

Пошаговая реализация JWT-аутентификации для FastAPI + React

Руководство по внедрению

7 декабря 2025 г.

Аннотация

Данное руководство предоставляет пошаговый план внедрения системы регистрации, аутентификации и авторизации пользователей с использованием JWT-токенов в существующее приложение на FastAPI (бэкенд) и React (фронтенд). Каждый этап включает конкретные изменения кода, инструкции по тестированию и проверки работоспособности.

Содержание

1	Исходное состояние	2
2	Пошаговый план модернизации	2
2.1	Этап 1. Добавить модель пользователя и обновить Note	2
2.1.1	Сервер (database_models.py)	2
2.1.2	Обновить БД	2
2.2	Этап 2. Добавить security-утилиты и Pydantic-схемы	2
2.2.1	Установите зависимости	2
2.2.2	Создайте security.py	3
2.2.3	Обновите validation_models.py	3
2.3	Этап 3. Реализовать /register и /token	4
2.3.1	Обновите main.py (частично — только auth-эндпоинты)	4
2.4	Этап 4. Защитить эндпоинты заметок	5
2.4.1	Обновите все CRUD-эндпоинты в main.py	5
2.5	Этап 5. Настроить axios-клиент на автоматическую отправку токена	7
2.5.1	Обновите api.ts (или ваш файл с axios)	7
2.6	Этап 6. Добавить аутентификационный сервис и UI-страницы	8
2.6.1	Создайте auth.ts	8
2.6.2	Создайте Login.tsx	9
2.6.3	Создайте Register.tsx	10
2.6.4	Создайте ProtectedRoute.tsx	12
2.6.5	Обновите router.ts	12
2.7	Этап 7. Добавить кнопку «Выйти»	13
2.7.1	Обновите App.tsx	13
3	Финальная архитектура системы	14
4	Проверка корректности и тестирование	14
4.1	Тестирование через Postman	14
4.2	Тестирование в браузере	14
5	Заключение	14

1 Исходное состояние

- Сервер на FastAPI: CRUD для заметок без авторизации
- Клиент на React + axios: работает с </api/notes/>
- БД: SQLite, модель Note
- Роутинг настроен через createBrowserRouter

2 Пошаговый план модернизации

2.1 Этап 1. Добавить модель пользователя и обновить Note

Цель: БД поддерживает пользователей и привязку заметок.

Тест через Postman: можно создать пользователя вручную, но API пока не трогаем.

2.1.1 Сервер (database_models.py)

```
1 from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
2
3 class Base(DeclarativeBase):
4     pass
5
6 class User(Base):
7     __tablename__ = "users"
8     id: Mapped[int] = mapped_column(primary_key=True)
9     email: Mapped[str] = mapped_column(unique=True, index=True)
10    hashed_password: Mapped[str]
11
12 class Note(Base):
13     __tablename__ = "notes"
14     id: Mapped[int] = mapped_column(primary_key=True)
15     title: Mapped[str]
16     content: Mapped[str]
17     owner_id: Mapped[int] # Новое поле для привязки к пользователю
```

Listing 1: database_models.py

2.1.2 Обновить БД

Удалите notes.db (или используйте миграции, но для простоты — пересоздайте):

```
1 rm notes.db
```

Запустите сервер — таблицы создадутся автоматически.

Проверка: откройте notes.db (например, в DB Browser for SQLite) — должны быть таблицы users и notes с полем owner_id.

2.2 Этап 2. Добавить security-утилиты и Pydantic-схемы

Цель: Механизм хеширования паролей и генерации JWT.

2.2.1 Установите зависимости

```
1 pip install python-jose[cryptography] passlib[bcrypt] pydantic[email]
```

2.2.2 Создайте security.py

```
1 from datetime import datetime, timedelta, timezone
2 from typing import Optional, Dict
3 from jose import jwt
4 from passlib.context import CryptContext
5
6 # Настройки безопасности (в продакшене используйте переменные окружения)
7 SECRET_KEY = "your-secret-key-change-in-production-2025"
8 ALGORITHM = "HS256"
9 ACCESS_TOKEN_EXPIRE_MINUTES = 30
10
11 pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
12
13 def verify_password(plain_password: str, hashed_password: str) -> bool:
14     Проверяет"" соответствие пароля хешу""
15     return pwd_context.verify(plain_password, hashed_password)
16
17 def get_password_hash(password: str) -> str:
18     Генерирует"" хеш пароля""
19     return pwd_context.hash(password)
20
21 def create_access_token(data: Dict[str, str], expires_delta: Optional[timedelta] = None)
22     -> str:
23     Создает"" JWT-токен с указанным временем жизни""
24     to_encode = data.copy()
25     expire = datetime.now(timezone.utc) + (expires_delta or timedelta(minutes=
26     ACCESS_TOKEN_EXPIRE_MINUTES))
27     to_encode.update({"exp": expire})
28     return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
```

Listing 2: security.py

2.2.3 Обновите validation_models.py

```
1 from pydantic import BaseModel, EmailStr, Field, ConfigDict
2 from typing import Annotated, Optional
3
4 # Типы с валидацией
5 TitleType = Annotated[str, Field(min_length=1, max_length=100)]
6 ContentType = Annotated[str, Field(min_length=1)]
7 IdType = Annotated[int, Field(gt=0)]
8
9 class NoteBase(BaseModel):
10     title: TitleType
11     content: ContentType
12
13 class NoteCreate(NoteBase):
14     pass
15
16 class NoteOut(NoteBase):
17     id: IdType
18     owner_id: IdType
19
20     model_config = ConfigDict(from_attributes=True)
21
22 # Схемы для аутентификации
23 class UserCreate(BaseModel):
24     email: EmailStr
25     password: str = Field(min_length=6, max_length=100)
26
27 class Token(BaseModel):
28     access_token: str
29     token_type: str = "bearer"
30
31 class TokenData(BaseModel):
32     id: Optional[str] = None
```

Listing 3: validation_models.py

Проверка: убедитесь, что сервер запускается без ошибок.

2.3 Этап 3. Реализовать `/register` и `/token`

Цель: Возможность зарегистрироваться и получить JWT.

2.3.1 Обновите `main.py` (частично — только `auth`-эндпоинты)

Добавьте в `main.py` **перед заметками** следующие импорты и зависимости:

```
1 from fastapi import Depends, HTTPException, status, APIRouter
2 from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
3 from sqlalchemy.orm import Session
4 from jose import JWTError, jwt
5
6 # Импорты локальных модулей
7 from database import get_session, SessionLocal
8 from database_models import User, Note
9 from validation_models import NoteOut, NoteCreate, UserCreate, Token
10 from security import (
11     verify_password,
12     get_password_hash,
13     create_access_token,
14     SECRET_KEY,
15     ALGORITHM,
16     ACCESS_TOKEN_EXPIRE_MINUTES
17 )
18
19 # OAuth2 схема для извлечения токена из заголовка
20 oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")
21
22 def get_current_user(
23     token: str = Depends(oauth2_scheme),
24     session: Session = Depends(get_session)
25 ) -> User:
26     Извлекает"" текущего пользователя из JWTтокена-""
27     credentials_exception = HTTPException(
28         status_code=status.HTTP_401_UNAUTHORIZED,
29         detail="Could not validate credentials",
30         headers={"WWW-Authenticate": "Bearer"},
31     )
32     try:
33         payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
34         user_id: str = payload.get("sub")
35         if user_id is None:
36             raise credentials_exception
37     except JWTError:
38         raise credentials_exception
39
40     user = session.query(User).filter(User.id == int(user_id)).first()
41     if user is None:
42         raise credentials_exception
43     return user
```

Теперь добавьте эндпоинты аутентификации:

```
1 # Эндпоинты аутентификации
2 @app.post("/register", response_model=Token, summaryРегистрация=" нового пользователя")
3 def register(user: UserCreate, session: Session = Depends(get_session)):
4     Регистрирует"" нового пользователя и возвращает JWTтокен-""
5     # Проверка на существование email
6     if session.query(User).filter(User.email == user.email).first():
7         raise HTTPException(
8             status_code=status.HTTP_400_BAD_REQUEST,
9             detail="Email already registered"
10        )
11
12     # Хеширование пароля и сохранение пользователя
13     hashed_password = get_password_hash(user.password)
14     db_user = User(email=user.email, hashed_password=hashed_password)
15     session.add(db_user)
```

```

16 session.commit()
17 session.refresh(db_user)
18
19 # Генерация JWTтокена-
20 access_token = create_access_token(
21     data={"sub": str(db_user.id)},
22     expires_delta=timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
23 )
24 return {"access_token": access_token, "token_type": "bearer"}
25
26 @app.post("/token", response_model=Token, summaryПолучение=" JWTтокена-")
27 def login_for_access_token(
28     form_data: OAuth2PasswordRequestForm = Depends(),
29     session: Session = Depends(get_session)
30 ):
31     Аутентифицирует"" пользователя и возвращает JWTтокен-""
32     # Поиск пользователя по email
33     user = session.query(User).filter(User.email == form_data.username).first()
34
35     # Проверка существования пользователя и соответствия пароля
36     if not user or not verify_password(form_data.password, user.hashed_password):
37         raise HTTPException(
38             status_code=status.HTTP_401_UNAUTHORIZED,
39             detail="Incorrect email or password",
40             headers={"WWW-Authenticate": "Bearer"},
41         )
42
43     # Генерация JWTтокена-
44     access_token = create_access_token(
45         data={"sub": str(user.id)},
46         expires_delta=timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
47     )
48     return {"access_token": access_token, "token_type": "bearer"}

```

Listing 4: auth_endpoints.py

Тест в Postman:

1. **POST** /register

```

1 {
2     "email": "test@example.com",
3     "password": "securepassword123"
4 }

```

→ получаете access_token

2. **POST** /token (Content-Type: application/x-www-form-urlencoded)

- username: test@example.com
- password: securepassword123

→ получаете токен

2.4 Этап 4. Защитить эндпоинты заметок

Цель: Только авторизованные пользователи могут работать с заметками.

2.4.1 Обновите все CRUD-эндпоинты в main.py

Добавьте параметр `current_user: User = Depends(get_current_user)` во все эндпоинты заметок и фильтруйте по `owner_id`:

```

1 # Создание заметки защищено()
2 @app.post("/api/notes/", response_model=NoteOut, summaryСоздание=" заметки")
3 def create_note(
4     note: NoteCreate,
5     session: Session = Depends(get_session),
6     current_user: User = Depends(get_current_user)
7 ):
8     Создает"" новую заметку для текущего пользователя""

```

```

9     db_note = Note(
10         title=note.title,
11         content=note.content,
12         owner_id=current_user.id # Привязка к текущему пользователю
13     )
14     session.add(db_note)
15     session.commit()
16     session.refresh(db_note)
17     return db_note
18
19 # Получение всех заметок только( свои)
20 @app.get("/api/notes/", response_model=list[NoteOut], summaryПолучение=" всех заметок")
21 def get_all_notes(
22     session: Session = Depends(get_session),
23     current_user: User = Depends(get_current_user)
24 ):
25     Возвращает"" все заметки текущего пользователя""
26     return session.query(Note).filter(Note.owner_id == current_user.id).all()
27
28 # Получение одной заметки
29 @app.get("/api/notes/{item_id}", response_model=NoteOut, summaryПолучение=" заметки по ID
30 ")
31 def get_note(
32     item_id: int,
33     session: Session = Depends(get_session),
34     current_user: User = Depends(get_current_user)
35 ):
36     Возвращает"" заметку по ID, только если она принадлежит текущему пользователю""
37     note = session.query(Note).filter(
38         Note.id == item_id,
39         Note.owner_id == current_user.id
40     ).first()
41
42     if not note:
43         raise HTTPException(
44             status_code=status.HTTP_404_NOT_FOUND,
45             detail="Note not found or not owned by you"
46         )
47     return note
48
49 # Обновление заметки
50 @app.patch("/api/notes/{item_id}", response_model=NoteOut, summaryОбновление=" заметки")
51 def update_note(
52     item_id: int,
53     note_update: NoteCreate,
54     session: Session = Depends(get_session),
55     current_user: User = Depends(get_current_user)
56 ):
57     Обновляет"" заметку, только если она принадлежит текущему пользователю""
58     note = session.query(Note).filter(
59         Note.id == item_id,
60         Note.owner_id == current_user.id
61     ).first()
62
63     if not note:
64         raise HTTPException(
65             status_code=status.HTTP_404_NOT_FOUND,
66             detail="Note not found or not owned by you"
67         )
68
69     note.title = note_update.title
70     note.content = note_update.content
71     session.commit()
72     session.refresh(note)
73     return note
74
75 # Удаление заметки
76 @app.delete("/api/notes/{item_id}", summaryУдаление=" заметки")
77 def delete_note(
78     item_id: int,
79     session: Session = Depends(get_session),
80     current_user: User = Depends(get_current_user)
81 ):

```

```

81     Удаляет"" заметку, только если она принадлежит текущему пользователю""
82     note = session.query(Note).filter(
83         Note.id == item_id,
84         Note.owner_id == current_user.id
85     ).first()
86
87     if not note:
88         raise HTTPException(
89             status_code=status.HTTP_404_NOT_FOUND,
90             detail="Note not found or not owned by you"
91         )
92
93     session.delete(note)
94     session.commit()
95     return {"detail": "Note deleted successfully"}

```

Listing 5: protected_endpoints.py

Тест в Postman:

1. Получите токен через [/token](#)
2. **POST** [/api/notes/](#) с заголовком:

Authorization: Bearer <ваш_токен>
Content-Type: application/json

```

{
  "title": "Тестовая заметка",
  "content": "Содержимое заметки"
}

```

→ заметка создается

3. Без заголовка авторизации → 401 Unauthorized
4. Попытка доступа к чужой заметке → 404 Not Found

2.5 Этап 5. Настроить axios-клиент на автоматическую отправку токена

Цель: Клиент автоматически авторизован, если есть токен.

2.5.1 Обновите api.ts (или ваш файл с axios)

```

1 import axios, { AxiosInstance, InternalAxiosRequestConfig } from "axios";
2
3 // Создание инстанса axios с базовыми настройками
4 const api: AxiosInstance = axios.create({
5   baseURL: "/api",
6   timeout: 10000,
7   headers: {
8     "Content-Type": "application/json",
9   },
10 });
11
12 // Интерцептор для добавления токена в каждый запрос
13 api.interceptors.request.use((config: InternalAxiosRequestConfig) => {
14   const token = localStorage.getItem("access_token");
15   if (token) {
16     config.headers.Authorization = `Bearer ${token}`;
17   }
18   return config;
19 });
20
21 // Интерцептор для обработки ошибок 401 неавторизован()
22 api.interceptors.response.use(
23   (response) => response,
24   (error) => {

```

```

25     if (error.response?.status === 401) {
26         // Удаляем токен при ошибке 401
27         localStorage.removeItem("access_token");
28
29         // Можно добавить редирект на страницу входа
30         window.location.href = "/login";
31     }
32     return Promise.reject(error);
33 }
34 );
35
36 export default api;

```

Listing 6: api.ts

Проверка: убедитесь, что файл обновлен и нет ошибок компиляции TypeScript.

2.6 Этап 6. Добавить аутентификационный сервис и UI-страницы

Цель: Пользователь может зайти через браузер.

2.6.1 Создайте auth.ts

```

1 import api from "./api";
2
3 /**
4  * Аутентифицирует пользователя и сохраняет токен
5  * @param email Email пользователя
6  * @param password Пароль пользователя
7  */
8 export const login = async (email: string, password: string): Promise<void> => {
9     try {
10         const response = await api.post<{ access_token: string }>("/token",
11             new URLSearchParams({
12                 username: email,
13                 password: password,
14             }),
15             {
16                 headers: {
17                     "Content-Type": "application/x-www-form-urlencoded",
18                 },
19             }
20         );
21
22         localStorage.setItem("access_token", response.data.access_token);
23     } catch (error) {
24         console.error("Login failed:", error);
25         throw error;
26     }
27 };
28
29 /**
30  * Регистрирует нового пользователя
31  * @param email Email пользователя
32  * @param password Пароль пользователя
33  */
34 export const register = async (email: string, password: string): Promise<void> => {
35     try {
36         await api.post("/register", { email, password });
37     } catch (error) {
38         console.error("Registration failed:", error);
39         throw error;
40     }
41 };
42
43 /** Выход из системы */
44 export const logout = (): void => {
45     localStorage.removeItem("access_token");
46 };
47

```



```

48 /** Проверяет, авторизован ли пользователь */
49 export const isAuthenticated = (): boolean => {
50   return !!localStorage.getItem("access_token");
51 };

```

Listing 7: auth.ts

2.6.2 Создайте Login.tsx

```

1 import { useState } from "react";
2 import { useNavigate } from "react-router-dom";
3 import { Box, TextField, Button, Typography, Alert } from "@mui/material";
4 import { login } from "./auth";
5
6 export default function Login() {
7   const [email, setEmail] = useState("");
8   const [password, setPassword] = useState("");
9   const [error, setError] = useState("");
10  const [loading, setLoading] = useState(false);
11  const navigate = useNavigate();
12
13  const handleSubmit = async (e: React.FormEvent) => {
14    e.preventDefault();
15    setLoading(true);
16    setError("");
17
18    try {
19      await login(email, password);
20      navigate("/notes", { replace: true });
21    } catch (err) {
22      setErrorНеверный(" email или пароль. Попробуйте еще раз.");
23    } finally {
24      setLoading(false);
25    }
26  };
27
28  return (
29    <Box sx={{ maxWidth: 400, mx: "auto", mt: 8, p: 3 }}>
30      <Typography variant="h4" gutterBottom>
31        Вход в систему
32      </Typography>
33
34      {error && (
35        <Alert severity="error" sx={{ mb: 2 }}>
36          {error}
37        </Alert>
38      )}
39
40      <Box component="form" onSubmit={handleSubmit} noValidate>
41        <TextField
42          margin="normal"
43          required
44          fullWidth
45          label="Email"
46          type="email"
47          value={email}
48          onChange={(e) => setEmail(e.target.value)}
49          disabled={loading}
50          autoFocus
51        />
52
53        <TextField
54          margin="normal"
55          required
56          fullWidth
57          label="Пароль"
58          type="password"
59          value={password}
60          onChange={(e) => setPassword(e.target.value)}
61          disabled={loading}

```

```

62     />
63
64     <Button
65       type="submit"
66       fullWidth
67       variant="contained"
68       disabled={loading}
69       sx={{ mt: 3, mb: 2 }}
70     >
71       {loading ? Вход"..." : Войти""}
72     </Button>
73
74     <Typography variant="body2" align="center">
75       Нет аккаунта?{" "}
76       <Button
77         variant="text"
78         onClick={() => navigate("/register")}
79         disabled={loading}
80       >
81         Зарегистрироваться
82       </Button>
83     </Typography>
84   </Box>
85 </Box>
86 );
87 }

```

Listing 8: Login.tsx

2.6.3 Создайте Register.tsx

```

1 import { useState } from "react";
2 import { useNavigate } from "react-router-dom";
3 import { Box, TextField, Button, Typography, Alert } from "@mui/material";
4 import { register, login } from "../auth";
5
6 export default function Register() {
7   const [email, setEmail] = useState("");
8   const [password, setPassword] = useState("");
9   const [confirmPassword, setConfirmPassword] = useState("");
10  const [error, setError] = useState("");
11  const [loading, setLoading] = useState(false);
12  const navigate = useNavigate();
13
14  const handleSubmit = async (e: React.FormEvent) => {
15    e.preventDefault();
16    setLoading(true);
17    setError("");
18
19    if (password !== confirmPassword) {
20      setErrorПароли(" не совпадают");
21      setLoading(false);
22      return;
23    }
24
25    try {
26      // Регистрация пользователя
27      await register(email, password);
28
29      // Автоматический вход после регистрации
30      await login(email, password);
31
32      navigate("/notes", { replace: true });
33    } catch (err: any) {
34      setError(err.response?.data?.detail || Ошибка" регистрации. Попробуйте другой email
35      .");
36    } finally {
37      setLoading(false);
38    }
39  };

```

```

39
40 return (
41   <Box sx={{ maxWidth: 400, mx: "auto", mt: 8, p: 3 }}>
42     <Typography variant="h4" gutterBottom>
43       Регистрация
44     </Typography>
45
46     {error && (
47       <Alert severity="error" sx={{ mb: 2 }}>
48         {error}
49       </Alert>
50     )}
51
52     <Box component="form" onSubmit={handleSubmit} noValidate>
53       <TextField
54         margin="normal"
55         required
56         fullWidth
57         label="Email"
58         type="email"
59         value={email}
60         onChange={e => setEmail(e.target.value)}
61         disabled={loading}
62         autoFocus
63       />
64
65       <TextField
66         margin="normal"
67         required
68         fullWidth
69         label="Пароль"
70         type="password"
71         value={password}
72         onChange={e => setPassword(e.target.value)}
73         disabled={loading}
74       />
75
76       <TextField
77         margin="normal"
78         required
79         fullWidth
80         label="Подтвердите пароль"
81         type="password"
82         value={confirmPassword}
83         onChange={e => setConfirmPassword(e.target.value)}
84         disabled={loading}
85       />
86
87       <Button
88         type="submit"
89         fullWidth
90         variant="contained"
91         disabled={loading}
92         sx={{ mt: 3, mb: 2 }}
93       >
94         {loading ? "Регистрация..." : "Зарегистрироваться"}
95     </Button>
96
97     <Typography variant="body2" align="center">
98       Уже есть аккаунт?{" "}
99       <Button
100         variant="text"
101         onClick={() => navigate("/login")}
102         disabled={loading}
103       >
104         Войти
105     </Button>
106   </Typography>
107 </Box>
108 </Box>
109 );
110 }

```

Listing 9: Register.tsx

2.6.4 Создайте ProtectedRoute.tsx

```
1 import { Navigate, Outlet } from "react-router-dom";
2 import { isAuthenticated } from "../auth";
3
4 /**
5  * Компонентобертка- для защиты маршрутов.
6  * Если пользователь не авторизован, перенаправляет на страницу входа.
7  */
8 export default function ProtectedRoute() {
9   return isAuthenticated() ? <Outlet /> : <Navigate to="/login" replace />;
10 }
```

Listing 10: ProtectedRoute.tsx

2.6.5 Обновите router.ts

```
1 import { createBrowserRouter, RouterProvider } from "react-router-dom";
2 import App from "../App";
3 import Notes from "../Notes";
4 import NoteCreate from "../NoteCreate";
5 import NoteEdit from "../NoteEdit";
6 import Login from "../Login";
7 import Register from "../Register";
8 import ProtectedRoute from "../ProtectedRoute";
9
10 export const router = createBrowserRouter([
11   {
12     path: "/",
13     element: <App />,
14     children: [
15       // Публичные маршруты
16       { path: "/login", element: <Login /> },
17       { path: "/register", element: <Register /> },
18
19       // Защищенные маршруты
20       {
21         element: <ProtectedRoute />,
22         children: [
23           { index: true, element: <Notes /> },
24           {
25             path: "notes",
26             children: [
27               { index: true, element: <Notes /> },
28               { path: "create", element: <NoteCreate /> },
29               { path: ":id/edit", element: <NoteEdit /> },
30             ],
31           },
32         ],
33       },
34     ],
35   },
36 ]);
37
38 // В main.tsx или index.tsx:
39 // ReactDOM.createRoot(document.getElementById('root')).render(
40 //   <React.StrictMode>
41 //     <RouterProvider router={router} />
42 //   </React.StrictMode>
43 // );
```

Listing 11: router.ts

Тест в браузере:

1. Зайдите на [/](#) → перенаправляет на </login>
2. Зарегистрируйтесь → автоматический вход → переход на </notes>
3. Создайте заметку — она сохраняется и отображается
4. Обновите страницу — авторизация сохраняется (токен в localStorage)

2.7 Этап 7. Добавить кнопку «Выйти»

Цель: Возможность выхода из системы.

2.7.1 Обновите App.tsx

```

1 import { Outlet, useNavigate } from "react-router-dom";
2 import { Button, Box, AppBar, Toolbar, Typography } from "@mui/material";
3 import { isAuthenticated, logout } from "./auth";
4
5 export default function App() {
6   const navigate = useNavigate();
7
8   const handleLogout = () => {
9     logout();
10    navigate("/login", { replace: true });
11  };
12
13  return (
14    <Box sx={{ display: "flex", flexDirection: "column", minHeight: "100vh" }}>
15      <AppBar position="static" color="primary">
16        <Toolbar>
17          <Typography variant="h6" sx={{ flexGrow: 1 }}>
18            Менеджер заметок
19          </Typography>
20          {isAuthenticated() && (
21            <Button
22              color="inherit"
23              onClick={handleLogout}
24              sx={{ "&:hover": { backgroundColor: "rgba(255,255,255,0.1)" } }}
25            >
26              Выйти
27            </Button>
28          )}
29        </Toolbar>
30      </AppBar>
31
32      <Box component="main" sx={{ p: 3, flexGrow: 1 }}>
33        <Outlet />
34      </Box>
35
36      <Box component="footer" sx={{ p: 2, textAlign: "center", bgcolor: "grey.100" }}>
37        <Typography variant="body2" color="text.secondary">
38          © {new Date().getFullYear()} Менеджер заметок. Все права защищены.
39        </Typography>
40      </Box>
41    </Box>
42  );
43 }

```

Listing 12: App.tsx

Тест:

- После входа в шапке появляется кнопка «Выйти»
- Клик по кнопке → выход и редирект на </login>
- После выхода доступ к </notes> невозможен без входа

Компонент	Состояние
База данных	Поддерживает пользователей и привязку заметок через owner_id
FastAPI сервер	Реализует полный цикл аутентификации: регистрация, вход, JWT, защита эндпоинтов
React клиент	Автоматическая авторизация через axios interceptors, защита маршрутов, UI для управления аутентификацией
Postman	Все эндпоинты протестированы с JWT-токенами
Безопасность	Пароли хешируются, токены имеют ограниченное время жизни, разделение прав доступа

Таблица 1: Состояние системы после внедрения JWT-аутентификации

3 Финальная архитектура системы

4 Проверка корректности и тестирование

4.1 Тестирование через Postman

1. **Регистрация:** **POST** /register с корректными данными → получение токена
2. **Вход:** **POST** /token с корректными учетными данными → получение токена
3. **Создание заметки:** **POST** /api/notes/ с токеном → успешное создание
4. **Попытка доступа без токена:** любой защищенный эндпоинт без заголовка → 401 Unauthorized
5. **Попытка доступа к чужой заметке:** запрос к заметке другого пользователя → 404 Not Found
6. **Истечение срока действия токена:** после 30 минут запрос с токеном → 401 Unauthorized

4.2 Тестирование в браузере

1. **Первый вход:** переход на / → редирект на /login
2. **Регистрация:** успешная регистрация → автоматический вход
3. **Работа с заметками:** создание, редактирование, удаление работают корректно
4. **Обновление страницы:** авторизация сохраняется (токен в localStorage)
5. **Выход:** кнопка «Выйти» работает, редирект на /login
6. **Попытка доступа к защищенным маршрутам после выхода:** редирект на /login

5 Заключение

Система аутентификации и авторизации успешно внедрена в приложение. Ключевые достижения:

- **Безопасность:** пароли хешируются с использованием bcrypt, JWT-токены имеют ограниченное время жизни
- **Разделение прав доступа:** пользователи видят только свои заметки
- **Удобство использования:** автоматическая авторизация в клиенте, интуитивно понятный UI
- **Масштабируемость:** архитектура позволяет легко добавлять новые функции без-опасности (роли, refresh-токены)
- **Тестируемость:** каждый этап был протестирован отдельно через Postman и браузер

Дальнейшие улучшения могут включать:

- Добавление refresh-токенов для продления сессии без повторного входа
- Реализация подтверждения email при регистрации

- Добавление двухфакторной аутентификации
- Логирование событий безопасности
- Rate limiting для эндпоинтов аутентификации

Данное руководство обеспечивает надежную основу для безопасного веб-приложения с современной аутентификацией на основе JWT.