Пошаговая реализация JWT-аутентификации для FastAPI + React

Руководство по внедрению

22 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

- 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`)

```
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column

class Base(DeclarativeBase):
    pass

class User(Base):
    __tablename__ = "users"
    id: Mapped[int] = mapped_column(primary_key=True)
    email: Mapped[str] = mapped_column(unique=True, index=True)
    hashed_password: Mapped[str]

class Note(Base):
    __tablename__ = "notes"
    id: Mapped[int] = mapped_column(primary_key=True)
    title: Mapped[str]
    content: Mapped[str]
    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):

```
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

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

2.2.2 Создайте `security.py`

```python
from datetime import datetime, timedelta, timezone
from typing import Optional, Dict
from jose import jwt
from passlib.context import CryptContext

# Настройки безопасности в( продакшене используйте переменные окружения)
SECRET_KEY = "your-secret-key-change-in-production-2025"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def verify_password(plain_password: str, hashed_password: str) -> bool:
    Проверяет""" соответствие пароля хешу"""
    return pwd_context.verify(plain_password, hashed_password)

def get_password_hash(password: str) -> str:
    Генерирует""" хеш пароля"""
    return pwd_context.hash(password)

def create_access_token(data: Dict[str, str], expires_delta: Optional[timedelta] = None)
    -> str:
    Создает""" JWTтокен- с указанным временем жизни"""
    to_encode = data.copy()
    expire = datetime.now(timezone.utc) + (expires_delta or timedelta(minutes=
    ACCESS_TOKEN_EXPIRE_MINUTES))
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
```

Listing 2: security.py

2.2.3 Обновите `validation_models.py`

```python
from pydantic import BaseModel, EmailStr, Field, ConfigDict
from typing import Annotated, Optional

# Типы с валидацией
TitleType = Annotated[str, Field(min_length=1, max_length=100)]
ContentType = Annotated[str, Field(min_length=1)]
IdType = Annotated[int, Field(gt=0)]

class NoteBase(BaseModel):
    title: TitleType
    content: ContentType

class NoteCreate(NoteBase):
    pass

class NoteOut(NoteBase):
    id: IdType
    owner_id: IdType

    model_config = ConfigDict(from_attributes=True)

# Схемы для аутентификации
class UserCreate(BaseModel):
    email: EmailStr
    password: str = Field(min_length=6, max_length=100)

class Token(BaseModel):
    access_token: str
    token_type: str = "bearer"

class TokenData(BaseModel):
    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:

```
from fastapi import Depends, HTTPException, status, APIRouter
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm
from sqlalchemy.orm import Session
from jose import JWTError, jwt

# Импорты локальных модулей
from database import get_session, SessionLocal
from database_models import User, Note
from validation_models import NoteOut, NoteCreate, UserCreate, Token
from security import (
    verify_password,
    get_password_hash,
    create_access_token,
    SECRET_KEY,
    ALGORITHM,
    ACCESS_TOKEN_EXPIRE_MINUTES
)

# OAuth2 схема для извлечения токена из заголовка
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

def get_current_user(
    token: str = Depends(oauth2_scheme),
    session: Session = Depends(get_session)
) -> User:
    Извлекает""" текущего пользователя из JWTтокена-"""
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        user_id: str = payload.get("sub")
        if user_id is None:
            raise credentials_exception
    except JWTError:
        raise credentials_exception

    user = session.query(User).filter(User.id == int(user_id)).first()
    if user is None:
        raise credentials_exception
    return user
```

Теперь добавьте эндпоинты аутентификации:

```
# Эндпоинты аутентификации
@app.post("/register", response_model=Token, summaryРегистрация=" нового пользователя")
def register(user: UserCreate, session: Session = Depends(get_session)):
    Регистрирует""" нового пользователя и возвращает JWTтокен-"""
    # Проверка на существование email
    if session.query(User).filter(User.email == user.email).first():
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Email already registered"
        )

    # Хеширование пароля и сохранение пользователя
    hashed_password = get_password_hash(user.password)
    db_user = User(email=user.email, hashed_password=hashed_password)
    session.add(db_user)
```

```
    session.commit()
    session.refresh(db_user)

    # Генерация JWTтокена-
    access_token = create_access_token(
        data={"sub": str(db_user.id)},
        expires_delta=timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    )
    return {"access_token": access_token, "token_type": "bearer"}

@app.post("/token", response_model=Token, summaryПолучение=" JWTтокена-")
def login_for_access_token(
    form_data: OAuth2PasswordRequestForm = Depends(),
    session: Session = Depends(get_session)
):
    Аутентифицирует""" пользователя и возвращает JWTтокен-"""
    # Поиск пользователя по email
    user = session.query(User).filter(User.email == form_data.username).first()

    # Проверка существования пользователя и соответствия пароля
    if not user or not verify_password(form_data.password, user.hashed_password):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect email or password",
            headers={"WWW-Authenticate": "Bearer"},
        )

    # Генерация JWTтокена-
    access_token = create_access_token(
        data={"sub": str(user.id)},
        expires_delta=timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    )
    return {"access_token": access_token, "token_type": "bearer"}
```

Listing 4: auth_endpoints.py

Тест в Postman:

1. **POST** `/register`

```
{
    "email": "test@example.com",
    "password": "securepassword123"
}
```

   → 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`:

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

```
    db_note = Note(
        title=note.title,
        content=note.content,
        owner_id=current_user.id  # Привязка к текущему пользователю
    )
    session.add(db_note)
    session.commit()
    session.refresh(db_note)
    return db_note

# Получение всех заметок только( свои)
@app.get("/api/notes/", response_model=list[NoteOut], summaryПолучение=" всех заметок")
def get_all_notes(
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_user)
):
    Возвращает""" все заметки текущего пользователя"""
    return session.query(Note).filter(Note.owner_id == current_user.id).all()

# Получение одной заметки
@app.get("/api/notes/{item_id}", response_model=NoteOut, summaryПолучение=" заметки по ID
    ")
def get_note(
    item_id: int,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_user)
):
    Возвращает""" заметку по ID, только если она принадлежит текущему пользователю"""
    note = session.query(Note).filter(
        Note.id == item_id,
        Note.owner_id == current_user.id
    ).first()

    if not note:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Note not found or not owned by you"
        )
    return note

# Обновление заметки
@app.patch("/api/notes/{item_id}", response_model=NoteOut, summaryОбновление=" заметки")
def update_note(
    item_id: int,
    note_update: NoteCreate,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_user)
):
    Обновляет""" заметку, только если она принадлежит текущему пользователю"""
    note = session.query(Note).filter(
        Note.id == item_id,
        Note.owner_id == current_user.id
    ).first()

    if not note:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Note not found or not owned by you"
        )

    note.title = note_update.title
    note.content = note_update.content
    session.commit()
    session.refresh(note)
    return note

# Удаление заметки
@app.delete("/api/notes/{item_id}", summaryУдаление=" заметки")
def delete_note(
    item_id: int,
    session: Session = Depends(get_session),
    current_user: User = Depends(get_current_user)
):
```

```
    Удаляет""" заметку, только если она принадлежит текущему пользователю"""
    note = session.query(Note).filter(
        Note.id == item_id,
        Note.owner_id == current_user.id
    ).first()

    if not note:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Note not found or not owned by you"
        )

    session.delete(note)
    session.commit()
    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)

```
import axios, { AxiosInstance, InternalAxiosRequestConfig } from "axios";

// Создание инстанса axios с базовыми настройками
const api: AxiosInstance = axios.create({
  baseURL: "/api",
  timeout: 10000,
  headers: {
    "Content-Type": "application/json",
  },
});

// Интерцептор для добавления токена в каждый запрос
api.interceptors.request.use((config: InternalAxiosRequestConfig) => {
  const token = localStorage.getItem("access_token");
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Интерцептор для обработки ошибок 401 неавторизован()
api.interceptors.response.use(
  (response) => response,
  (error) => {
```

```
    if (error.response?.status === 401) {
      // Удаляем токен при ошибке 401
      localStorage.removeItem("access_token");

      // Можно добавить редирект на страницу входа
      window.location.href = "/login";
    }
    return Promise.reject(error);
  }
);

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`

```
import api from "./api";

/**
 * Аутентифицирует пользователя и сохраняет токен
 * @param email Email пользователя
 * @param password Пароль пользователя
 */
export const login = async (email: string, password: string): Promise<void> => {
  try {
    const response = await api.post<{ access_token: string }>("/token",
      new URLSearchParams({
        username: email,
        password: password,
      }),
      {
        headers: {
          "Content-Type": "application/x-www-form-urlencoded",
        },
      }
    );

    localStorage.setItem("access_token", response.data.access_token);
  } catch (error) {
    console.error("Login failed:", error);
    throw error;
  }
};

/**
 * Регистрирует нового пользователя
 * @param email Email пользователя
 * @param password Пароль пользователя
 */
export const register = async (email: string, password: string): Promise<void> => {
  try {
    await api.post("/register", { email, password });
  } catch (error) {
    console.error("Registration failed:", error);
    throw error;
  }
};

/** Выход из системы */
export const logout = (): void => {
  localStorage.removeItem("access_token");
};

```

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

Listing 7: auth.ts

2.6.2 Создайте `Login.tsx`

```
import { useState } from "react";
import { useNavigate } from "react-router-dom";
import { Box, TextField, Button, Typography, Alert } from "@mui/material";
import { login } from "./auth";

export default function Login() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(false);
  const navigate = useNavigate();

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setLoading(true);
    setError("");

    try {
      await login(email, password);
      navigate("/notes", { replace: true });
    } catch (err) {
      setErrorНеверный(" email или пароль. Попробуйте еще раз.");
    } finally {
      setLoading(false);
    }
  };

  return (
    <Box sx={{ maxWidth: 400, mx: "auto", mt: 8, p: 3 }}>
      <Typography variant="h4" gutterBottom>
        Вход в систему
      </Typography>

      {error && (
        <Alert severity="error" sx={{ mb: 2 }}>
          {error}
        </Alert>
      )}

      <Box component="form" onSubmit={handleSubmit} noValidate>
        <TextField
          margin="normal"
          required
          fullWidth
          label="Email"
          type="email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          disabled={loading}
          autoFocus
        />

        <TextField
          margin="normal"
          required
          fullWidth
          labelПароль=""
          type="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          disabled={loading}
```

```
        />

        <Button
          type="submit"
          fullWidth
          variant="contained"
          disabled={loading}
          sx={{ mt: 3, mb: 2 }}
        >
          {loading ? Вход"..." : Войти""}
        </Button>

        <Typography variant="body2" align="center">
          Нет аккаунта?{" "}
          <Button
            variant="text"
            onClick={() => navigate("/register")}
            disabled={loading}
          >
            Зарегистрироваться
          </Button>
        </Typography>
      </Box>
    </Box>
  );
}
```

Listing 8: Login.tsx

2.6.3 Создайте `Register.tsx`

```
import { useState } from "react";
import { useNavigate } from "react-router-dom";
import { Box, TextField, Button, Typography, Alert } from "@mui/material";
import { register, login } from "./auth";

export default function Register() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [confirmPassword, setConfirmPassword] = useState("");
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(false);
  const navigate = useNavigate();

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    setLoading(true);
    setError("");

    if (password !== confirmPassword) {
      setErrorПароли(" не совпадают");
      setLoading(false);
      return;
    }

    try {
      // Регистрация пользователя
      await register(email, password);

      // Автоматический вход после регистрации
      await login(email, password);

      navigate("/notes", { replace: true });
    } catch (err: any) {
      setError(err.response?.data?.detail || Ошибка" регистрации. Попробуйте другой email
    ."); } finally {
      setLoading(false);
    }
  };
```

```
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            labelПароль=""
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            labelПодтвердите=" 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`

```tsx
import { Navigate, Outlet } from "react-router-dom";
import { isAuthenticated } from "./auth";

/**
 * Компонентобертка- для защиты маршрутов.
 * Если пользователь не авторизован, перенаправляет на страницу входа.
 */
export default function ProtectedRoute() {
  return isAuthenticated() ? <Outlet /> : <Navigate to="/login" replace />;
}
```

Listing 10: ProtectedRoute.tsx

2.6.5 Обновите `router.ts`

```tsx
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import App from "./App";
import Notes from "./Notes";
import NoteCreate from "./NoteCreate";
import NoteEdit from "./NoteEdit";
import Login from "./Login";
import Register from "./Register";
import ProtectedRoute from "./ProtectedRoute";

export const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
    children: [
      // Публичные маршруты
      { path: "/login", element: <Login /> },
      { path: "/register", element: <Register /> },

      // Защищенные маршруты
      {
        element: <ProtectedRoute />,
        children: [
          { index: true, element: <Notes /> },
          {
            path: "notes",
            children: [
              { index: true, element: <Notes /> },
              { path: "create", element: <NoteCreate /> },
              { path: ":id/edit", element: <NoteEdit /> },
            ],
          },
        ],
      },
    ],
  },
]);

// В main.tsx или index.tsx:
// ReactDOM.createRoot(document.getElementById('root')!).render(
//   <React.StrictMode>
//     <RouterProvider router={router} />
//   </React.StrictMode>
// );
```

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`

```tsx
import { Outlet, useNavigate } from "react-router-dom";
import { Button, Box, AppBar, Toolbar, Typography } from "@mui/material";
import { isAuthenticated, logout } from "./auth";

export default function App() {
  const navigate = useNavigate();

  const handleLogout = () => {
    logout();
    navigate("/login", { replace: true });
  };

  return (
    <Box sx={{ display: "flex", flexDirection: "column", minHeight: "100vh" }}>
      <AppBar position="static" color="primary">
        <Toolbar>
          <Typography variant="h6" sx={{ flexGrow: 1 }}>
            Менеджер заметок
          </Typography>
          {isAuthenticated() && (
            <Button
              color="inherit"
              onClick={handleLogout}
              sx={{ "&:hover": { backgroundColor: "rgba(255,255,255,0.1)" } }}
            >
              Выйти
            </Button>
          )}
        </Toolbar>
      </AppBar>

      <Box component="main" sx={{ p: 3, flexGrow: 1 }}>
        <Outlet />
      </Box>

      <Box component="footer" sx={{ p: 2, textAlign: "center", bgcolor: "grey.100" }}>
        <Typography variant="body2" color="text.secondary">
          © {new Date().getFullYear()} Менеджер заметок. Все права защищены.
        </Typography>
      </Box>
    </Box>
  );
}
```

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 безопасности (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.