🔐 Руководство по миграции бэкенда на httpOnly cookies

Обзор изменений

Этот документ описывает необходимые изменения в бэкенде для перехода с JWT токенов в localStorage на безопасные httpOnly cookies.

📋 Требуемые изменения

1. Обновление эндпоинта POST /v2/auth/token

Текущее поведение:

```python
@app.post("/v2/auth/token")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    # ... проверка пользователя ...
    access_token = create_access_token(data={"sub": user.email})
    return {"access_token": access_token, "token_type": "bearer"}
```

Новое поведение:

```python
from fastapi import Response
from datetime import timedelta

@app.post("/v2/auth/token")
async def login(
    response: Response,
    form_data: OAuth2PasswordRequestForm = Depends()
):
    # ... проверка пользователя ...
    access_token = create_access_token(data={"sub": user.email})

    # Устанавливаем токен в httpOnly cookie
    response.set_cookie(
        key="access_token",
        value=access_token,
        httponly=True,  # Защита от XSS
        secure=True,    # Только HTTPS (в продакшене)
        samesite="lax", # Защита от CSRF
        max_age=60 * 60 * 24 * 7,  # 7 дней
        path="/"
    )

    # Возвращаем успешный ответ без токена
    return {"message": "Login successful"}
```

2. Обновление middleware для чтения токена из cookies

Текущий код (чтение из Authorization header):

```python
from fastapi.security import OAuth2PasswordBearer

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="/v2/auth/token")

async def get_current_user(token: str = Depends(oauth2_scheme)):
    # ... валидация токена ...
    return user
```

Новый код (чтение из cookies):

```python
from fastapi import Cookie, HTTPException, status
from typing import Optional

async def get_current_user(access_token: Optional[str] = Cookie(None)):
    if not access_token:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Not authenticated"
        )

    try:
        # Декодирование и валидация JWT токена
        payload = jwt.decode(
            access_token,
            SECRET_KEY,
            algorithms=[ALGORITHM]
        )
        email: str = payload.get("sub")
        if email is None:
            raise HTTPException(
                status_code=status.HTTP_401_UNAUTHORIZED,
                detail="Invalid token"
            )
    except JWTError:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid token"
        )

    # Получение пользователя из БД
    user = get_user_by_email(email)
    if user is None:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="User not found"
        )

    return user
```

3. Добавление эндпоинта POST /v2/auth/logout

```python
@app.post("/v2/auth/logout")
async def logout(response: Response):
    """
    Выход из системы - удаление httpOnly cookie
    """
    response.delete_cookie(
        key="access_token",
        path="/",
        httponly=True,
        secure=True,
        samesite="lax"
    )
    return {"message": "Logout successful"}
```

4. Обновление CORS настроек

Критически важно: Необходимо разрешить отправку credentials (cookies) с фронтенда.

```python
from fastapi.middleware.cors import CORSMiddleware

app.add_middleware(
    CORSMiddleware,
    allow_origins=[
        "http://localhost:3000",      # Для разработки
        "https://upak.space",          # Продакшен
        "https://www.upak.space"       # Продакшен с www
    ],
    allow_credentials=True,  # ⚠️ КРИТИЧЕСКИ ВАЖНО!
    allow_methods=["*"],
    allow_headers=["*"],
)
```

⚠️ **ВАЖНО:** При `allow_credentials=True` нельзя использовать `allow_origins=["*"]`. Необходимо явно указать разрешенные домены.

5. Обновление всех защищенных эндпоинтов

Все эндпоинты, требующие аутентификации, должны использовать новую функцию `get_current_user`:

```python
@app.get("/v2/me")
async def get_me(current_user: User = Depends(get_current_user)):
    return {
        "email": current_user.email,
        "subscription_type": current_user.subscription_type,
        "subscription_expires": current_user.subscription_expires,
        "cards_limit": current_user.cards_limit,
        "cards_used": current_user.cards_used
    }

@app.get("/v2/cards")
async def get_cards(
    current_user: User = Depends(get_current_user),
    limit: int = 20,
    offset: int = 0
):
    # ... логика получения карточек пользователя ...
    return cards

@app.post("/v2/payments/create")
async def create_payment(
    package: str,
    current_user: User = Depends(get_current_user)
):
    # ... логика создания платежа ...
    return payment_data
```

🔒 Настройки безопасности

Параметры cookie

Параметр	Значение	Описание
`httponly`	`True`	Защита от XSS атак - cookie недоступен для JavaScript
`secure`	`True`	Cookie передается только по HTTPS (в продакшене)
`samesite`	`"lax"`	Защита от CSRF атак
`max_age`	`604800`	Время жизни cookie (7 дней в секундах)
`path`	`"/"`	Cookie доступен для всех путей

Для разработки (HTTP)

В режиме разработки на localhost можно использовать `secure=False`:

```python
import os

IS_PRODUCTION = os.getenv("ENVIRONMENT") == "production"

response.set_cookie(
    key="access_token",
    value=access_token,
    httponly=True,
    secure=IS_PRODUCTION,  # False для localhost
    samesite="lax",
    max_age=60 * 60 * 24 * 7,
    path="/"
)
```

📝 Полный пример файла auth.py

```python
from datetime import datetime, timedelta
from typing import Optional
from fastapi import Depends, HTTPException, status, Response, Cookie
from fastapi.security import OAuth2PasswordRequestForm
from jose import JWTError, jwt
from passlib.context import CryptContext
import os

# Настройки JWT
SECRET_KEY = os.getenv("SECRET_KEY", "your-secret-key-here")
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_DAYS = 7

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

def verify_password(plain_password: str, hashed_password: str) -> bool:
    return pwd_context.verify(plain_password, hashed_password)

def get_password_hash(password: str) -> str:
    return pwd_context.hash(password)

def create_access_token(data: dict, expires_delta: Optional[timedelta] = None):
    to_encode = data.copy()
    if expires_delta:
        expire = datetime.utcnow() + expires_delta
    else:
        expire = datetime.utcnow() + timedelta(days=ACCESS_TOKEN_EXPIRE_DAYS)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt

async def get_current_user(access_token: Optional[str] = Cookie(None)):
    """
    Получение текущего пользователя из httpOnly cookie
    """
    if not access_token:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Not authenticated",
            headers={"WWW-Authenticate": "Bearer"},
        )

    try:
        payload = jwt.decode(access_token, SECRET_KEY, algorithms=[ALGORITHM])
        email: str = payload.get("sub")
        if email is None:
            raise HTTPException(
                status_code=status.HTTP_401_UNAUTHORIZED,
                detail="Invalid authentication credentials"
            )
    except JWTError:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid authentication credentials"
        )

    # Здесь должна быть логика получения пользователя из БД
    user = get_user_by_email(email)  # Ваша функция
    if user is None:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="User not found"
```

```
        )

    return user

# Эндпоинты аутентификации
from fastapi import APIRouter

router = APIRouter(prefix="/v2/auth", tags=["auth"])

@router.post("/token")
async def login(
    response: Response,
    form_data: OAuth2PasswordRequestForm = Depends()
):
    """
    Вход в систему - устанавливает httpOnly cookie с JWT токеном
    """
    # Проверка пользователя
    user = authenticate_user(form_data.username, form_data.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect email or password"
        )

    # Создание токена
    access_token = create_access_token(data={"sub": user.email})

    # Установка cookie
    IS_PRODUCTION = os.getenv("ENVIRONMENT") == "production"
    response.set_cookie(
        key="access_token",
        value=access_token,
        httponly=True,
        secure=IS_PRODUCTION,
        samesite="lax",
        max_age=60 * 60 * 24 * ACCESS_TOKEN_EXPIRE_DAYS,
        path="/"
    )

    return {"message": "Login successful"}

@router.post("/logout")
async def logout(response: Response):
    """
    Выход из системы - удаление httpOnly cookie
    """
    response.delete_cookie(
        key="access_token",
        path="/",
        httponly=True,
        secure=os.getenv("ENVIRONMENT") == "production",
        samesite="lax"
    )
    return {"message": "Logout successful"}
```

🧪 Тестирование

1. Тест входа в систему

```
curl -X POST "http://localhost:8000/v2/auth/token" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "username=test@upak.space&password=StrongPass123" \
  -c cookies.txt
```

Проверьте, что в `cookies.txt` появился cookie `access_token`.

2. Тест защищенного эндпоинта

```
curl -X GET "http://localhost:8000/v2/me" \
  -b cookies.txt
```

Должен вернуть данные пользователя.

3. Тест выхода

```
curl -X POST "http://localhost:8000/v2/auth/logout" \
  -b cookies.txt \
  -c cookies.txt
```

Cookie должен быть удален.

🚀 Развертывание

1. Обновите переменные окружения

```
# .env
SECRET_KEY=your-super-secret-key-here-change-this
ENVIRONMENT=production
```

2. Перезапустите сервер

```
# Если используете systemd
sudo systemctl restart upak-backend

# Если используете Docker
docker-compose down
docker-compose up -d --build

# Если используете uvicorn напрямую
pkill -f uvicorn
uvicorn main:app --host 0.0.0.0 --port 8000
```

3. Проверьте HTTPS

Убедитесь, что сервер работает по HTTPS, иначе `secure=True` не позволит установить cookie.

⚠️ Важные замечания

1. **CORS настройки**: `allow_credentials=True` требует явного указания доменов в `allow_origins`
2. **HTTPS обязателен**: В продакшене `secure=True` требует HTTPS
3. **SameSite**: Значение `"lax"` обеспечивает баланс между безопасностью и удобством
4. **Время жизни**: Cookie и JWT токен должны иметь одинаковое время жизни
5. **Path**: Убедитесь, что `path="/"` для доступа ко всем эндпоинтам

📚 Дополнительные ресурсы

- FastAPI Security (https://fastapi.tiangolo.com/tutorial/security/)
- OWASP Cookie Security (https://owasp.org/www-community/controls/SecureCookieAttribute)
- MDN Set-Cookie (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie)

🔗 Связанные изменения

- Frontend PR: https://github.com/Yuriuser1/Upak-frontend-NEW-v3/pull/5