Министерство Образования Республики Беларусь

УО Брестский Государственный Технический Университет

Кафедра интеллектуальных информационных технологий

Отчет

По дисциплине: «СЯРД»

Тема: «**Приложение для ведения финансового учета**»

За 3 семестр

Выполнил:

Студент 2 курса

Группы ПО-12

Гомберг Д.П.

Проверил:

Войцехович Г.Ю.

Цель: демонстрация функциональных возможностей проекта

# 1. Авторизация и аутентификация в Bonum

## Описание

Система авторизации и аутентификации в Bonum позволяет пользователям регистрироваться, входить в систему, обновлять данные и управлять сессиями через JWT-токены. Бэкенд реализован на Node.js с Express, PostgreSQL, bcrypt и jsonwebtoken. Фронтенд использует React и TypeScript, предоставляя формы для входа и регистрации. Основные функции включают создание пользователя с автоматической генерацией категорий, выдачу access и refresh токенов, а также управление данными пользователя.

## Фрагмент кода

**Бэкенд: Регистрация пользователя** (UserController.registrUser):

async registrUser(req, res) {

const { email, password } = req.body;

try {

const existingUser = await db.query('SELECT \* FROM users WHERE email = $1', [email]);

if (existingUser.rows.length > 0) {

return res.status(400).json({ error: 'User with this email already exists' });

}

const hashedPassword = await bcrypt.hash(password, 10);

const newPerson = await db.query(

'INSERT INTO users (email, password, username) VALUES ($1, $2, $3) RETURNING \*',

[email, hashedPassword, email]

);

const userId = newPerson.rows[0].id;

const parentCategories = [

{ name: 'Еда' },

{ name: 'Транспорт' },

{ name: 'Доход' },

];

const parentCategoryIds = {};

for (const category of parentCategories) {

const insertedCategory = await db.query(

'INSERT INTO categories (name, user\_id, parent\_id) VALUES ($1, $2, $3) RETURNING id',

[category.name, userId, null]

);

parentCategoryIds[category.name] = insertedCategory.rows[0].id;

}

const childCategories = [

{ name: 'Продукты', parent\_name: 'Еда' },

{ name: 'Рестораны', parent\_name: 'Еда' },

{ name: 'Мойка', parent\_name: 'Транспорт' },

{ name: 'Заправка', parent\_name: 'Транспорт' },

{ name: 'Зарплата', parent\_name: 'Доход' },

{ name: 'Алименты', parent\_name: 'Доход' },

];

for (const category of childCategories) {

const parentId = parentCategoryIds[category.parent\_name];

await db.query(

'INSERT INTO categories (name, user\_id, parent\_id) VALUES ($1, $2, $3)',

[category.name, userId, parentId]

);

}

const accessToken = jwt.sign({ email }, ACCESS\_TOKEN\_SECRET, { expiresIn: '15m' });

const refreshToken = jwt.sign({ email }, REFRESH\_TOKEN\_SECRET, { expiresIn: '30d' });

refreshTokens.push(refreshToken);

res.cookie('refreshToken', refreshToken, { httpOnly: true });

res.json({ accessToken });

} catch (err) {

console.error('Error creating user:', err);

res.status(500).json({ error: err.message });

}

}

**Фронтенд: Форма входа** (LoginPage):

const handleLogin = async () => {

if (!validateFields()) return;

try {

const response = await fetch('http://localhost:8080/api/login', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ email, password }),

credentials: 'include',

});

if (response.ok) {

showNotification('Успешный вход', 'success');

login();

navigate('/home');

} else {

showNotification('Неверные email или пароль', 'error');

console.error('Login failed');

}

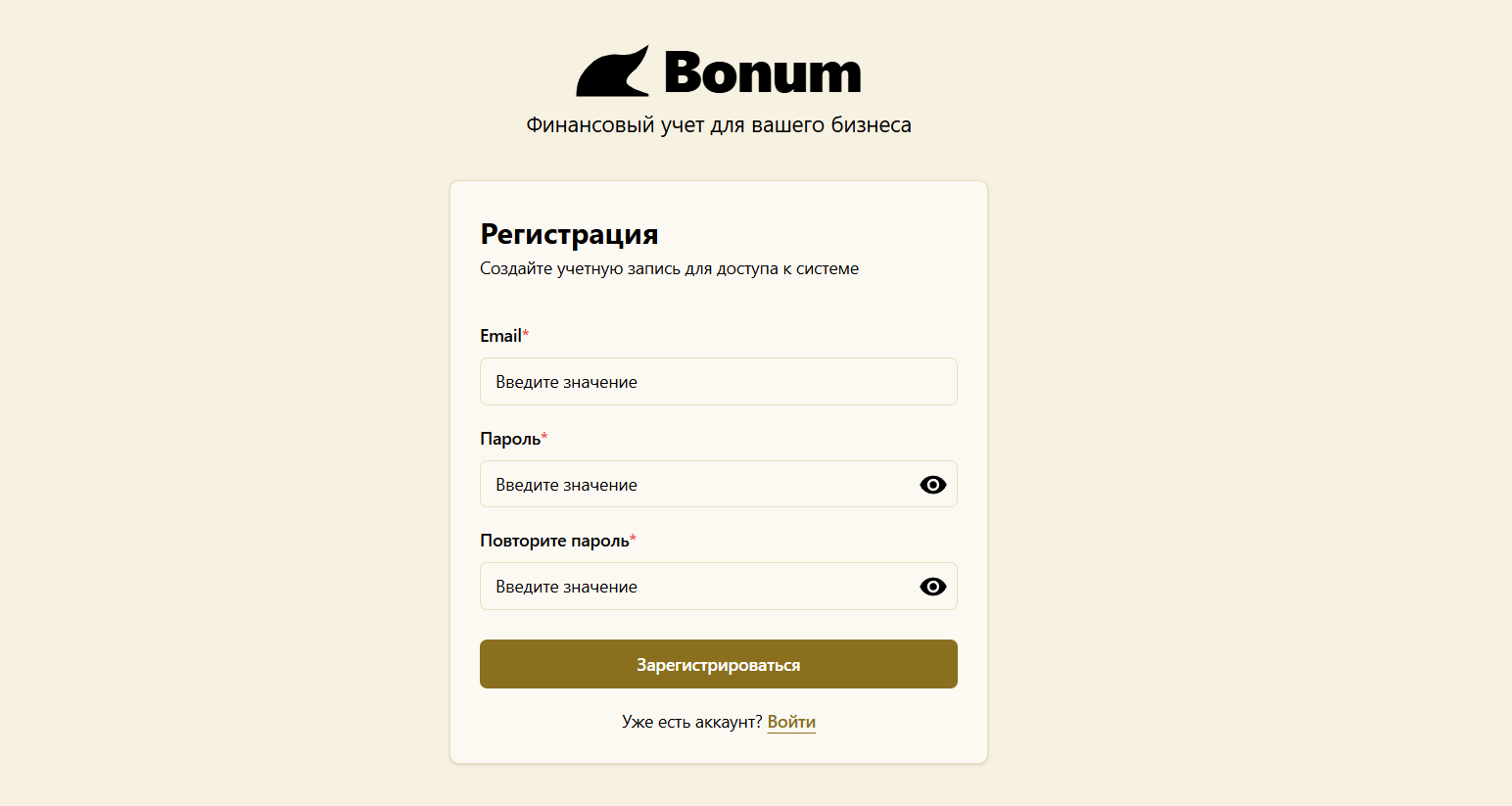
} catch (error) {

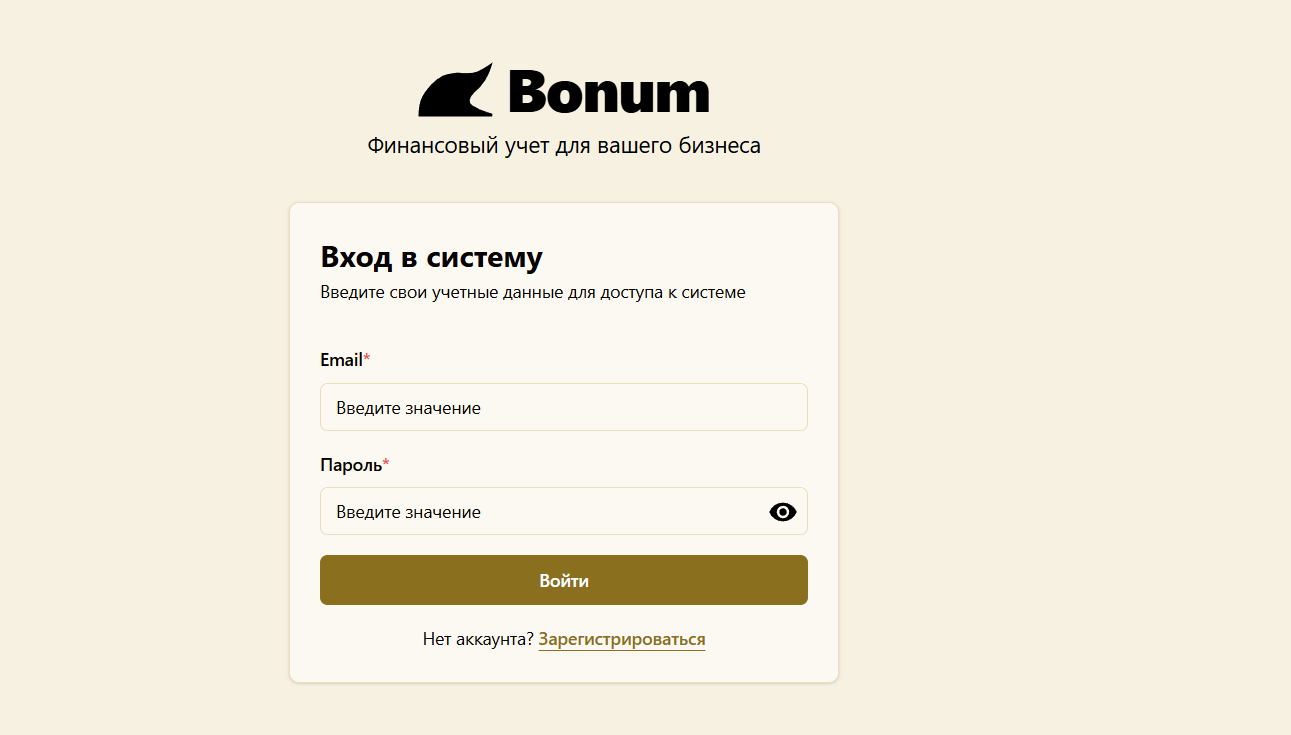
console.error('Error:', error);

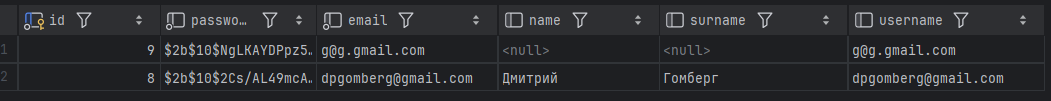
}

};

* **Функциональность**: Система позволяет регистрировать пользователей, входить в систему и обновлять токены. При регистрации автоматически создаются категории и подкатегории, что удобно для финансового учета. Фронтенд предоставляет интуитивно понятные формы с валидацией.
* **Безопасность**:
  + Хеширование паролей с bcrypt (соль = 10) обеспечивает надежную защиту.
  + Использование access (15 минут) и refresh (30 дней) токенов разделяет краткосрочный и долгосрочный доступ.
* **Валидация**: Фронтенд проверяет email, длину пароля и совпадение паролей.







# 2. База данных в Bonum

## Описание

Проект Bonum использует реляционную базу данных PostgreSQL для хранения данных о пользователях, счетах, категориях, валютах и транзакциях. PostgreSQL обеспечивает надежное хранение данных с поддержкой связей между таблицами через внешние ключи, что идеально подходит для финансового приложения с четкой структурой данных. База данных состоит из пяти таблиц: users, accounts, category, currencyType и transactions, которые обеспечивают функциональность управления финансами.

## Таблицы и их использование

### Таблица users

* **Поля**:
  + id (integer): Первичный ключ, уникальный идентификатор пользователя.
  + password (text): Хешированный пароль для авторизации.
  + email (text): Электронная почта, используется для входа.
  + name (text): Имя пользователя.
  + surname (text): Фамилия пользователя.
  + username (text): Псевдоним пользователя.
* **Использование**: Хранит данные пользователей, используется для авторизации и персонализации (например, отображения имени в интерфейсе).

### Таблица accounts

* **Поля**:
  + id (integer): Первичный ключ, уникальный идентификатор счета.
  + name (text): Название счета (например, "Основной счет").
  + value (real): Текущий баланс счета.
  + currency\_id (integer): Внешний ключ, ссылающийся на currencyType.
  + user\_id (integer): Внешний ключ, ссылающийся на users.
* **Использование**: Хранит счета пользователей, связанные с определенной валютой. Баланс (value) обновляется на основе транзакций.

### Таблица category

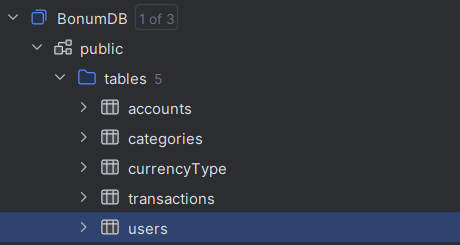
* **Поля**:
  + id (integer): Первичный ключ, уникальный идентификатор категории.
  + name (text): Название категории (например, "Еда").
  + icon (text): Иконка для категории.
  + user\_id (integer): Внешний ключ, ссылающийся на users.
  + parent\_id (integer): Внешний ключ, ссылающийся на category (для вложенных категорий).
* **Использование**: Хранит категории и подкатегории для классификации транзакций. Поддерживает иерархию (например, "Еда" → "Продукты").

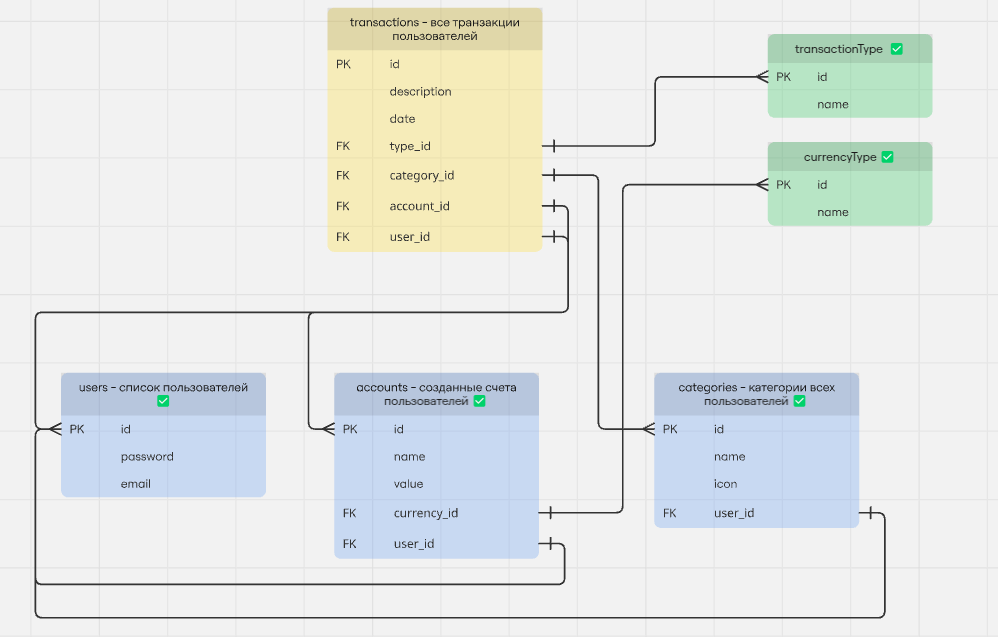
### Таблица currencyType

* **Поля**:
  + id (integer): Первичный ключ, уникальный идентификатор валюты.
  + name (text): Название валюты (например, "RUB").
* **Использование**: Хранит список валют, используется для определения валюты счетов (accounts.currency\_id).

### Таблица transactions

* **Поля**:
  + id (integer): Первичный ключ, уникальный идентификатор транзакции.
  + description (text): Описание транзакции.
  + date (text): Дата транзакции.
  + type\_id (integer): Внешний ключ, ссылающийся на тип транзакции (таблица типов не видна).
  + category\_id (integer): Внешний ключ, ссылающийся на category.
  + account\_id (integer): Внешний ключ, ссылающийся на accounts (счет-источник).
  + user\_id (integer): Внешний ключ, ссылающийся на users.
  + sum (numeric): Сумма транзакции.
  + account\_to\_id (integer): Внешний ключ, ссылающийся на accounts (счет-назначение).
* **Использование**: Хранит финансовые операции, включая доходы, расходы и переводы между счетами. Связывает транзакции с категориями, счетами и пользователями.





# 3. Фильтрация транзакций в Bonum

## Описание

Фильтрация транзакций в Bonum реализована на фронтенде с использованием React и TypeScript. Компонент TransactionsPage позволяет пользователю отображать список транзакций и фильтровать их по категории и типу транзакции (доход, расход, перевод). Данные загружаются с бэкенда через API, а фильтрация выполняется на стороне клиента. Пользовательский интерфейс включает выпадающие списки для выбора категории и типа, а также таблицу для отображения отфильтрованных транзакций с визуальным выделением сумм по типам.

## Фрагмент кода

**Логика фильтрации** (TransactionsPage):

useEffect(() => {

let filtered = transactions;

if (selectedCategory) {

filtered = filtered.filter((t) => t.category\_name === selectedCategory);

}

if (selectedType) {

filtered = filtered.filter((t) => t.type\_id === selectedType);

}

setFilteredTransactions(filtered);

}, [selectedCategory, selectedType, transactions]);

**Интерфейс фильтров**:

<div className="flex gap-4 mb-4">

<UiSelect

value={categories.indexOf(selectedCategory)}

onChange={(selected) => setSelectedCategory(selected && selected.id !== -1 ? categories[selected.id] : '')}

options={[{ id: -1, label: 'Все категории' }, ...categories.map((category, index) => ({ id: index, label: category }))]}

className={'max-w-[220px]'}

placeholder="Выберите категорию"

/>

<UiSelect

value={selectedType !== '' ? selectedType : -1}

onChange={(selected) => setSelectedType(selected && selected.id !== -1 ? selected.id : '')}

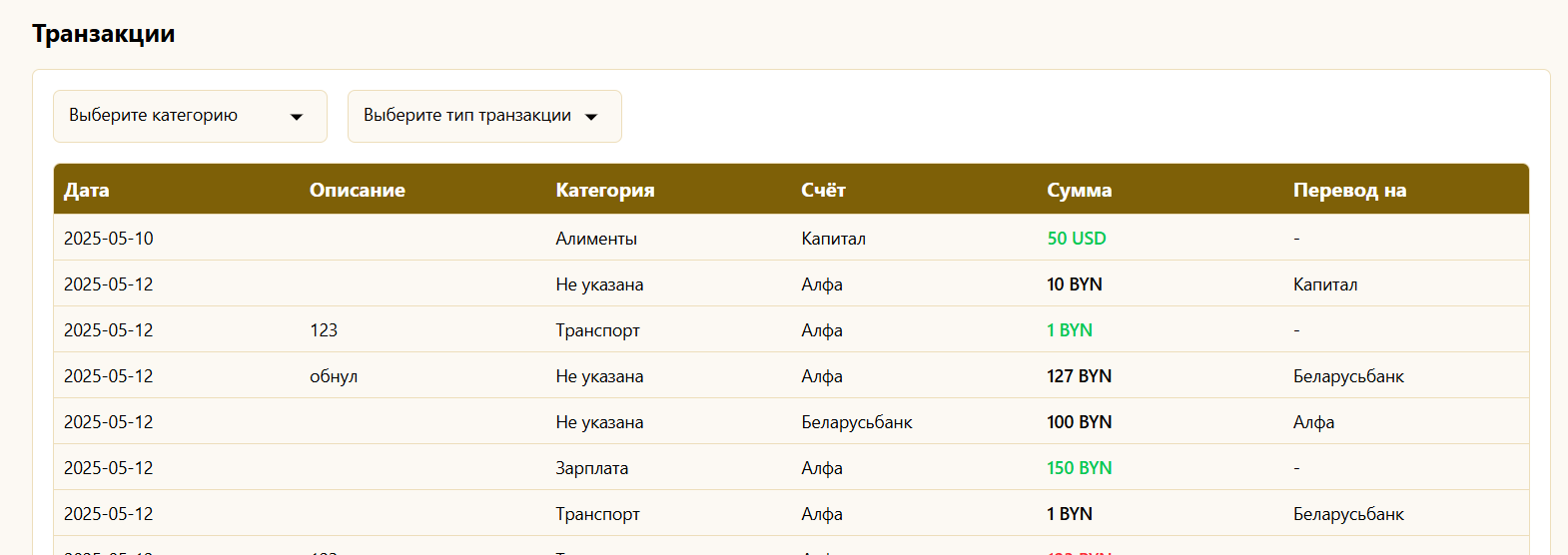
options={[{ id: -1, label: 'Все типы' }, ...transactionTypes.map((type) => ({ id: type.id, label: type.label }))]}

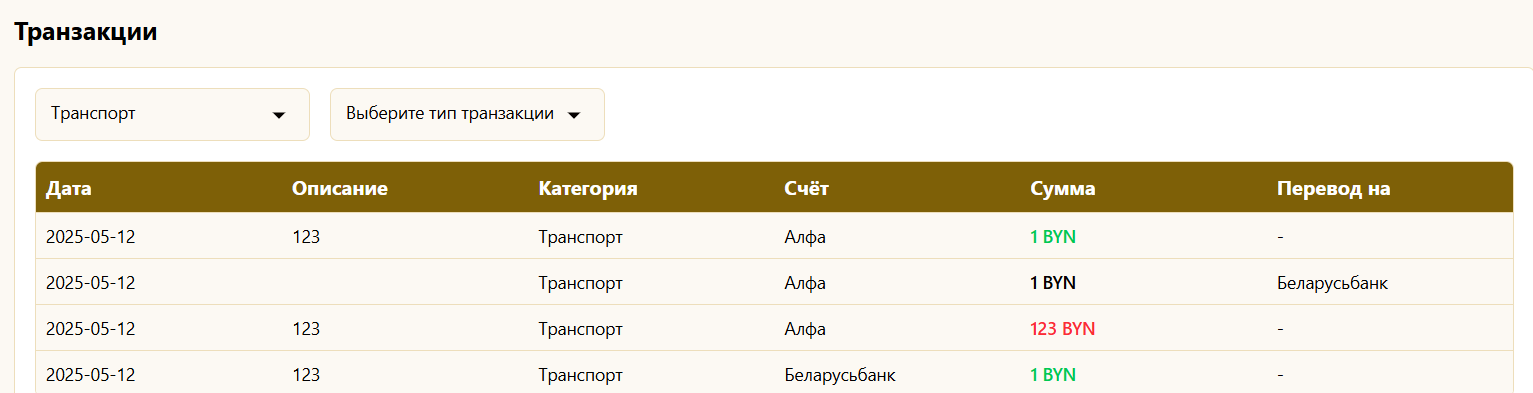
className={'max-w-[220px]'}

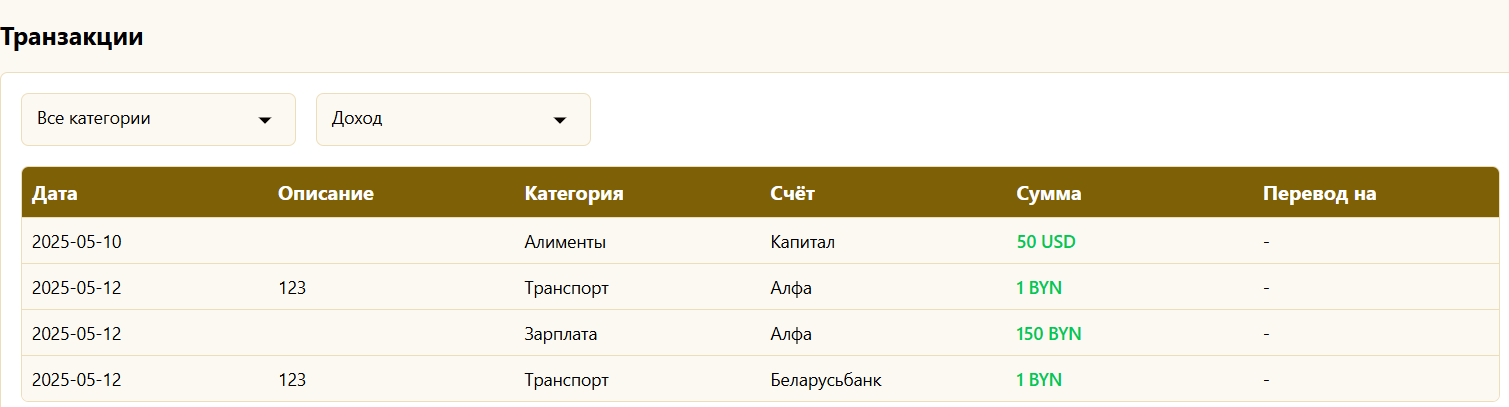
placeholder="Выберите тип транзакции"

/>

</div>







Вывод: продемонстрировали возможности проекта