order was successfully processed. Your total is 1599.98  
Jun 22, 2024, 11:30 AM  
[Mark as read](#)

About Us Products Orders     

Mark all as read

Your order was successfully processed. Your total is 1599.98  
Jun 22, 2024, 11:30 AM  
[Mark as read](#)

PULSEBUY About us Contact Subscribe to our Newsletter Enter your email  

## Order created External Inbox x

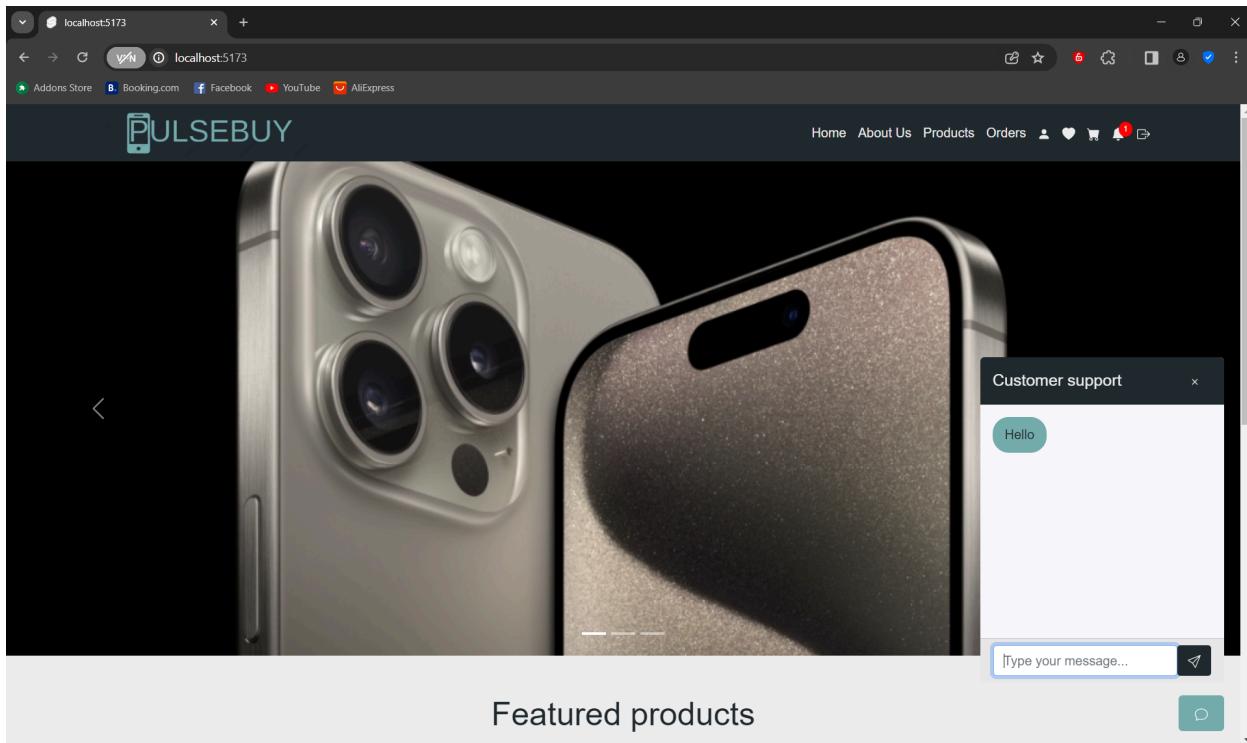


No Reply <noreply.cookitup@gmail.com>  
to me ▾

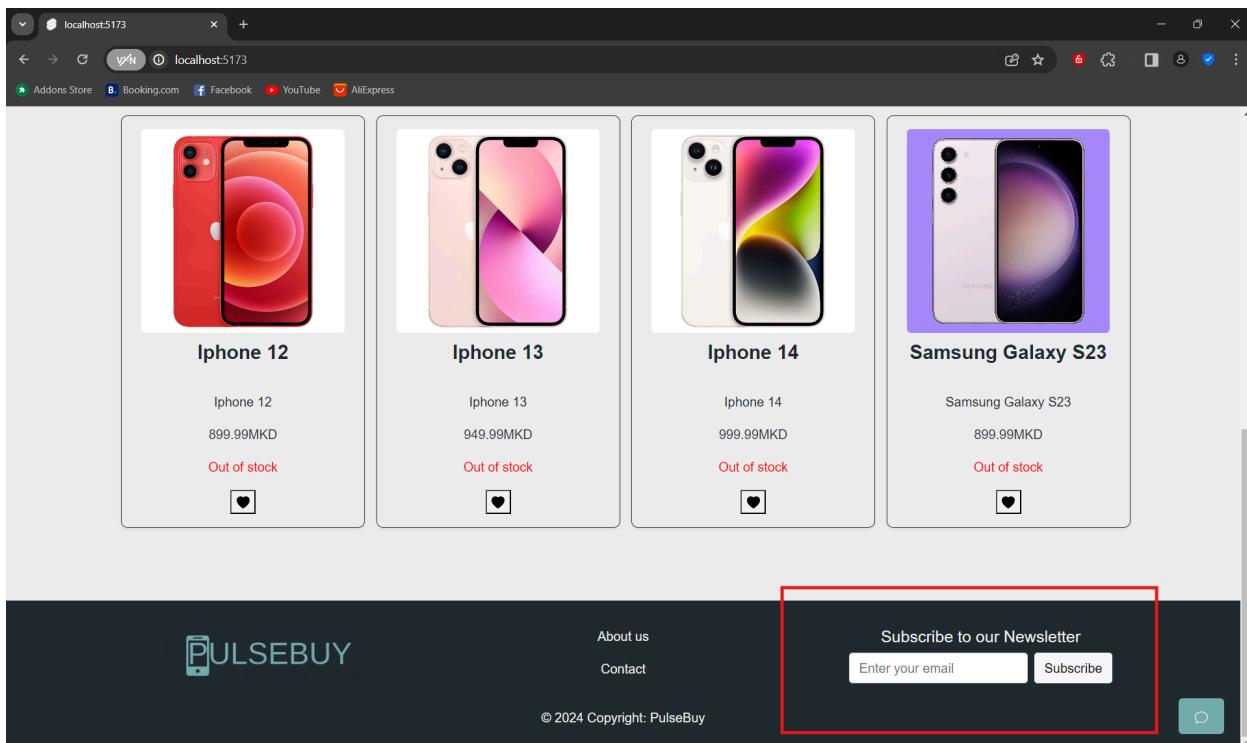
Your order was successfully processed. Your total is 60000

---

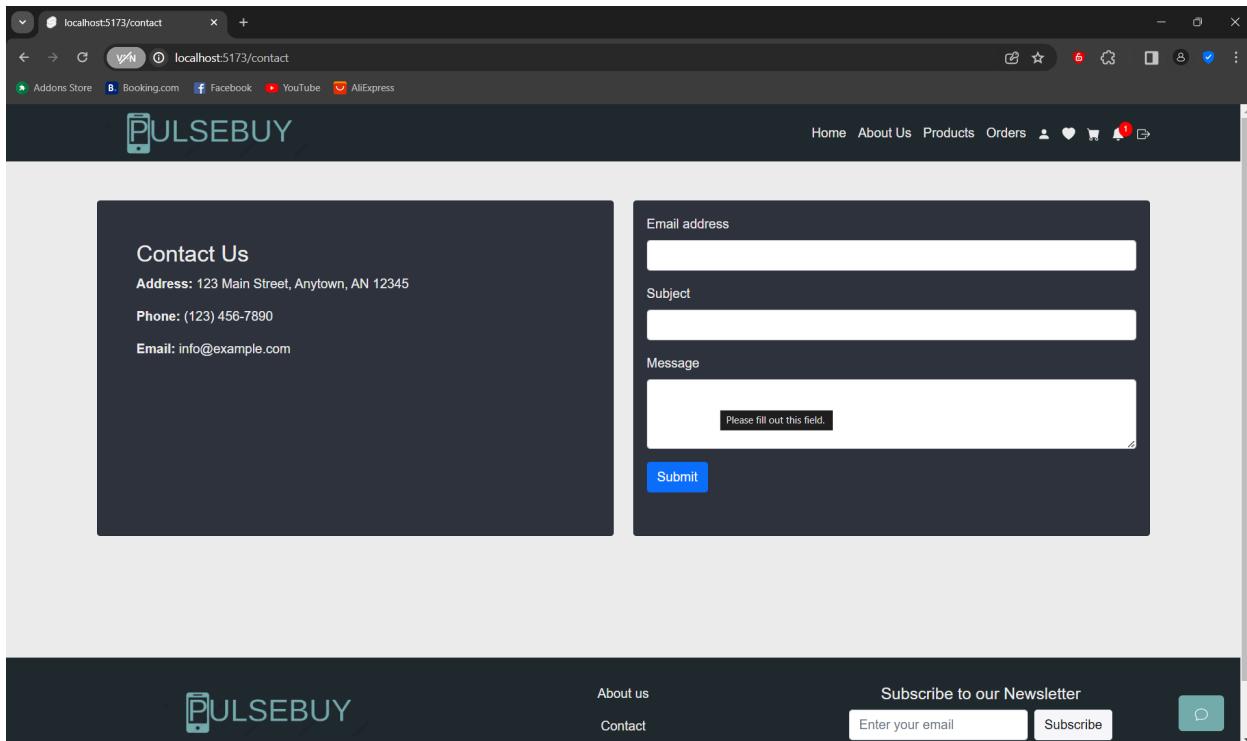
### 6. Контактирање на корисничка поддршка



## 7. Претплата на newsletter

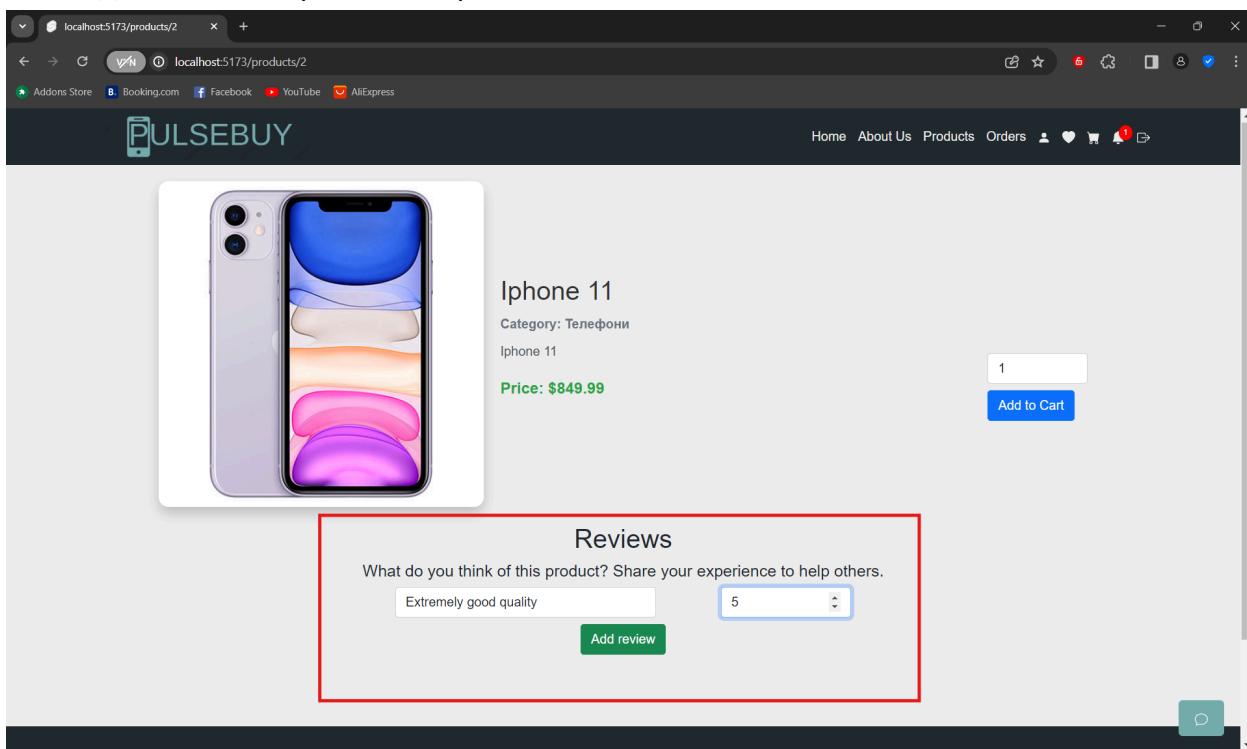


## 8. Контактирање на админите на апликацијата

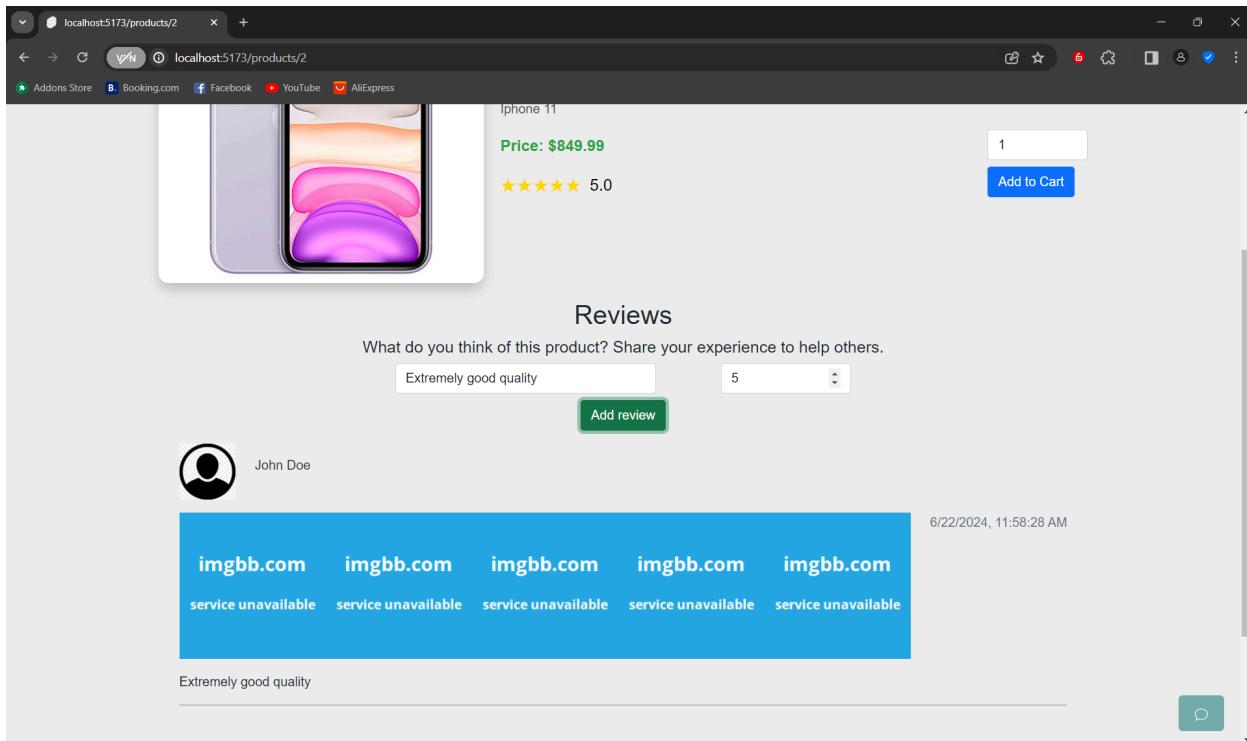


The screenshot shows the contact page of a website named PULSEBUY. The header includes a logo, navigation links for Home, About Us, Products, Orders, and user account options, and a notification icon with a '1'. The main content area has a dark background. On the left, there's a "Contact Us" section with address, phone number, and email information. On the right, there are input fields for Email address, Subject, and Message, with a note "Please fill out this field." and a blue "Submit" button.

## 9. Додавање на критика на производ



The screenshot shows the product details page for an iPhone 11. The top navigation bar and header are identical to the contact page. The product image is a light purple iPhone 11. To its right, the product name "Iphone 11" is displayed, along with its category "Category: Телефони", price "\$849.99", and a quantity selector set to "1". Below the product image is a red-bordered "Reviews" section containing a text input field ("Extremely good quality"), a rating dropdown set to "5", and a green "Add review" button. The bottom of the page features a dark footer bar with a small message icon.



10. Админ панел - менацирање со залихи

localhost:5173/admin/manage Stock Management

Product Name	Current Stock	Add Stock
Iphone 12	0	Add Stock
Iphone 13	0	Add Stock
Iphone 14	0	Add Stock
Samsung Galaxy S23	0	Add Stock
Samsung Galaxy S24	0	Add Stock
Samsung Galaxy S24 Ultra	0	Add Stock
Dell Inspiron	0	Add Stock
Dell Vostro	0	Add Stock
Dell Latitude	0	Add Stock
Dell XPS	0	Add Stock

localhost:5173/admin/manage Stock Management

Product Name	Current Stock	Add Stock
Iphone 12	Add Stock for Iphone 12 Amount: <input type="text" value="0"/> <button>Submit</button> <button>Discard</button>	
Iphone 13		Add Stock
Iphone 14		Add Stock
Samsung Galaxy S23	0	Add Stock
Samsung Galaxy S24	0	Add Stock
Samsung Galaxy S24 Ultra	0	Add Stock
Dell Inspiron	0	Add Stock
Dell Vostro	0	Add Stock
Dell Latitude	0	Add Stock
Dell XPS	0	Add Stock

11. Админ панел - менаџирање со корисници

The screenshot shows a browser window with the URL `localhost:5173/admin/users`. The page title is "Manage User Permissions". It displays a table with columns: #, Username, Email, and Admin Permission. The table contains seven rows:

#	Username	Email	Admin Permission
1	testuser1	testuser1@gmail.com	<input type="checkbox"/> Has admin permissions
2	testuser2	testuser2@gmail.com	<input type="checkbox"/> Has admin permissions
3	testuser3	testuser3@gmail.com	<input type="checkbox"/> Has admin permissions
4	testuser4	testuser4@gmail.com	<input type="checkbox"/> Has admin permissions
5	testadmin1	testadmin1@gmail.com	<input checked="" type="checkbox"/> Has admin permissions
7	john.doe	john.doe@test.com	<input type="checkbox"/> Has admin permissions

The footer of the page includes the PULSEBUY logo, links for "About us" and "Contact", and a newsletter subscription form with fields for "Enter your email" and "Subscribe".

## 12. Админ панел - додавање на распродажба

The screenshot shows a browser window with the URL `localhost:5173/admin/add-sale`. The page title is "Add New Sale". The form has the following fields:

- Product: Iphone 12
- Sale Percentage: 20
- Date From: 06/23/2024
- Date To: 06/29/2024

A blue "Add Sale" button is located at the bottom of the form.

The screenshot shows the 'Sales' section of the PULSEBUY admin panel. At the top, there are two tabs: 'Active Sales' (highlighted) and 'Inactive Sales'. Under 'Active Sales', there is a card for an offer: 'Iphone 12 - 20% off' from '2024-06-23 - 2024-06-29'. Below the tabs is a blue button labeled 'Add a new sale'.

### 13. Админ панел - додавање на производ/категорија

The screenshot shows the 'Add New Product' form. It includes fields for Product Name (input field), Product Price (input field with value '0'), Product Description (text area), Product Image (input field), Product Category (dropdown menu), Number in stock (input field with value '0'), and an 'Add Product' button at the bottom.

A screenshot of a web browser window titled "Add Category". The URL in the address bar is "localhost:5173/admin/add-category". The page has a dark header with the "PULSEBUY" logo. Below the header is a modal dialog titled "Add New Category". The modal contains two input fields: "Category Name:" and "Category Description:", both currently empty. At the bottom of the modal is a blue button labeled "Add Category".

A screenshot of a web browser window titled "Add Category" (though the URL is "localhost:5173/admin/customer-support"). The page has a dark header with the "PULSEBUY" logo. Below the header is a sidebar labeled "Chats" with a single item "John Doe". The main area shows a conversation with "John Doe": "Hello" and "Hello John, how can i help you". At the bottom is a message input field with placeholder "Type your message..." and a send icon.

**14. Админ панел - враќање на пораки на корисничка поддршка**

The screenshot shows the same interface as above, but with a larger view of the chat window. The message "Hello John, how can i help you" is highlighted in a callout bubble.

## Поважни features:

### Повратни информации за корисниците со помош на Toastr Notifications

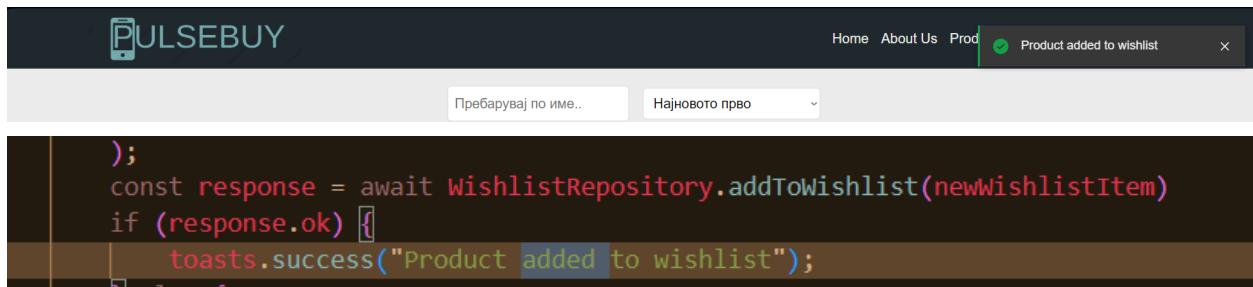
За да го подобриме корисничкото искуство и да обезбедиме непосредни повратни информации до корисниците, нашата апликација користи Toastr notifications. Toastr е JavaScript библиотека која ни овозможува да покажеме известувања за успехот или неуспехот на корисничките акции - Toastr известувања.

Toastr известувањата се мали, ненаметливи пораки кои се појавуваат привремено на еcranот. Овие известувања се дизајнирани да обезбедат визуелни повратни информации за корисниците, потврдувајќи дека нивните дејства се успешно завршени или предупредувајќи ги за грешки на кои треба да им се обрне внимание. Овој пристап обезбедува корисниците веднаш да бидат известени за важни настани без да се наруши нивната интеракција со апликацијата. Клучните карактеристики на Toastr се следниве:

- Неблокирачки - известувањата не се мешаат во акциите на корисникот, што не им попречува во нивното целокупно искуство на апликацијата
- Привремени - известувањата автоматски исчезнуваат по одредено време, иако корисниците можат и рачно да ги тргнат
- Контекстуални - различните видови на известувања (за успех, за грешка, за предупредување) се разликуваат по бои, што им олеснува на корисниците да ја разберат природата на пораката на прв поглед, дури и без да ја прочитаат содржината

Типичните сценарија каде што се користат toastr нотификациите во нашата апликација се следните:

- Успешни дејства - кога корисникот успешно ќе заврши дејство, како што е зачувување на податоци или пополнување на некоја форма, се прикажува известувањето за успех

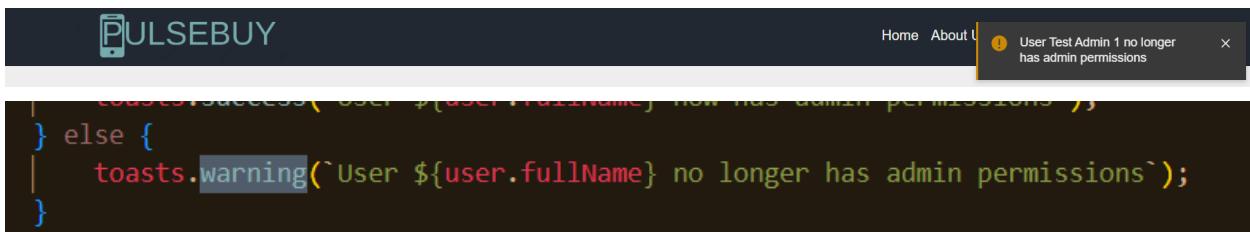


- Справување со грешки - кога некоја корисничка акција ќе предизвика грешка во апликацијата, односно нема да заврши до крај, на пример тоа би можеле да бидат погрешно внесени податоци, или некој неисполнет услов, се известува корисникот за проблемот



```
    } else {  
      const errorMessage = await response.json();  
      toasts.error(errorMessage.message);  
    }  
  }  
}
```

- Предупредување - овие известувања ги користиме за предупредување на корисниците за потенцијални проблеми



Како заклучок, toastr нотификациите се витална компонента на нашиот систем. Тие ни обезбедуваат неколку важни придобивки, како што се:

- Подобреното корисничко искуство - корисниците добиваат повратни информации веднаш што ја подобрува интеракцијата со апликацијата
- Јасна комуникација со корисниците - контекстуалната природа на нотификациите гарантира дека брзо и лесно корисниците ја разбираат пораката
- Намалената фрустрација - со обезбедувањето навремени повратни информации, помала е веројатноста дека корисниците би биле збунети или фрустрирани од одговорот на апликацијата на нивните постапки

## Закажани извршувања на задачи(Scheduled Task Execution)

Закажните извршувања на задачи се однесува на практиката за дефинирање и извршување на задачи во предефинирани времиња или интервали. Тие се дефинираат кога ни треба автоматско извршување на некои повторливи акции во апликацијата.

Во нашата апликација, имаме повеќе вакви извршувања на задачи. Ги дефинираме како Cron Jobs, користејќи CronExpression, со кои го специфицираме точното време на задачата. Нив ги сместуваме во сервис task.service.ts кој се наоѓа во TasksModule. Cron jobs во нашата апликација се следниве:

- **handleUnsentContacts** - праќање на непратените контакти во апликацијата, имено во нашата апликација постои дел "Contact us", каде што секој може да напише наслов, текст и своја e-mail адреса и да исконтактира со админите на страната. Бидејќи не знаеме колку вакви може да има во исто/приближно време, сите контакти гичуваме во база со статус(пратени/непратени) и на секој 1 час, ги

зимаме сите непратени контакти и праќаме mail на админите. Потоа се праќа и mail назад до човекот што пробува да контактира дека mailот е успешно пратен.

```
@Cron(CronExpression.EVERY_HOUR)
async handleUnsentContacts() {
    this.logger.debug('Sending unsent contacts');
    const contacts = await this.contactService.findAllNotSent();
    this.logger.debug(
        'Number of unsent contacts: ' + contacts.length.toString(),
    );
    for (const contact of contacts) {
        await this.sendContactToDevelopers(contact);
        this.logger.debug(
            'Contact with id: ' +
            contact.id.toString() +
            ' sent. The email is: ' +
            contact.email,
        );
        contact.is_sent = true;
        await this.contactService.contactSent(contact);
        await this.mailerService.sendMail({
            to: contact.email,
            subject: 'Re: ' + contact.subject,
            text: 'Thank you for contacting us. We will get back to you as
soon as possible.',
        });
    }
}
```

- **handleNewsletter** - праќање на билтен(newsletter), во нашата апликација има дел каде што секој може да ја остави својата e-mail адреса и така да добива известувања за новости што се случуваат во апликацијата. Така нашиот билтен се праќа еднаш дневно на пладне, со што се праќаат информации до сите преплатници за новите продукти што се додадени во апликацијата

```
@Cron(CronExpression.EVERY_DAY_AT_NOON)
async handleNewsletter() {
    this.logger.debug('Sending newsletter');
    const subscribers = await this.newsletterService.findAll();
    this.logger.debug(
        'Number of subscribers: ' + subscribers.length.toString(),
    );
    for (const subscriber of subscribers) {
        await this.mailerService.sendMail({
            to: subscriber.email,
            subject: 'Newsletter',
        });
    }
}
```

- ```

        text: await this.generateNewsletter(subscriber),
    ) );
}
}

- handleExpiredSales - менување на статусот на попустите во апликацијата - ова се случува секој ден во 23:59, односно попустите што траат до тој ден, следниот ден да бидат во неактивен статус

```

```

@Cron( "0 59 23 * * *")
async handleExpiredSales() {
    this.logger.debug('Expiring sales');
    const pastSales = await
this.saleService.findAllDateToInThePastAndActive();
    this.logger.debug(
        'Number of sales: ' + pastSales.length.toString(),
    );
    await this.saleService.setStatusToInactive(pastSales);
}

```

## Менаџирање со регистрирани и нерегистрирани корисници

Во нашата апликација, менаџираме со нерегистрираните корисници со користење на колачиња. Користејќи колачиња за складирање на идентификатори на сесии, му дозволуваме на серверот да ги препознае корисниците кои се враќаат. Тоа го правиме со доделување UUID на нерегистрирани корисници складирани во колаче, со време на истекување за управување со времетраењето на сесијата. Во контекст на нашата апликација, тоа го правиме за интеракција со Customer support chat, на следниов начин:

- Нерегистрирани корисници: Кога нерегистриран корисник комуницира со Customer support, уникатен UUID се генерира и се складира во колаче. Ова колаче се користи за идентификување на корисникот во следните интеракции додека не истече по 3 дена од неговото креирање. На frontend тоа изгледа така:

```

export async function getSessionCookie() {
    let session = localStorage.getItem('session')
    if (!session) {
        return null
    }
    let expiry = JSON.parse(session)["expiry"]
    if (!expiry) {
        return null
    }
    if (new Date(expiry) < new Date()) {
        localStorage.removeItem('session')
        return null
    }
}

```

```

        }
        return localStorage.getItem('session')
    }

export async function setUpSessionCookie() {
    const session = await getSessionCookie();
    if (!session) {
        const response = await generateCookie();
        if (response.ok) {
            const data = await response.json();
            localStorage.setItem('session', JSON.stringify(data));
        }
    }
}

export async function generateCookie() {
    return await interceptedFetch('/cookie', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({}),
    });
}

```

Додека пак на backend, се генерира единствен UUID на следниов начин:

```

@Controller('cookie')
export class CookieController {
    @Post()
    createCookie() {
        return {
            value: uuidv4(),
            expiry: new Date(Date.now() + 3 * 24 * 60 * 60 * 1000)
        }
    }
}

```

- Регистрирани корисници: Кога корисникот се најавува, системот за Customer support chat го користи неговиот кориснички ID за идентификација, обезбедувајќи беспрекорно и поперсонализирано искуство.

## Настани

По извршување на одредени акции во нашата апликација се испалуваат настани кои што го известуваат останатиот дел од апликацијата дека одредена акција била извршена. Овие настани како што понатаму ќе биде објаснето се клучни доколку сакаме клиентската страна на апликација да се ажурира во реално време и корисниците секогаша да ги

гледаат најновите информации и дополнително со помош на „фаќање“ на соодветните настани ние испраќаме и нотификацији кон корисниците.

Класите кои што треба да ги претставуваат настаните во апликацијата се организирани во една хиерархиска структура на класи и подкласи. На врвот на оваа хиерархија е класата PulseBuyEvent, која што всушност е и абстракна класа. Оваа класа содржи информации за идентификаторот за агрегатот на кој што се однесува тој настан, името на тој настан и функција која враќа стринг во формат: `\${this.getAggregateType()}.\${this.eventType}`. Оваа функција ни кажува кој настан се испалил и за кој агрегат се испалил тој настан. Функцијата `getAggregateType()` е апстракна функција.

PulseBuyEvent класата потоа ја наследуваат класите кои што ги претставуваат настаните на ниво на агрегат. Овие класи прават `override` на функцијата `getAggregateType()` и наведуваат за кој агрегат се однесува таа класа и сите класи кои што ќе наследат од неа. Пример за ваква класа е OrderEvent, која претставува суперкласа за сите настани од агрегатот Order. Класите пак што наследуваат од овие класи, потребно е само да специфицираат кои податоци во себе ќе ги содржат освен оние наведени во PulseBuy абстрактната класа. Пример за овој вид на класа е OrderCreatedEvent.

Настаните се испалуваат со користење на `EventService` сервисот и неговата функција `emitEvent()` која како параметар го прима настанот кој што треба да се испали:

```
emitEvent(event: PulseBuyEvent) {  
  
    this.eventEmitter.emit(` ${PULSE_BUY_EVENT} .${this.moduleName} .${event.eventType}` , event);  
}
```

За испалување на настаните во оваа функција го користиме модулот `EventEmitter2` и неговата функција `emit`, која како параметри го бара името на настанот и самиот настан. Како име на настанот испраќаме дека се работи за настан од `PulseBuy`, за конкретен агрегат и на крај името на самиот настан.

Секој агрегат кој што треба да испалува настани го наследува овој сервис и го наведува неговото име во конструкторот.

```
@Injectable()  
export class SaleEventsService extends EventService {  
    constructor(eventEmitter: EventEmitter2) {  
        super(eventEmitter, SALE_MODULE_NAME);  
    }  
}
```

## WebSocket комуникација

Со цел информациите прикажани на клиентската апликација да се ажурираат во реално време, имплементираме WebSocket комуникација помеѓу серверската и клиентската апликација користејќи го Socket.io пакетот во серверската апликација.

### Серверска страна

Логиката околу конектирање, дисконектирање, еmitување на пораки преку WebSocket-от е сместена во SocketService сервисот.

Со цел подобро справување со тоа кои пораки треба да ги добие еден корисник ја искористивме функционалноста на додавање на корисникот во одредени „соби“, врз основа на тоа за кои „topics“ сака да прима информации во реално време. На пример, една соба може да биде само за една конкретна комуникација со одделот за комуникација со клиенти и во оваа соба би биле додадени корисникот кој ја иницирал комуникацијата и можеби еден од администраторите на апликацијата и со тоа само тие двајца би „слушале“ за настани испалени за таа инстанца. Друг пример е кога даден корисник се наоѓа на страната која наликува на каталог на производи, тогаш тој корисник би бил во собата која се однесува на целиот агрегат Product и би слушал тој корисник за сите настани испалени за било кој производ додаден во системот.

Ова во SocketService е имплементирано во функцијата broadcastMessageToRoom(roomId: string, event: string, data: any): void:

```
// Method to broadcast a message to all clients in a specific room
broadcastMessageToRoom(roomId: string, event: string, data: any): void {
    this.connectedClients.forEach((socket) => {
        if (socket.rooms.has(roomId)) {
            socket.emit(event, data);
        }
    });
}
```

Постои и SocketGateway кој што едноставно само ги повикува методите од SocketService.

Со цел да можеме да ја известуваме клиентската апликацијата за испалените настани во апликацијата, го креирајме PulseBuyEventListener, кој што слуша дали бил испален било кој PulseBuy настаните. Доколку бил испален некој од овие настани, функцијата handlePulseBuyEvent(event: PulseBuyEvent): void испраќа порака со помош на SocketGateway до собата за агрегатот на кој што се однесува настанот и до собата намената за конкретната инстанца од тој агрегатот на кој што се однесува настанот:

```
@OnEvent(`${PULSE_BUY_EVENT}.**`, {async: true})
handlePulseBuyEvent(event: PulseBuyEvent): void {
    const specificAggregateRoomId = `${event.getAggregateType()} / ${event.id}`;
```

```
const aggregateRoomId = event.getAggregateType();
this.socketGateway.sendEventToRoom(specificAggregateRoomId,
event.getEventType(), event);
this.socketGateway.sendEventToRoom(aggregateRoomId, event.getEventType(),
event);
}
```

## Клиентска страна

На клиентска страна беше потребно да ја конфигурираме комуникацијата со WebSocket-от од серверската страна и со таа цел го додадовме webSocketConnection.ts фајлот каде што се декларира ioClient-от од Socket.io.

Во компонентите каде што е потребно да имаме ажурирање во реално време го правиме следново:

- Во onMount функцијата се регистрираме во сите соби од кои што сакаме да бидеме известувани за испалени настани од апликацијата. На пример регистрирање во собата за даден производ:
  - const roomId = `Product/\${\$page.params.id}`;
  - 
  - io.emit("joinRoom", roomId);
- Во onMount функцијата специфицираме како се справуваме со добиените настани. На пример по ажурирање на дадениот производ, сакаме да направиме refetch на информациите за тој производ:
  - io.on("Product.ProductUpdatedEvent", () => {
  - fetchProduct()
  - })
- Ви функцијата onDestroy испраќаме порака дека ги напуштаме собите на кои сме се регистрирале во onMount функцијата. Ова е важно за да не слушаме за настани за кои не сме заинтересирани веќе:

```
onDestroy(async () => {
    const roomId = `Product/${$page.params.id}`
    io.emit("leaveRoom", roomId);
});
```

## Нотификацији

Во нашиот систем постојат два вида на нотификацији и тоа: Емаил Нотификацији и Нотификацији кои се прикаживаат во самата апликација, т.е. In App Notifications. И двата вида ќе бидат во детали објаснети во продолжение.

Со цел полесно и поефикасно рендерирање на нотификациите, користевме таканаречени html темплејти. Овие темплејти не претставуваат ништо друго освен истанци од класата HtmlTemplates кои се чуваат во нашата база на податоци и за секоја инстанца ние чуваме

ид, за кој настан кој се испалил се однесува темплејтот, наслов, содржина на темплејтот и тип (постојат два типа на темплејти, по еден за секој вид на нотификација). Најбитен атрибут на овие инстанци е содржината на темплејтот која претставува темплејт напишан во HandleBars јазикот за темплејтот кој ни овозможува да вметнеме одредени изрази во нашиот темплејт кои подоцна, кога ќе се генерира соодветната нотификација ќе бидат заменети со вистниски вредности.

Еден пример за html темплејт е темплејтот за настанот ProductUpdatedEvent кој ја има следнава содржина: 'Product: {{event.name}} has been updated'. При евалуација, со помош на функцијата compile од модулот handlebars која како влез ќе ја прими содржината на темплејтот и во случајов инстанца од настанот ProductUpdatedEvent ќе се изгенерира содржината на нотификацијата за соодветниот испален настан.

Со цел процесирање на нотификациите и правилна евалуација на html темплејтите го искостивме шаблонот за дизајн на софтвер, Single chain of responsibility. Овој шаблон го имплементираме на тој начин што додадовме една абстрактна класа за EventNotificationService сервисот кој има два абстрактни метода:

- applicableTo(event: PulseBuyEvent, channel: HtmlTemplateContext):Promise<boolean> кој одредува дали конкретната имплементација на овој сервис е одговорна за процесирање на нотификацијата за испратениот настан во соодветниот канал (кој означува тип на нотификација)
- notificationRecipients(event: PulseBuyEvent): Promise<User[]> кој одредува кој корисник треба да ја прими нотификацијата за испратениот настан

Дополнително овој сервис содржи и уште еден метод кој има default-на имплементација и тој метод е buildNotification(event: PulseBuyEvent, channel: HtmlTemplateContext, user: User): Promise<EventNotification> кој ја генерира крајната нотификација преку евалуација на пронајдениот темплејт за испратениот настан и канал.

Дополнително додадовме и default-на имплементација на овој сервис и го нарековме DefaultEventNotificationService. Овој сервис е способен да ја испроцесира нотификацијата за било кој настан кој е испален, а за кој има соодветен html темплејт за наведениот канал. Исто така, нотификациите испроцесирани од овој сервис би се испратиле на сите регистрирани корисници во системот.

Понекогаш за нотификациите за одредени настани во апликацијата ни е потребно поспецифична логика од онаа во default-ната имплементација, како на пример за настанот OrderCreatedEvent кога сакаме да го известиме дека нарачката е успешно креирана само корисникот кој ја направил нарачката. Поради оваа причина можеме да креираме бесконечно многу имплементации на EventNotificationService сервисот.

Во компонентите кои се одговорни за „фаќање“ на испалените настани од апликацијата се додава како зависност листа од EventNotificationService сервиси, која ќе биде популирана од сите регистрирани имплементација на EventNotificationService сервисот од стана на Nest.js рамката за развој. Како и сите други сервиси потребно е да се наведе и овој сервис

во providers листата на соодветниот модул, но поради тоа што овој сервис има повеќе имплементации и нас ни е потребно да можеме да ги инјектираме како листа сите, потребно беше да додадеме специфичен провејдер за овој сервис:

```
{  
    provide: 'eventNotificationServices',  
    useFactory: (  
        orderCreatedEventNotificationService:  
OrderCreatedEventNotificationService,  
        wishlistEventNotificationService: WishlistEventNotificationService,  
        defaultEventNotificationService: DefaultEventNotificationService,  
    ) => [orderCreatedEventNotificationService,  
wishlistEventNotificationService, defaultEventNotificationService],  
    inject: [OrderCreatedEventNotificationService,  
WishlistEventNotificationService, DefaultEventNotificationService],  
}
```

Во компонентите каде што ги користиме овие сервиси, најчесто ги изминуваме сите сервиси се додека не најдеме еден, чиј applicableTo метод враќа true. И токму поради оваа причина овој провајдер е од огромно значење затоа што тута го специфицираме редоследот по кој ќе бидат проверени конкретните имплементации на сервисот, па така би требало прво да ги провериме поспецифичните имплементации, а default-ната имплементација да ја провериме последна.

Во продолжение ќе бидат објаснето како се процесираат во детали двата вида на нотификацији:

## Email Notifications

Овие нотификацијии се испраќаат на корисниците на мејл адресите со кои тие се имаат регистрирано корисниците. Со помош на овие нотификацијии, корисниците ќе бидат во тек со статусот на обележаните производи, нивните нарачки итн. додека тие нема да бидат најавени на апликацијата.

Овие нотификацијии се процесираат во EmailNotificationListener комонентата која е одговорна за „фаќање“ на сите испалени настани во апликацијата, со цел да се провери дали некој од регистрираните сервиси може да изгенерира нотификација за испалениот настан. Доколку постои таков сервис, тој ќе ја врати првин информацијата за корисниците кои треба да ја добијат нотификација на нивната email адреса, а потоа за секој таков корисник ќе се испрати мејл порака со соодветен наслов и содржина, кои ќе ги содржи ново-генерираната нотификација од страна на соодветниот EventNotificationService.

## In App Notifications

Овие нотификацијии се испраќаат на корисниците во рамките на клиент апликацијата. Со помош на овие нотификацијии, корисниците ќе бидат во тек со статусот на обележаните производи, нивните нарачки итн. додека тие се најавени на апликацијата.

За овие нотификации постои InAppNotificationsListener компонента, која на сличен начин како и EmailNotificationListener е одговорна за „фаќање“ на сите испалени настани во апликацијата и потоа генерирање на нотификации за соодветните корисници, но тута наместо да се испрати мејл порака, ново-генерираната нотификација се додава во NotificationManager инстанцата за соодветниот корисник.

За секој корисник, при негова регистрација се креира инстанца од класата NotificationManager во која се чуваат In App Notifications за соодветниот корисник. За секој корисник се чуваат ограничен број на ваков вид на нотификации, а кога ќе се надмине овој број се отстранува најодамна додадената нотификација. Дополнително за секој ваков вид на нотификација се чува статус кој означува дали конкретната нотификација е означена како прочитана од страна на корисникот или не.

Овие нотификации на клиент апликацијата се прикажуваат во реално време како што се додаваат во NotificationManager инстанцата за соодветниот корисник со помош на WebSocket комуникацијата, а дополнително корисникот во секое време може да ги прегледа сите нотификации што се зачувани во неговиот NotificationManager и да им го промени нивниот статус.