**Создание викторины на FastAPI и React**

Написать код викторины для всех файлов на fastapi, python, react, sqlite с пояснениями для ребенка каждого шага программирования и создания каждого файла, компонента, переменной, оператора, функции, модуля , их взаимодействия написать код для файлов и как подключить fastapi, react, создать базу банных для модели-

BACKEND

-database.py

-requests.py

-main.py # это FastAPI

FRONTED REACT-APP

src/ components/

-ListVariant.jsx - Отображение всех вариантов

-ViewVariant.jsx - Отображение варианта с заданиями

-Task.jsx - Конкретная задача

-App.jsx

-main.jsx

**Викторина на FastAPI, React и SQLite - Пошаговое руководство для детей**

Давай создадим викторину вместе! Это будет веб-приложение, где можно отвечать на вопросы. У нас будет серверная часть (backend) на FastAPI и клиентская часть (frontend) на React, а данные будем хранить в SQLite.

**Часть 1: Backend (серверная часть)**

**1. database.py - создание базы данных**

# Импортируем нужные модули

from sqlalchemy import create\_engine, Column, Integer, String

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

# Это как коробка, где мы будем хранить все наши модели (таблицы)

Base = declarative\_base()

# Создаем модель для вопросов викторины

class QuizQuestion(Base):

\_\_tablename\_\_ = 'quiz\_questions' # Название таблицы в базе данных

# Поля таблицы:

id = Column(Integer, primary\_key=True) # Уникальный номер вопроса

question = Column(String) # Текст вопроса

option1 = Column(String) # Вариант ответа 1

option2 = Column(String) # Вариант ответа 2

option3 = Column(String) # Вариант ответа 3

option4 = Column(String) # Вариант ответа 4

correct\_answer = Column(Integer) # Номер правильного ответа (1-4)

variant\_id = Column(Integer) # К какому варианту викторины относится вопрос

# Подключаемся к базе данных SQLite (это как тетрадка для записей)

# Если файла нет, он создастся автоматически

engine = create\_engine('sqlite:///quiz.db')

# Создаем все таблицы в базе данных (если их еще нет)

Base.metadata.create\_all(engine)

# Создаем "сессию" - это как ручка, которой мы будем писать в тетрадку

Session = sessionmaker(bind=engine)

session = Session()

# Добавляем несколько тестовых вопросов, если база пустая

if not session.query(QuizQuestion).first():

questions = [

QuizQuestion(

question="Какая планета ближе всего к Солнцу?",

option1="Венера",

option2="Марс",

option3="Меркурий",

option4="Земля",

correct\_answer=3,

variant\_id=1

),

QuizQuestion(

question="Сколько будет 2+2?",

option1="3",

option2="4",

option3="5",

option4="6",

correct\_answer=2,

variant\_id=1

),

# Можно добавить больше вопросов

]

session.add\_all(questions)

session.commit()

Объяснение:

* Мы создаем базу данных SQLite с одной таблицей quiz\_questions
* В таблице хранятся вопросы, варианты ответов и правильный ответ
* Base = declarative\_base() - это основа для всех наших моделей (как чертеж)
* Column - это столбец в таблице (как колонка в тетради)
* session - это как ручка, которой мы записываем данные в базу

**2. requests.py - обработка запросов**

from fastapi import APIRouter

from database import session, QuizQuestion

# Создаем "маршрутизатор" - это как указатель, куда идти

router = APIRouter()

# Получить все варианты викторины

@router.get("/variants")

def get\_variants():

# Находим все уникальные variant\_id из базы данных

variants = session.query(QuizQuestion.variant\_id).distinct().all()

# Преобразуем результат в простой список чисел

return {"variants": [v[0] for v in variants]}

# Получить вопросы для конкретного варианта

@router.get("/variant/{variant\_id}")

def get\_variant(variant\_id: int):

# Находим все вопросы для указанного варианта

questions = session.query(QuizQuestion).filter\_by(variant\_id=variant\_id).all()

# Преобразуем вопросы в словари для отправки клиенту

result = []

for q in questions:

result.append({

"id": q.id,

"question": q.question,

"options": [q.option1, q.option2, q.option3, q.option4],

"correct\_answer": q.correct\_answer

})

return {"questions": result}

# Проверить ответ на вопрос

@router.post("/check\_answer/{question\_id}")

def check\_answer(question\_id: int, selected\_answer: int):

# Находим вопрос в базе данных

question = session.query(QuizQuestion).get(question\_id)

if not question:

return {"error": "Question not found"}

# Проверяем, правильный ли ответ

is\_correct = selected\_answer == question.correct\_answer

return {

"is\_correct": is\_correct,

"correct\_answer": question.correct\_answer

}

Объяснение:

* APIRouter - это как карта маршрутов для нашего API
* @router.get - это маршрут для GET-запросов (когда браузер просит данные)
* @router.post - это маршрут для POST-запросов (когда браузер отправляет данные)
* Мы создаем 3 маршрута: для получения вариантов, вопросов и проверки ответов
* session.query - это как задать вопрос базе данных

**3. main.py - FastAPI приложение**

from fastapi import FastAPI

from fastapi.middleware.cors import CORSMiddleware

from requests import router as quiz\_router

# Создаем приложение FastAPI - это как главный пульт управления

app = FastAPI(title="Викторина для детей")

# Разрешаем запросы из браузера (это важно для React)

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"], # Разрешаем все источники (для разработки)

allow\_methods=["\*"], # Разрешаем все методы (GET, POST и т.д.)

allow\_headers=["\*"], # Разрешаем все заголовки

)

# Подключаем наши маршруты из requests.py

app.include\_router(quiz\_router, prefix="/api")

# Простой маршрут для проверки, что сервер работает

@app.get("/")

def read\_root():

return {"message": "Добро пожаловать в викторину!"}

Объяснение:

* FastAPI() - создаем наше приложение
* CORSMiddleware - это как пропуск, который разрешает React приложению общаться с нашим сервером
* include\_router - подключаем наши маршруты из requests.py
* @app.get("/") - корневой маршрут, который просто приветствует пользователей

**Часть 2: Frontend (клиентская часть на React)**

**1. Создаем React приложение**

Сначала нужно создать React приложение. Открой терминал и выполни:

npx create-react-app quiz-frontend

cd quiz-frontend

**2. App.jsx - главный компонент**

import { useState, useEffect } from 'react';

import ListVariant from './components/ListVariant';

import ViewVariant from './components/ViewVariant';

function App() {

const [variants, setVariants] = useState([]); // Список вариантов викторины

const [selectedVariant, setSelectedVariant] = useState(null); // Выбранный вариант

const [loading, setLoading] = useState(true); // Загрузка данных

// Загружаем варианты викторины при загрузке страницы

useEffect(() => {

fetch('http://localhost:8000/api/variants')

.then(response => response.json())

.then(data => {

setVariants(data.variants);

setLoading(false);

})

.catch(error => {

console.error('Ошибка:', error);

setLoading(false);

});

}, []);

if (loading) {

return <div>Загрузка...</div>;

}

return (

<div className="app">

<h1>Викторина для детей</h1>

{selectedVariant ? (

<ViewVariant

variantId={selectedVariant}

onBack={() => setSelectedVariant(null)}

/>

) : (

<ListVariant

variants={variants}

onSelect={setSelectedVariant}

/>

)}

</div>

);

}

export default App;

Объяснение:

* useState - это как коробочка, где хранятся данные компонента
* useEffect - выполняется при загрузке страницы (как onLoad)
* fetch - отправляет запрос на наш сервер FastAPI
* Мы показываем либо список вариантов, либо выбранный вариант

**3. components/ListVariant.jsx - список вариантов**

function ListVariant({ variants, onSelect }) {

return (

<div className="variant-list">

<h2>Выберите вариант викторины:</h2>

<ul>

{variants.map(variant => (

<li key={variant}>

<button onClick={() => onSelect(variant)}>

Вариант {variant}

</button>

</li>

))}

</ul>

</div>

);

}

export default ListVariant;

Объяснение:

* Это просто список кнопок с вариантами викторины
* Когда нажимаем на кнопку, вызывается onSelect с номером варианта
* map - превращает массив чисел в массив элементов JSX

**4. components/ViewVariant.jsx - просмотр варианта**

import { useState, useEffect } from 'react';

import Task from './Task';

function ViewVariant({ variantId, onBack }) {

const [questions, setQuestions] = useState([]);

const [loading, setLoading] = useState(true);

// Загружаем вопросы для выбранного варианта

useEffect(() => {

fetch(`http://localhost:8000/api/variant/${variantId}`)

.then(response => response.json())

.then(data => {

setQuestions(data.questions);

setLoading(false);

})

.catch(error => {

console.error('Ошибка:', error);

setLoading(false);

});

}, [variantId]);

if (loading) {

return <div>Загрузка вопросов...</div>;

}

return (

<div className="variant-view">

<button onClick={onBack}>← Назад к списку</button>

<h2>Вариант {variantId}</h2>

<div className="questions">

{questions.map((question, index) => (

<Task

key={question.id}

question={question}

number={index + 1}

/>

))}

</div>

</div>

);

}

export default ViewVariant;

Объяснение:

* Загружает вопросы для выбранного варианта
* Показывает кнопку "Назад" и список вопросов
* Каждый вопрос рендерит компонент Task

**5. components/Task.jsx - отдельный вопрос**

import { useState } from 'react';

function Task({ question, number }) {

const [selectedOption, setSelectedOption] = useState(null);

const [result, setResult] = useState(null);

const handleSubmit = () => {

if (selectedOption === null) return;

// Отправляем ответ на сервер для проверки

fetch(`http://localhost:8000/api/check\_answer/${question.id}`, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ selected\_answer: selectedOption }),

})

.then(response => response.json())

.then(data => {

setResult(data);

})

.catch(error => {

console.error('Ошибка:', error);

});

};

return (

<div className="task">

<h3>Вопрос {number}: {question.question}</h3>

<div className="options">

{question.options.map((option, idx) => (

<div key={idx} className="option">

<label>

<input

type="radio"

name={`question-${question.id}`}

checked={selectedOption === idx + 1}

onChange={() => setSelectedOption(idx + 1)}

/>

{option}

</label>

</div>

))}

</div>

<button onClick={handleSubmit} disabled={selectedOption === null}>

Проверить

</button>

{result && (

<div className={`result ${result.is\_correct ? 'correct' : 'wrong'}`}>

{result.is\_correct ? 'Правильно! 🎉' : `Неправильно. Правильный ответ: ${question.options[result.correct\_answer - 1]}`}

</div>

)}

</div>

);

}

export default Task;

Объяснение:

* Показывает один вопрос с вариантами ответов
* Позволяет выбрать ответ и отправить его на сервер
* Показывает результат (правильно/неправильно)
* useState хранит выбранный вариант и результат проверки
* fetch отправляет ответ на сервер и получает результат

**6. main.jsx - входная точка**

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

import './index.css';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

Объяснение:

* Это стартовая точка React приложения
* Рендерит наш главный компонент App в элемент с id="root"

**Как запустить приложение**

1. **Backend**:
   * Установи Python (если еще не установлен)
   * Установи зависимости:

pip install fastapi uvicorn sqlalchemy

* + Запусти сервер:

uvicorn main:app --reload

* + Сервер будет доступен по адресу: [http://localhost:8000](http://localhost:8000/)

1. **Frontend**:
   * Перейди в папку quiz-frontend
   * Установи зависимости (если нужно):

npm install

* + Запусти React приложение:

npm start

* + Приложение откроется в браузере по адресу: [http://localhost:3000](http://localhost:3000/)

**Как все работает вместе**

1. Пользователь открывает React приложение в браузере
2. React приложение запрашивает у FastAPI сервера список вариантов викторины
3. FastAPI обращается к базе данных SQLite и возвращает список вариантов
4. Когда пользователь выбирает вариант, React запрашивает вопросы для этого варианта
5. FastAPI находит вопросы в базе данных и возвращает их
6. Когда пользователь отвечает на вопрос, React отправляет ответ на сервер
7. FastAPI проверяет ответ и возвращает результат
8. React показывает пользователю, правильно ли он ответил

Теперь у тебя есть собственная викторина! Ты можешь добавлять новые вопросы в базу данных, менять стили или добавлять новые функции.

# Викторина на FastAPI, React и SQLite - Пошаговое руководство для детей

Давай создадим викторину вместе! Это будет веб-приложение, где можно отвечать на вопросы. У нас будет серверная часть (backend) на FastAPI и клиентская часть (frontend) на React, а данные будем хранить в SQLite.

## Часть 1: Backend (серверная часть)

### 1. database.py - создание базы данных

```python

# Импортируем нужные модули

from sqlalchemy import create\_engine, Column, Integer, String

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

# Это как коробка, где мы будем хранить все наши модели (таблицы)

Base = declarative\_base()

# Создаем модель для вопросов викторины

class QuizQuestion(Base):

\_\_tablename\_\_ = 'quiz\_questions' # Название таблицы в базе данных

# Поля таблицы:

id = Column(Integer, primary\_key=True) # Уникальный номер вопроса

question = Column(String) # Текст вопроса

option1 = Column(String) # Вариант ответа 1

option2 = Column(String) # Вариант ответа 2

option3 = Column(String) # Вариант ответа 3

option4 = Column(String) # Вариант ответа 4

correct\_answer = Column(Integer) # Номер правильного ответа (1-4)

variant\_id = Column(Integer) # К какому варианту викторины относится вопрос

# Подключаемся к базе данных SQLite (это как тетрадка для записей)

# Если файла нет, он создастся автоматически

engine = create\_engine('sqlite:///quiz.db')

# Создаем все таблицы в базе данных (если их еще нет)

Base.metadata.create\_all(engine)

# Создаем "сессию" - это как ручка, которой мы будем писать в тетрадку

Session = sessionmaker(bind=engine)

session = Session()

# Добавляем несколько тестовых вопросов, если база пустая

if not session.query(QuizQuestion).first():

questions = [

QuizQuestion(

question="Какая планета ближе всего к Солнцу?",

option1="Венера",

option2="Марс",

option3="Меркурий",

option4="Земля",

correct\_answer=3,

variant\_id=1

),

QuizQuestion(

question="Сколько будет 2+2?",

option1="3",

option2="4",

option3="5",

option4="6",

correct\_answer=2,

variant\_id=1

),

# Можно добавить больше вопросов

]

session.add\_all(questions)

session.commit()

```

Объяснение:

- Мы создаем базу данных SQLite с одной таблицей `quiz\_questions`

- В таблице хранятся вопросы, варианты ответов и правильный ответ

- `Base = declarative\_base()` - это основа для всех наших моделей (как чертеж)

- `Column` - это столбец в таблице (как колонка в тетради)

- `session` - это как ручка, которой мы записываем данные в базу

### 2. requests.py - обработка запросов

```python

from fastapi import APIRouter

from database import session, QuizQuestion

# Создаем "маршрутизатор" - это как указатель, куда идти

router = APIRouter()

# Получить все варианты викторины

@router.get("/variants")

def get\_variants():

# Находим все уникальные variant\_id из базы данных

variants = session.query(QuizQuestion.variant\_id).distinct().all()

# Преобразуем результат в простой список чисел

return {"variants": [v[0] for v in variants]}

# Получить вопросы для конкретного варианта

@router.get("/variant/{variant\_id}")

def get\_variant(variant\_id: int):

# Находим все вопросы для указанного варианта

questions = session.query(QuizQuestion).filter\_by(variant\_id=variant\_id).all()

# Преобразуем вопросы в словари для отправки клиенту

result = []

for q in questions:

result.append({

"id": q.id,

"question": q.question,

"options": [q.option1, q.option2, q.option3, q.option4],

"correct\_answer": q.correct\_answer

})

return {"questions": result}

# Проверить ответ на вопрос

@router.post("/check\_answer/{question\_id}")

def check\_answer(question\_id: int, selected\_answer: int):

# Находим вопрос в базе данных

question = session.query(QuizQuestion).get(question\_id)

if not question:

return {"error": "Question not found"}

# Проверяем, правильный ли ответ

is\_correct = selected\_answer == question.correct\_answer

return {

"is\_correct": is\_correct,

"correct\_answer": question.correct\_answer

}

```

Объяснение:

- `APIRouter` - это как карта маршрутов для нашего API

- `@router.get` - это маршрут для GET-запросов (когда браузер просит данные)

- `@router.post` - это маршрут для POST-запросов (когда браузер отправляет данные)

- Мы создаем 3 маршрута: для получения вариантов, вопросов и проверки ответов

- `session.query` - это как задать вопрос базе данных

### 3. main.py - FastAPI приложение

```python

from fastapi import FastAPI

from fastapi.middleware.cors import CORSMiddleware

from requests import router as quiz\_router

# Создаем приложение FastAPI - это как главный пульт управления

app = FastAPI(title="Викторина для детей")

# Разрешаем запросы из браузера (это важно для React)

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"], # Разрешаем все источники (для разработки)

allow\_methods=["\*"], # Разрешаем все методы (GET, POST и т.д.)

allow\_headers=["\*"], # Разрешаем все заголовки

)

# Подключаем наши маршруты из requests.py

app.include\_router(quiz\_router, prefix="/api")

# Простой маршрут для проверки, что сервер работает

@app.get("/")

def read\_root():

return {"message": "Добро пожаловать в викторину!"}

```

Объяснение:

- `FastAPI()` - создаем наше приложение

- `CORSMiddleware` - это как пропуск, который разрешает React приложению общаться с нашим сервером

- `include\_router` - подключаем наши маршруты из requests.py

- `@app.get("/")` - корневой маршрут, который просто приветствует пользователей

## Часть 2: Frontend (клиентская часть на React)

### 1. Создаем React приложение

Сначала нужно создать React приложение. Открой терминал и выполни:

```bash

npx create-react-app quiz-frontend

cd quiz-frontend

```

### 2. App.jsx - главный компонент

```jsx

import { useState, useEffect } from 'react';

import ListVariant from './components/ListVariant';

import ViewVariant from './components/ViewVariant';

function App() {

const [variants, setVariants] = useState([]); // Список вариантов викторины

const [selectedVariant, setSelectedVariant] = useState(null); // Выбранный вариант

const [loading, setLoading] = useState(true); // Загрузка данных

// Загружаем варианты викторины при загрузке страницы

useEffect(() => {

fetch('http://localhost:8000/api/variants')

.then(response => response.json())

.then(data => {

setVariants(data.variants);

setLoading(false);

})

.catch(error => {

console.error('Ошибка:', error);

setLoading(false);

});

}, []);

if (loading) {

return <div>Загрузка...</div>;

}

return (

<div className="app">

<h1>Викторина для детей</h1>

{selectedVariant ? (

<ViewVariant

variantId={selectedVariant}

onBack={() => setSelectedVariant(null)}

/>

) : (

<ListVariant

variants={variants}

onSelect={setSelectedVariant}

/>

)}

</div>

);

}

export default App;

```

Объяснение:

- `useState` - это как коробочка, где хранятся данные компонента

- `useEffect` - выполняется при загрузке страницы (как `onLoad`)

- `fetch` - отправляет запрос на наш сервер FastAPI

- Мы показываем либо список вариантов, либо выбранный вариант

### 3. components/ListVariant.jsx - список вариантов

```jsx

function ListVariant({ variants, onSelect }) {

return (

<div className="variant-list">

<h2>Выберите вариант викторины:</h2>

<ul>

{variants.map(variant => (

<li key={variant}>

<button onClick={() => onSelect(variant)}>

Вариант {variant}

</button>

</li>

))}

</ul>

</div>

);

}

export default ListVariant;

```

Объяснение:

- Это просто список кнопок с вариантами викторины

- Когда нажимаем на кнопку, вызывается `onSelect` с номером варианта

- `map` - превращает массив чисел в массив элементов JSX

### 4. components/ViewVariant.jsx - просмотр варианта

```jsx

import { useState, useEffect } from 'react';

import Task from './Task';

function ViewVariant({ variantId, onBack }) {

const [questions, setQuestions] = useState([]);

const [loading, setLoading] = useState(true);

// Загружаем вопросы для выбранного варианта

useEffect(() => {

fetch(`http://localhost:8000/api/variant/${variantId}`)

.then(response => response.json())

.then(data => {

setQuestions(data.questions);

setLoading(false);

})

.catch(error => {

console.error('Ошибка:', error);

setLoading(false);

});

}, [variantId]);

if (loading) {

return <div>Загрузка вопросов...</div>;

}

return (

<div className="variant-view">

<button onClick={onBack}>← Назад к списку</button>

<h2>Вариант {variantId}</h2>

<div className="questions">

{questions.map((question, index) => (

<Task

key={question.id}

question={question}

number={index + 1}

/>

))}

</div>

</div>

);

}

export default ViewVariant;

```

Объяснение:

- Загружает вопросы для выбранного варианта

- Показывает кнопку "Назад" и список вопросов

- Каждый вопрос рендерит компонент `Task`

### 5. components/Task.jsx - отдельный вопрос

```jsx

import { useState } from 'react';

function Task({ question, number }) {

const [selectedOption, setSelectedOption] = useState(null);

const [result, setResult] = useState(null);

const handleSubmit = () => {

if (selectedOption === null) return;

// Отправляем ответ на сервер для проверки

fetch(`http://localhost:8000/api/check\_answer/${question.id}`, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ selected\_answer: selectedOption }),

})

.then(response => response.json())

.then(data => {

setResult(data);

})

.catch(error => {

console.error('Ошибка:', error);

});

};

return (

<div className="task">

<h3>Вопрос {number}: {question.question}</h3>

<div className="options">

{question.options.map((option, idx) => (

<div key={idx} className="option">

<label>

<input

type="radio"

name={`question-${question.id}`}

checked={selectedOption === idx + 1}

onChange={() => setSelectedOption(idx + 1)}

/>

{option}

</label>

</div>

))}

</div>

<button onClick={handleSubmit} disabled={selectedOption === null}>

Проверить

</button>

{result && (

<div className={`result ${result.is\_correct ? 'correct' : 'wrong'}`}>

{result.is\_correct ? 'Правильно! 🎉' : `Неправильно. Правильный ответ: ${question.options[result.correct\_answer - 1]}`}

</div>

)}

</div>

);

}

export default Task;

```

Объяснение:

- Показывает один вопрос с вариантами ответов

- Позволяет выбрать ответ и отправить его на сервер

- Показывает результат (правильно/неправильно)

- `useState` хранит выбранный вариант и результат проверки

- `fetch` отправляет ответ на сервер и получает результат

### 6. main.jsx - входная точка

```jsx

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

import './index.css';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

```

Объяснение:

- Это стартовая точка React приложения

- Рендерит наш главный компонент `App` в элемент с id="root"

## Как запустить приложение

1. \*\*Backend\*\*:

- Установи Python (если еще не установлен)

- Установи зависимости:

```bash

pip install fastapi uvicorn sqlalchemy

```

- Запусти сервер:

```bash

uvicorn main:app --reload

```

- Сервер будет доступен по адресу: http://localhost:8000

2. \*\*Frontend\*\*:

- Перейди в папку `quiz-frontend`

- Установи зависимости (если нужно):

```bash

npm install

```

- Запусти React приложение:

```bash

npm start

```

- Приложение откроется в браузере по адресу: http://localhost:3000

## Как все работает вместе

1. Пользователь открывает React приложение в браузере

2. React приложение запрашивает у FastAPI сервера список вариантов викторины

3. FastAPI обращается к базе данных SQLite и возвращает список вариантов

4. Когда пользователь выбирает вариант, React запрашивает вопросы для этого варианта

5. FastAPI находит вопросы в базе данных и возвращает их

6. Когда пользователь отвечает на вопрос, React отправляет ответ на сервер

7. FastAPI проверяет ответ и возвращает результат

8. React показывает пользователю, правильно ли он ответил

Теперь у тебя есть собственная викторина! Ты можешь добавлять новые вопросы в базу данных, менять стили или добавлять новые функции.