

Sprawozdanie z projektu BitShop

Miłosz Junak

Szczepan Rzeszutek

Karol Sewiło

Julia Smerdel

Technologie	3
Opis	3
Baza danych	4
Frontend	6
Funkcjonalności	9
Dodawanie produktu do koszyka	9
Funkcja updateProductQuantity	9
Funkcja addToCart	9
Dodawanie nowego produktu	10
Funkcja handleSubmit	10
Funkcja dodająca produkt po stronie serwera	11
Usuwanie produktu	12
Funkcja delete po stronie serwera	12
Funkcja removeProduct	12
Przyciski sortowania i filtrowania produktów	13
Kod Przycisków w komponencie Product	13
Funkcja matchProducts	13
Funkcja zwracająca przefiltrowane produkty po stronie serwera	13
Funkcja sortująca produkty po stronie serwera	14
Logowanie i rejestracja	15
Funkcja handleLogin	15
Funkcja handleSingup	16
Wylogowywanie użytkownika po stronie serwera	17
Funkcja po stronie serwera, która zwraca aktualnie zalogowanego użytkownika	17
Logowanie użytkownika po stronie serwera	18
Rejestracja po stronie serwera	19
Funkcje odpowiedzialne za haszowanie hasła i weryfikowanie go	20
Funkcja getCurrentUser	20
Funkcja isLoggedIn	21
Kupowanie produktu i koszyk	22
Funkcja kupowania po stronie serwera	22
Funkcja obsługująca kupowanie	23
Funkcja po stronie serwera, która zwraca koszyk dla danego klienta	24
Funkcja pobierająca przedmioty z koszyka	25

Technologie

Tematem projektu jest sklep internetowy „BitShop”.

Do zrobienia bazy danych wykorzystane zostało MongoDB. Inne użyte technologie to Atlas, React, NodeJS oraz Javascript.

Opis

Strona internetowa umożliwia zalogowanym użytkownikom kupowanie elektroniki.

Po zalogowaniu klient jest w stanie dodać wybrany produkt do koszyka, klikając przycisk “Buy”. Następnie produkt pojawia się w koszyku, gdzie jest możliwość zakupienia go przyciskiem “Purchase”.

Zakup jest możliwy wyłącznie wtedy, jeśli produkt jest dostępny (jeśli `product_quantity` jest większe od 0). W przeciwnym wypadku przycisk “Buy” jest niewidoczny.

Nie dodano możliwości usuwania produktów z koszyka.

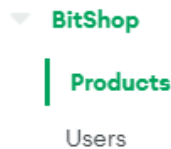
Przy dodawaniu produktów do koszyka, są one rezerwowane (zmniejsza się `product_quantity`).

Możliwe jest dodawanie produktów bezpośrednio na stronie (“Add product” w zakładce “Products”) wypełniając przygotowany formularz.

Produkty można usunąć z bazy klikając przycisk “Remove”.

Baza danych

Posłużyliśmy się mongoDB Atlas, aby uzyskać chmurową i łatwą w użytkowaniu bazę danych.

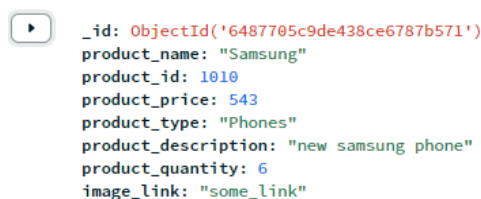


Kolekcja Products:

Każdy produkt posiada nazwę (product_name), id (product_id), cenę (product_price), typ (product_type), opis (product_description), ilość (product_quantity) oraz link do zdjęcia (image_link).

Oto przykładowy produkt:

W bazie



W formacie JSON

```
{
  "_id": {
    "$oid": "6487706b2b41bb45799bf559"
  },
  "product_name": "Samsung",
  "product_id": {
    "$numberInt": "1011"
  },
  "product_price": {
    "$numberInt": "543"
  },
  "product_type": "Phones",
  "product_description": "new samsung phone",
  "product_quantity": {
    "$numberInt": "9"
  },
  "image_link": "some_link"
}
```

Kolekcja Users:

Każdy użytkownik posiada nazwę (name), email (email), zahaszkowane hasło (password), koszyk (basket) oraz historię transakcji (history).

Oto przykładowy użytkownik:

W bazie:

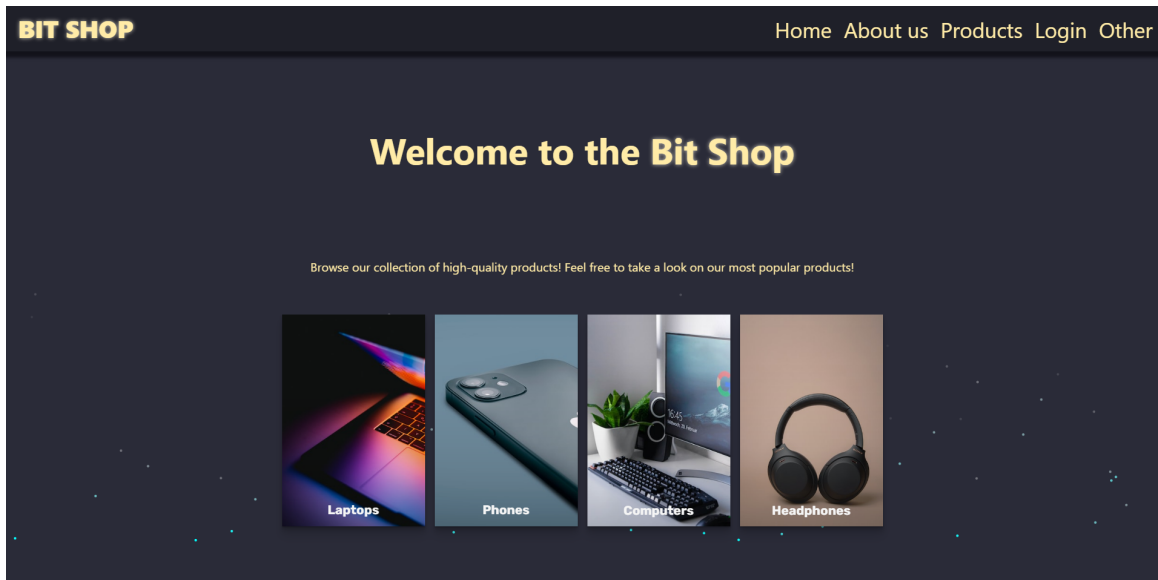
```
▶ {
  "_id": ObjectId('64563bc426d80cbb5c12dc62'),
  "name": "admin",
  "email": "admin@gmail.com",
  "password": "$2b$10$0BX7zx6zcW6TByHL/cm0v.1Je.40R4P9wuCkLpNJ7VmlurTjedYUW",
  "basket": Array
    ▶ 0: Object
    ▶ 1: Object
  "history": Array
    ▶ 0: Object
    ▶ 1: Object
    ▶ 2: Object
    ▶ 3: Object
}
```

W formacie JSON:

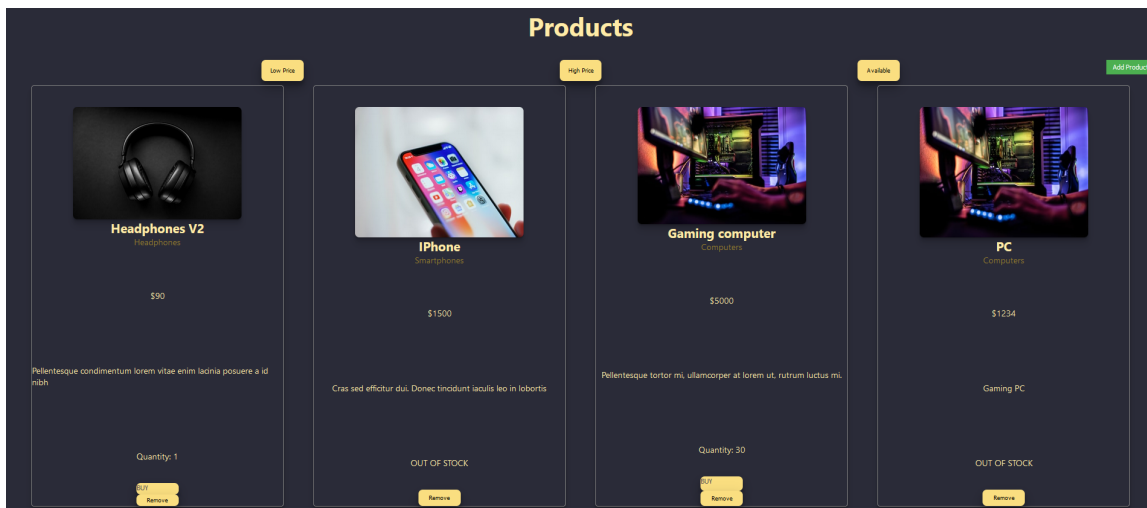
```
{
  "_id": {
    "$oid": "64563bc426d80cbb5c12dc62"
  },
  "name": "admin",
  "email": "admin@gmail.com",
  "password": "$2b$10$0BX7zx6zcW6TByHL/cm0v.1Je.40R4P9wuCkLpNJ7VmlurTjedYUW",
  "basket": [
    {
      "_id": {
        "$oid": "6454f23f699573147f166fc1"
      },
      "image_link": "1636487410194-7830594388bc",
      "product_description": "Pellentesque tortor mi, ullamcorper at lorem ut, rutrum luctus mi.",
      "product_id": {
        "$numberInt": "1007"
      },
      "product_name": "Gaming computer",
      "product_price": {
        "$numberInt": "5000"
      },
      "product_type": "Computers",
      "product_quantity": {
        "$numberInt": "23"
      },
      "date": {
        "$date": {
          "$numberLong": "1686599804791"
        }
      }
    },
    ...
  ],
  "history": [
    {
      "_id": "6454f23f699573147f166fc1",
      "image_link": "1636487410194-7830594388bc",
      "product_description": "Pellentesque tortor mi, ullamcorper at lorem ut, rutrum luctus mi.",
      "product_id": {
        "$numberInt": "1007"
      },
      "product_name": "Gaming computer",
      "product_price": {
        "$numberInt": "5000"
      },
      "product_type": "Computers",
      "product_quantity": {
        "$numberInt": "26"
      },
      "date": "2023-06-12T19:55:47.878Z"
    },
    ...
  ]
}
```

Frontend

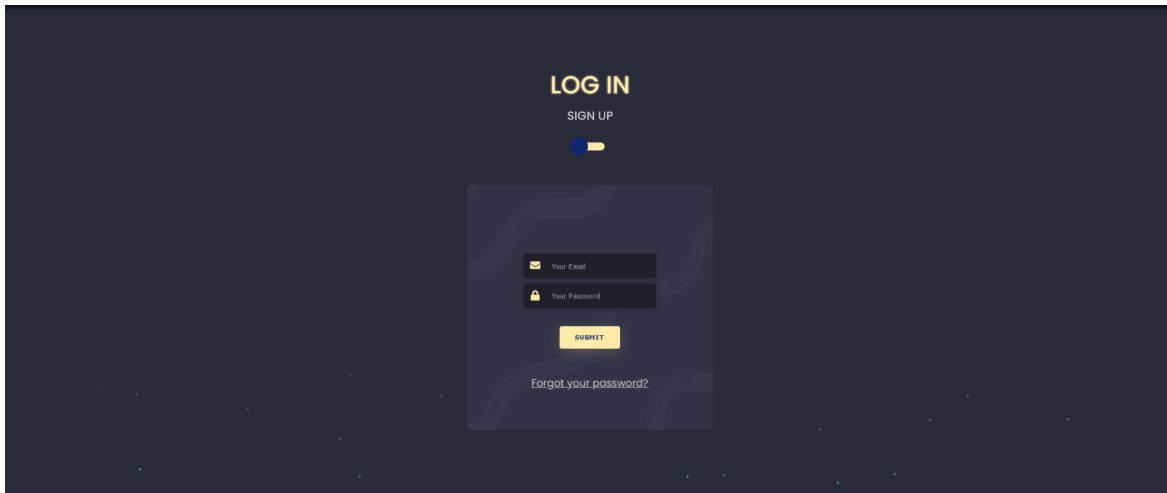
Strona home:



Strona products:



Strona login / signup:



The image shows a login and signup form on a dark background. At the top, there are two links: "LOG IN" and "SIGN UP". Below these links is a toggle switch. The form itself is a light gray box with two input fields: "Your Email" and "Your Password". Below the input fields is a "SUBMIT" button. At the bottom of the form, there is a link: "Forgot your password?".

LOG IN
SIGN UP

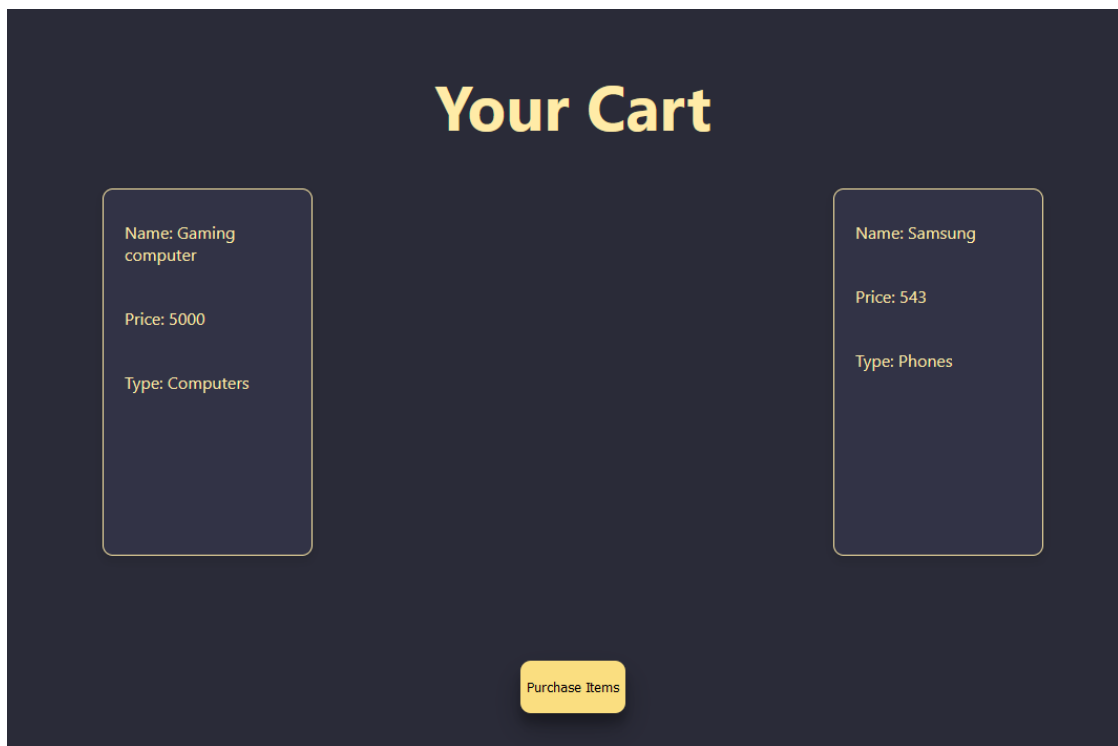
☐

Your Email
Your Password

SUBMIT

[Forgot your password?](#)

Strona koszyka:



The image shows a "Your Cart" page. The title "Your Cart" is at the top. Below the title, there are two items in the cart. The first item is a "Gaming computer" with a price of 5000 and type "Computers". The second item is a "Samsung" phone with a price of 543 and type "Phones". At the bottom of the page, there is a "Purchase Items" button.

Your Cart


Name: Gaming computer
Price: 5000
Type: Computers

Name: Samsung
Price: 543
Type: Phones


Purchase Items

Formularz dodania produktu:

Products



Headphones V2
Headphones



iPhone
Smartphone

Add Product

Name

Price

Type

Image Link

Description

Submit

Funkcjonalności

Dodawanie produktu do koszyka

Podczas dodawania produktu do koszyka jest on zarezerwowany, czyli automatycznie zmniejszana jest jego liczba. Takie podejście chroni przed chęcią zakupu produktu, który znajduje się w czymś koszyku.

Funkcja updateProductQuantity

Ta funkcja jest asynchroniczna i służy do aktualizowania ilości produktów w systemie.

```
const updateProductQuantity = async (productId, changeInQuantity) => {  
  try {  
    const res = await axios.patch(`/api/products/${productId}`, { product_quantity: changeInQuantity });  
    // Update products in state  
    setProducts(products.map((product) => {  
      if (product._id === productId) {  
        product.product_quantity = res.data.product_quantity;  
        return product;  
      }  
      return product;  
    }));  
  } catch (err) {  
    console.error('Error updating product quantity:', err);  
    setError(err);  
  }  
};
```

Funkcja addToCart

```
const addToCart = async (productId, changeInQuantity) => {  
  try {  
    await updateProductQuantity(productId, changeInQuantity);  
    const user = await getCurrentUser();  
    const response = await axios.post(`/api/users/buy/${user._id}`, { product_id: productId, product_quantity: changeInQuantity, user_id: user._id });  
  } catch (err) {  
    setError(err);  
  }  
};
```

Użycie w komponencie Product

```
{isLoggedInStatus && product.product_quantity > 0 &&  
  
<div className={styles['button']} onClick={() => onBuy(product._id, -1)}>BUY</div>
```

Dodawanie nowego produktu

Dodawanie nowego produktu możliwe jest w formularzu dostępnym na podstronie “Products”. Po wpisaniu danych nowy produkt pojawia się w bazie danych oraz jest widoczny na stronie.

Funkcja handleSubmit

Ta funkcja jest asynchroniczna i służy do obsługi przesyłania formularza w celu dodania nowego produktu. Funkcja jest wywoływana po zdarzeniu `submit` formularza.

```
const handleSubmit = async (e) => {

  e.preventDefault();

  try {

    const newProduct = {

      product_name: formValues.name,

      product_price: formValues.price,

      product_type: formValues.type,

      image_link: formValues.imageLink,

      product_quantity: formValues.quantity,

      product_description: formValues.description

    };

    const response = await axios.post('/api/products', newProduct);

    props.AddProduct(response.data);

    handleCloseForm();

  } catch (err) {

    console.error('Error adding product:', err);

    setError(err);

  }

};
```

Funkcja dodająca produkt po stronie serwera

Funkcja sprawdza podane dane, a następnie dodaje na ich podstawie nowy produkt.

```
app.post('/api/products', async (request, response) => {

    let { product_name, product_price, product_type, image_link, product_quantity, product_description } = request.body;

    if (!product_name || !product_price || !product_type || !image_link || !product_description || !product_quantity) {

        response.status(400).send('you need to fill out form before adding to database');

        return;

    }

    try {

        const db = client.db('BitShop');

        const productsCollection = db.collection('Products');

        const maxIdProduct = await productsCollection.aggregate([

            { $sort: { product_id: -1 } },

            { $limit: 1 }

        ]).toArray();

        const product_id = maxIdProduct[0].product_id + 1;

        product_price = parseFloat(product_price);

        product_quantity = parseInt(product_quantity);

        const newProduct = {

            product_name,

            product_id,

            product_price,

            product_type,

            product_description,

            product_quantity,

            image_link

        };

        const result = await productsCollection.insertOne(newProduct);

        const addedProduct = await productsCollection.findOne({ _id: result.insertedId });

        response.status(201).json(addedProduct);

    } catch (err) {

        console.error(err);

        response.status(500).send('Error connecting to the database');

    }

});
```

Usuwanie produktu

Funkcja delete po stronie serwera

Funkcja usuwa produkt z bazy danych.

```
app.delete('/api/products/:productId', async (request, response) => {  
  try {  
    const {productId} = request.params;  
    const db = client.db('BitShop');  
    const productsCollection = db.collection('Products');  
  
    const result = await productsCollection.deleteOne({_id: new  
ObjectId(productId)});  
    if (result.deletedCount === 0) {  
      response.status(404).send('Product not found');  
    } else {  
      response.status(204).send();  
    }  
  } catch (err) {  
    console.error(err);  
    response.status(500).send('Error connecting to the database');  
  }  
});
```

Funkcja removeProduct

Funkcja usuwa podany produkt. W setProducts tworzona jest nowa tablica bez odpowiedniego indeksu produktu.

```
const removeProduct = async (productId) => {  
  try {  
    await axios.delete(`/api/products/${productId}`);  
    setProducts(products.filter((product) => product._id !== productId));  
  } catch (err) {  
    console.error('Error removing product:', err);  
    setError(err);  
  }  
};
```

Przyciski sortowania i filtrowania produktów

Kod Przycisków w komponencie Product

```
<button className={styles2['button']} onClick={() =>sortPrice({product_price:1})}>Low Price</button>  
<button className={styles2['button']} onClick={() =>sortPrice({product_price:-1})}>High Price</button>  
<button className={styles2['button']} onClick={() =>matchProducts({product_quantity:{>0}},setProducts)}>Available</button>
```

Funkcja matchProducts

```
const matchProducts = async (strategy,setProducts) => {  
  
  const response = await axios.get('/api/products/match', { params: {  
  
    strategy: JSON.stringify(strategy)  
  
  }});  
  
  setProducts(response.data);  
  
};
```

Funkcja zwracająca przefiltrowane produkty po stronie serwera

Funkcja pobiera produkty spełniające określone wymogi.

```
app.get('/api/products/match', async (request, response) => {;  
  
  const strategy = JSON.parse(request.query.strategy);  
  
  if (!strategy) {  
  
    response.status(400).send('You must provide an object to update the product');  
  
    return;  
  
  }  
  
  try {  
  
    const db = client.db('BitShop');  
  
    const productsCollection = db.collection('Products');  
  
    const products = await productsCollection.find(strategy).toArray();  
  
    response.json(products);  
  
  } catch (err) {  
  
    response.status(500).send('Error connecting to the database');  
  
  }  
  
});
```

Funkcja sortująca produkty po stronie serwera

```
app.get('/api/products/sort', async (request, response) => {

  const strategy = request.query;

  if (!strategy)

  {
    response.status(400).send('You must provide an object to update the product');
    return;
  }

  try {
    const db = client.db('BitShop');
    const productsCollection = db.collection('Products');
    const products = await productsCollection.find().sort(strategy).toArray();
    response.json(products);
  } catch (err) {
    console.error(err);
    response.status(500).send('Error connecting to the database');
  }
});
```

Logowanie i rejestracja

Funkcja handleLogin

Funkcja obsługuje logowanie użytkownika. Po pomyślnym zalogowaniu użytkownik jest przekierowywany na stronę główną.

```
const handleLogInSubmit = async (e) => {  
  e.preventDefault();  
  if (!email || !password) {  
    setError('Both email and password are required');  
    return;  
  }  
  try {  
    const response = await fetch('/api/users/login', {  
      method: 'POST', headers: {  
        'Content-Type': 'application/json',  
      }, body: JSON.stringify({email, password}),  
    });  
    if (response.status === 200) {  
      const user = await response.json();  
      navigate('/');  
      window.location.reload();  
    } else {  
      const errorMessage = await response.text();  
      setError(errorMessage);  
    }  
  } catch (err) {  
    setError('Error logging in');  
  }  
};
```

Funkcja handleSingup

Funkcja obsługuje rejestrację użytkownika. Po pomyślnym zalogowaniu użytkownik jest przekierowywany na stronę główną.

```
const handleSignUpSubmit = async (e) => {
  e.preventDefault();

  if (!fullName || !signUpEmail || !signUpPassword) {
    setError('Full name, email, and password are required for sign up');
    return;
  }

  try {
    const response = await fetch('/api/users/signup', {
      method: 'POST', headers: {
        'Content-Type': 'application/json',
      }, body: JSON.stringify({
        name: fullName, email: signUpEmail, password: signUpPassword,
      })),
    });

    if (response.status === 201) {
      const user = await response.json();
      navigate('/');
      window.location.reload();
    } else {
      const errorMessage = await response.text();
      setError(errorMessage);
    }
  } catch (err) {
    setError('Error signing up');
  }
};
```


Wylogowywanie użytkownika po stronie serwera

```
//Logout
app.post('/api/users/logout', (request, response) => {
  if (!request.session.userId) {
    response.status(400).send('No user is currently logged in');
    return;
  }

  request.session.destroy((err) => {
    if (err) {
      console.error(err);
      response.status(500).send('Error logging out');
    } else {
      response.status(200).send('Logged out successfully');
    }
  });
});
```

Funkcja po stronie serwera, która zwraca aktualnie zalogowanego użytkownika

```
app.get('/api/users/current', async (request, response) => {
  if (!request.session.userId) {
    response.status(401).send('No user is currently logged in');
    return;
  }

  try {
    const db = client.db('BitShop');
    const usersCollection = db.collection('Users');

    const user = await usersCollection.findOne({_id: new ObjectId(request.session.userId)});
    if (!user) {
      response.status(404).send('User not found');
      return;
    }

    response.status(200).json(user);
  } catch (err) {
    response.status(500).send('Error connecting to the database');
  }
});
```

Logowanie użytkownika po stronie serwera

Obsługuje logowanie użytkownika - sprawdza, czy podane dane są poprawne.

```
app.post('/api/users/login', async (request, response) => {  
  const { email, password } = request.body;  
  
  if (!email || !password) {  
    response.status(400).send('Email and password are required for login');  
    return;  
  }  
  
  try {  
    const db = client.db('BitShop');  
    const usersCollection = db.collection('Users');  
  
    const user = await usersCollection.findOne({ email });  
    if (!user) {  
      response.status(404).send('User not found');  
      return;  
    }  
  
    const isPasswordCorrect = await verifyPassword(password, user.password);  
    if (!isPasswordCorrect) {  
      response.status(401).send('Invalid password');  
      return;  
    }  
  
    request.session.userId = user._id;  
    response.status(200).json(user);  
  } catch (err) {  
    response.status(500).send('Error connecting to the database');  
  }  
});
```

Rejestracja po stronie serwera

Podczas rejestracji sprawdzane jest, czy wszystkie pola są wypełnione oraz czy użytkownik o podanym adresie już istnieje. gdy rejestracja przebiegła pomyślnie, hasło użytkownika jest hashowane i dane są zapisywane w bazie.

```
//signup
app.post('/api/users/signup', async (request, response) => {
  const { name, email, password } = request.body;

  if (!name || !email || !password) {
    response.status(400).send('Full name, email, and password are required for sign up');
    return;
  }
  try {
    const db = client.db('BitShop');
    const usersCollection = db.collection('Users');

    const existingUser = await usersCollection.findOne({ email });
    if (existingUser) {
      response.status(409).send('An account with this email already exists');
      return;
    }
    const hashedPassword = await hashPassword(password);
    const newUser = {
      name,
      email,
      password: hashedPassword,
    };
    const result = await usersCollection.insertOne(newUser);
    request.session.userId = result.insertedId;
    response.status(201).json(result.ops[0]);
  } catch (err) {
    response.status(500).send('Error connecting to the database');
  }
});
```

Funkcje odpowiedzialne za haszowanie hasła i weryfikowanie go

```
const saltRounds = parseInt(process.env.BCRYPT_SALT_ROUNDS, 10);

export const hashPassword = async (password) => {
  try {
    return await bcrypt.hash(password, saltRounds);
  } catch (err) {
    throw err;
  }
};

export const verifyPassword = async (password, hashedPassword) => {
  try {
    return await bcrypt.compare(password, hashedPassword);
  } catch (err) {
    throw err;
  }
};
```

Funkcja getCurrentUser

Funkcja znajdująca aktualnie zalogowanego użytkownika.

```
export const getCurrentUser = async () => {
  try {
    const response = await fetch('/api/users/current');
    if (response.status === 200) {
      const user = await response.json();
      return user;
    } else {
      const errorMessage = await response.text();
      return null;
    }
  } catch (err) {
    return null;
  }
};
```

Funkcja isLoggedIn

Funkcja sprawdzająca, czy użytkownik jest zalogowany.

```
export const isLoggedIn = async () => {  
  const currentUser = await getCurrentUser();  
  if (currentUser) {  
    return true;  
  } else {  
    return false;  
  }  
};
```

Funkcja wylogowywująca użytkownika

```
export const logoutUser = async () => {  
  try {  
    const response = await fetch('/api/users/logout', {  
      method: 'POST',  
    });  
  
    if (response.status === 200) {  
      window.location.reload();  
    } else {  
      const errorMessage = await response.text();  
    }  
  } catch (err) {  
  }  
};
```

Kupowanie produktu i koszyk

Funkcja kupowania po stronie serwera

Funkcja odpowiedzialna za kupowanie produktu. Po zakupieniu produktu aktualizowany jest koszyk oraz historia transakcji użytkownika.

```
//buy product
app.post('/api/users/purchase/:userId', async (request, response) => {
  try {
    const {userId} = request.params;
    const {basket, history} = request.body;
    const db = client.db('BitShop');
    const usersCollection = db.collection('Users');
    // Find the user with the provided ID
    const user = await usersCollection.findOne({_id: new ObjectId(userId)});
    if (!user) {
      response.status(404).send('User not found');
      return;
    }
    const updateResult = await usersCollection.updateOne(
      {_id: new ObjectId(userId)},
      {
        $set: {
          basket: basket,
        },
        $push: {
          history: {$each: history},
        },
      }
    );
    if (updateResult.modifiedCount === 0) {
      response.status(404).send('Failed to update user');
      return;
    }
    response.status(200).send('Purchase successful');
  } catch (err) {
    response.status(500).send('Error connecting to the database');
  }
});
```

Funkcja obsługująca kupowanie

Funkcja aktualizuje historię transakcji użytkownika oraz resetuje jego koszyk.

```
const handlePurchase = async () => {  
  try {  
    const user = await getCurrentUser();  
    const response = await axios.post(`api/users/purchase/${user._id}`, {  
      basket: [],  
      history: items  
    });  
    if (response.status === 200) {  
      setItems([]);  
      alert('Purchase successful!');  
    } else {  
      throw new Error('Failed to update the basket and history');  
    }  
  } catch (error) {  
    alert('Purchase failed, please try again.');  }  
};
```

Funkcja po stronie serwera, która zwraca koszyk dla danego klienta

Funkcja zwraca koszyk danego użytkownika, aby po ponownym zalogowaniu produkty z jego koszyka wciąż były widoczne.

```
app.get('/api/users/:userId/basket', async (request, response) => {  
  try {  
    const {userId} = request.params;  
    const db = client.db('BitShop');  
    const usersCollection = db.collection('Users');  
    const user = await usersCollection.findOne({_id: new ObjectId(userId)});  
    if (!user) {  
      response.status(404).send('User not found');  
      return;  
    }  
    if (!user.basket) {  
      response.status(404).send('Basket not found');  
      return;  
    }  
    response.status(200).json(user.basket);  
  } catch (err) {  
    console.error(err);  
    response.status(500).send('Error connecting to the database');  
  }  
});
```


Funkcja pobierająca przedmioty z koszyka

Funkcja pobiera produkty z koszyka danego użytkownika.

```
export const getBasketItems = async () => {  
  try {  
    const user = await getCurrentUser();  
    if (!user) {  
      return [];  
    }  
    const response = await fetch(`/api/users/${user._id}/basket`, {  
      method: 'GET',  
      headers: {  
        'Content-Type': 'application/json',  
      },  
    });  
    if (response.status === 200) {  
      const items = await response.json();  
      return items;  
    } else {  
      const errorMessage = await response.text();  
      return [];  
    }  
  } catch (err) {  
    return [];  
  }  
};
```