ПРИЛОЖЕНИЕ А

Исходный код основных файлов, которые отвечают за работу приложения

backend/app/auth.py

```
from passlib.context import CryptContext
from jose import JWTError, jwt
from datetime import datetime, timedelta
from . import models
from fastapi import Depends, HTTPException
from fastapi.security import OAuth2PasswordBearer
from sqlalchemy.orm import Session
from .database import get_db

SECRET_KEY = "secret_key"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

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

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="api/login")

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

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

def create_access_token(data: dict):
    to_encode = data.copy()
    expire = datetime.utcnow() +
        ↪ timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
```

```
29
30 def get_current_user(token: str = Depends(oauth2_scheme), db:
    ↪ Session = Depends(get_db)):
31     credentials_exception = HTTPException(
32         status_code=401,
33         detail="Невалидные учетные данные",
34         headers={"WWW-Authenticate": "Bearer"},
35     )
36     try:
37         payload = jwt.decode(token, SECRET_KEY,
            ↪ algorithms=[ALGORITHM])
38         login: str = payload.get("sub")
39         if login is None:
40             raise credentials_exception
41     except JWTError:
42         raise credentials_exception
43
44     user = db.query(models.User).filter(models.User.login ==
       ↪ login).first()
45     if user is None:
46         raise credentials_exception
47     return user
```

backend/app/database.py

```
1  # app/database.py
2
3  from sqlalchemy import create_engine, MetaData
4  from sqlalchemy.ext.declarative import declarative_base
5  from sqlalchemy.orm import sessionmaker
6
7  DATABASE_URL = "postgresql://meebin_user:password@db/meebin_db"
8
9  engine = create_engine(DATABASE_URL)
10 SessionLocal = sessionmaker(autocommit=False, autoflush=False,
    ↪ bind=engine)
```

```
11 metadata = MetaData()
12 Base = declarative_base()
13
14 def get_db():
15     db = SessionLocal()
16     try:
17         yield db
18     finally:
19         db.close()
```

backend/app/main.py

```
1  # app/main.py
2
3  from fastapi import FastAPI
4  from . import models, database
5  from .router import main_router
6
7  app = FastAPI()
8
9  # Создаем все таблицы
10 models.Base.metadata.create_all(bind=database.engine)
11
12 app.include_router(main_router, prefix="/api")
```

backend/app/models.py

```
1 from sqlalchemy import Column, Integer, String, BigInteger,
  ↪ ForeignKey, Float, TIMESTAMP
2 from sqlalchemy.orm import relationship
3 from .database import Base
4
5 class User(Base):
6     __tablename__ = "Users"
7
```

```
 8      id = Column(Integer, primary_key=True, index=True,
        ↪ unique=True)
 9      login = Column(String(50), nullable=False)
10      mail = Column(String(80), nullable=False)
11      password = Column(String(255), nullable=False)
12      name = Column(String(60), nullable=False)
13      lastname = Column(String(60), nullable=False)
14      surname = Column(String(60), nullable=True)
15      birthdate = Column(TIMESTAMP, nullable=False, default='-1')
16      report_counter = Column(BigInteger, default=0, nullable=False)
17      utilized_counter = Column(BigInteger, default=0,
        ↪ nullable=False)
18      rating = Column(Float, nullable=True)
19      city = Column(String(60), nullable=False)
20
21      roles = relationship("Role", secondary="Users_Roles",
        ↪ back_populates="users")
22      called_events = relationship("TrashEvent",
        ↪ foreign_keys="[TrashEvent.caller_id]",
        ↪ back_populates="caller")
23      utilized_events = relationship("TrashEvent",
        ↪ foreign_keys="[TrashEvent.utilizator_id]",
        ↪ back_populates="utilizator")
24
25
26  class TrashEvent(Base):
27      __tablename__ = "TrashEvent"
28
29      id = Column(Integer, primary_key=True, index=True,
        ↪ unique=True)
30      photo_url = Column(String(100), nullable=False)
31      address = Column(String(255), nullable=False)
32      caller_id = Column(BigInteger, ForeignKey("Users.id"),
        ↪ nullable=False)
```

```
33      utilizator_id = Column(BigInteger, ForeignKey("Users.id"),
        ↪ nullable=False)
34      event_status = Column(BigInteger,
        ↪ ForeignKey("TrashStatus.id"), default=0, nullable=False)
35      time_called = Column(TIMESTAMP, nullable=False)
36      time_cleaned = Column(TIMESTAMP, nullable=False)
37      comment = Column(String(255), nullable=False)
38      confirmation_photo_url = Column(String(255), nullable=False)
39      price = Column(BigInteger, nullable=False)
40
41      caller = relationship("User", foreign_keys=[caller_id],
        ↪ back_populates="called_events")
42      utilizator = relationship("User",
        ↪ foreign_keys=[utilizator_id],
        ↪ back_populates="utilized_events")
43      status = relationship("TrashStatus")
44
45
46  class Role(Base):
47      __tablename__ = "Roles"
48
49      id = Column(Integer, primary_key=True, index=True,
        ↪ unique=True)
50      title = Column(String(60), nullable=False)
51
52      users = relationship("User", secondary="Users_Roles",
        ↪ back_populates="roles")
53
54
55  class UsersRoles(Base):
56      __tablename__ = "Users_Roles"
57
58      id_roles = Column(BigInteger, ForeignKey("Roles.id"),
        ↪ primary_key=True)
```

```
59      id_users = Column(BigInteger, ForeignKey("Users.id"),
        ↪ primary_key=True)
60
61
62  class TrashStatus(Base):
63      __tablename__ = "TrashStatus"
64
65      id = Column(Integer, primary_key=True, index=True,
        ↪ unique=True)
66      title = Column(String(60), nullable=False)
67
```

backend/app/router.py

```
 1  from fastapi import APIRouter, Depends, HTTPException
 2  from sqlalchemy.orm import Session
 3  from . import models, schemas, database, auth
 4  from fastapi.security import OAuth2PasswordRequestForm
 5
 6  router = APIRouter()
 7
 8  @router.post("/register", response_model=schemas.UserOut)
 9  def register(user: schemas.UserCreate, db: Session =
    ↪ Depends(database.get_db)):
10      db_user = db.query(models.User).filter(models.User.login ==
        ↪ user.login).first()
11      if db_user:
12          raise HTTPException(status_code=400, detail="Такой
            ↪ пользователь уже зарегистрирован")
13
14      hashed_password = auth.get_password_hash(user.password)
15      new_user = models.User(
16          login=user.login,
17          mail=user.mail,
18          password=hashed_password,
```

```
19          name=user.name,
20          lastname=user.lastname,
21          surname=user.surname,
22          birthdate=user.birthdate,
23          city=user.city
24      )
25
26      db.add(new_user)
27      db.commit()
28      db.refresh(new_user)
29
30      return new_user
31
32  @router.post("/login")
33  def login(form_data: OAuth2PasswordRequestForm = Depends(), db:
    ↪  Session = Depends(database.get_db)):
34      user = db.query(models.User).filter(models.User.login ==
        ↪  form_data.username).first()
35      if not user or not auth.verify_password(form_data.password,
        ↪  user.password):
36          raise HTTPException(status_code=400, detail="Неверный
            ↪  логин или пароль")
37
38      access_token = auth.create_access_token(data={"sub":
        ↪  user.login})
39      return {"access_token": access_token, "token_type":
        ↪  "bearer"}
40
41
42  # Включаем маршруты заявок
43  from .routers import events, users
44
45  main_router = APIRouter()
46  main_router.include_router(router, prefix="/auth",
    ↪  tags=["Authentication"])
```

```
47 main_router.include_router(events.router)
48 main_router.include_router(users.router)
49
```

backend/app/schemas.py

```
 1 from pydantic import BaseModel, EmailStr, Field
 2 from typing import Optional
 3 from datetime import datetime
 4
 5 class UserBase(BaseModel):
 6     login: str
 7     mail: EmailStr
 8     name: str
 9     lastname: str
10     surname: Optional[str] = None
11     birthdate: datetime
12     city: str
13
14 class UserCreate(UserBase):
15     password: str
16
17 class UserOut(UserBase):
18     id: int
19
20     class Config:
21         orm_mode = True
22
23 class TrashEventBase(BaseModel):
24     photo_url: str
25     address: str
26     event_status: int
27     time_called: datetime
28     time_cleaned: datetime
29     comment: str
```

```
30      confirmation_photo_url: str
31      price: int
32
33  class TrashEventCreate(TrashEventBase):
34      utilizator_id: int
35
36  class TrashEventOut(TrashEventBase):
37      id: int
38      caller_id: int
39      utilizator_id: int
40
41      class Config:
42          orm_mode = True
```

backend/app/routers/events.py

```
1   # app/routers/events.py
2
3   from fastapi import APIRouter, Depends, HTTPException, status
4   from sqlalchemy.orm import Session
5   from typing import List
6   from datetime import timedelta
7   from .. import models, schemas, database, auth
8   from fastapi.security import OAuth2PasswordBearer
9   from jose import JWTError, jwt
10
11  router = APIRouter(prefix="/events", tags=["Events"])
12
13  oauth2_scheme = OAuth2PasswordBearer(tokenUrl="api/login")
14
15  def get_db():
16      db = database.SessionLocal()
17      try:
18          yield db
19      finally:
```

```
20          db.close()
21
22  # Функция для получения текущего пользователя
23  def get_current_user(token: str = Depends(oauth2_scheme), db:
    ↪  Session = Depends(get_db)):
24      credentials_exception = HTTPException(
25          status_code=status.HTTP_401_UNAUTHORIZED,
26          detail="Невалидные учетные данные",
27          headers={"WWW-Authenticate": "Bearer"},
28      )
29      try:
30          payload = jwt.decode(token, auth.SECRET_KEY,
            ↪  algorithms=[auth.ALGORITHM])
31          username: str = payload.get("sub")
32          if username is None:
33              raise credentials_exception
34      except JWTError:
35          raise credentials_exception
36      user = db.query(models.User).filter(models.User.username ==
        ↪  username).first()
37      if user is None:
38          raise credentials_exception
39      return user
40
41  @router.post("/", response_model=schemas.TrashEventOut)
42  def create_event(
43      event: schemas.TrashEventCreate,
44      db: Session = Depends(database.get_db),
45      current_user: models.User = Depends(auth.get_current_user)
46      ):
47      new_event = models.TrashEvent(
48          photo_url=event.photo_url,
49          address=event.address,
50          caller_id=current_user.id,
51          utilizator_id=event.utilizator_id,
```

```
52          event_status=event.event_status,
53          time_called=event.time_called,
54          time_cleaned=event.time_cleaned,
55          comment=event.comment,
56          confirmation_photo_url=event.confirmation_photo_url,
57          price=event.price
58      )
59      db.add(new_event)
60      db.commit()
61      db.refresh(new_event)
62      return new_event
63
64  @router.get("/", response_model=List[schemas.TrashEventOut])
65  def get_available_events(db: Session = Depends(get_db)):
66      events =
        ↪ db.query(models.TrashEvent).filter(models.TrashEvent.status
        ↪ == models.TrashStatus.available).all()
67      return events
68
69  @router.get("/my", response_model=List[schemas.TrashEventOut])
70  def get_my_events(db: Session = Depends(get_db), current_user:
    ↪ models.User = Depends(get_current_user)):
71      events = db.query(models.TrashEvent).filter(models.TrashEvent.
72      creator_id == current_user.id).all()
73      return events
74
75  @router.get("/accepted",
    ↪ response_model=List[schemas.TrashEventOut])
76  def get_accepted_events(db: Session = Depends(get_db),
    ↪ current_user: models.User = Depends(get_current_user)):
77      events = db.query(models.TrashEvent).filter(models.TrashEvent.
78      accepted_by == current_user.id).all()
79      return events
80
```

```
81 @router.get("/completed",
    ↪ response_model=List[schemas.TrashEventOut])
82 def get_completed_events(db: Session = Depends(get_db),
    ↪ current_user: models.User = Depends(get_current_user)):
83     events =
        ↪ db.query(models.TrashEvent).filter(models.TrashEvent.status
        ↪ == models.TrashStatus.completed,
        ↪ models.TrashEvent.accepted_by == current_user.id).all()
84     return events
85
86 @router.post("/{event_id}/accept",
    ↪ response_model=schemas.TrashEventOut)
87 def accept_event(event_id: int, db: Session = Depends(get_db),
    ↪ current_user: models.User = Depends(get_current_user)):
88     event =
        ↪ db.query(models.TrashEvent).filter(models.TrashEvent.id ==
        ↪ event_id, models.TrashEvent.status ==
        ↪ models.TrashStatus.available).first()
89     if not event:
90         raise HTTPException(status_code=404, detail="Заявка не
            ↪ найдена или уже принята")
91     event.status = models.TrashStatus.accepted
92     event.accepted_by = current_user.id
93     db.commit()
94     db.refresh(event)
95     return event
96
97 @router.post("/{event_id}/complete",
    ↪ response_model=schemas.TrashEventOut)
98 def complete_event(event_id: int, db: Session = Depends(get_db),
    ↪ current_user: models.User = Depends(get_current_user)):
```

```
99      event =
        ↪ db.query(models.TrashEvent).filter(models.TrashEvent.id ==
        ↪ event_id, models.TrashEvent.accepted_by ==
        ↪ current_user.id).first()
100     if not event:
101         raise HTTPException(status_code=404, detail="Заявка не
            ↪ найдена или вы её не приняли")
102     event.status = models.TrashStatus.completed
103     db.commit()
104     db.refresh(event)
105     return event
106
107 # Получить историю заявок пользователя
108 @router.get("/users/{user_id}/history",
    ↪ response_model=List[schemas.TrashEventOut])
109 def get_user_request_history(user_id: int, db: Session =
    ↪ Depends(get_db)):
110     requests = db.query(models.TrashEvent).filter(
111         (models.TrashEvent.caller_id == user_id) |
            ↪ (models.TrashEvent.utilizator_id == user_id),
112         models.TrashEvent.status == models.TrashStatus.completed
113     ).all()
114     return requests
```

backend/app/routers/users.py

```
1  from fastapi import APIRouter, Depends, HTTPException, status
2  from sqlalchemy.orm import Session
3  from .. import models, database, schemas, auth
4  from typing import List
5
6  router = APIRouter(prefix="/users", tags=["Users"])
7
8  # Получить всех пользователей
9  @router.get("/", response_model=List[schemas.UserOut])
```

```
10 def get_users(db: Session = Depends(database.get_db)):
11     users = db.query(models.User).all()
12     return users
13
14 @router.get("/{user_id}")
15 def get_users(user_ud: int, db: Session =
    ↪ Depends(database.get_db)):
16     users = db.query(models.User).all()
17     return users
18
19 # Удалить пользователя
20 @router.delete("/{user_id}")
21 def delete_user(user_id: int, db: Session =
    ↪ Depends(database.get_db)):
22     user = db.query(models.User).filter(models.User.id ==
        ↪ user_id).first()
23     if not user:
24         raise HTTPException(status_code=404, detail="Пользователь
            ↪ не найден")
25     db.delete(user)
26     db.commit()
27     return {"detail": "Пользователь удален"}
28
29 # Обновить информацию о пользователе
30 @router.put("/{user_id}", response_model=schemas.UserOut)
31 def update_user(user_id: int, user_update: schemas.UserCreate, db:
    ↪ Session = Depends(database.get_db)):
32     user = db.query(models.User).filter(models.User.id ==
        ↪ user_id).first()
33     if not user:
34         raise HTTPException(status_code=404, detail="Пользователь
            ↪ не найден")
35
36     user.login = user_update.login
37     user.mail = user_update.mail
```

```
38      user.password = auth.get_password_hash(user_update.password)
39      user.name = user_update.name
40      user.lastname = user_update.lastname
41      user.surname = user_update.surname,
42      user.birthdate = user_update.birthdate,
43      user.city = user_update.city
44
45      db.commit()
46      db.refresh(user)
47      return user
```

backend/docker-compose.yml

```
1       services:
2       web:
3         build: .  # Эта строка говорит Compose использовать
          ↪ Dockerfile в текущей директории
4         container_name: fastapi-app
5         restart: always
6         volumes:
7           - .:/app  # Монтируем локальную папку в контейнер
8         ports:
9           - "8080:8000"
10        depends_on:
11          - db
12        environment:
13          -
            ↪ DATABASE_URL=postgresql://meebin_user:password@db/meebin_db
14
15      db:
16        image: postgres:16.3
17        container_name: postgres-db
18        environment:
19          POSTGRES_USER: meebin_user
20          POSTGRES_PASSWORD: password
```

```
21          POSTGRES_DB: meebin_db
22        ports:
23          - "5432:5432"
24        volumes:
25          - postgres-data:/var/lib/postgresql/data
26          - ./init.sql:/docker-entrypoint-initdb.d/init.sql
27
28      alembic:
29        build: .
30        container_name: alembic
31        volumes:
32          - .:/app
33        depends_on:
34          - db
35        environment:
36          -
            ↪ DATABASE_URL=postgresql://meebin_user:password@db/meebin_db
37        command: alembic upgrade head
38
39    volumes:
40      postgres-data:
```

backend/Dockerfile

```
1     # Используем официальный Python образ
2     FROM python:3.11-slim
3
4     # Устанавливаем зависимости
5     WORKDIR /app
6     COPY requirements.txt .
7     RUN pip install --no-cache-dir -r requirements.txt
8
9     # Копируем все файлы в контейнер
10    COPY . .
11
```

```
12      COPY init.sql /docker-entrypoint-initdb.d/
13
14      # Открываем порт 8000
15      EXPOSE 8000
16
17      # Команда запуска FastAPI приложения
18      CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0",
        ↪  "--port", "8000"]
```

backend/alembic/env.py

```
1  # alembic/env.py
2
3  import sys
4  import os
5  from logging.config import fileConfig
6
7  from sqlalchemy import engine_from_config
8  from sqlalchemy import pool
9  from alembic import context
10
11 # Добавляем путь к проекту
12 sys.path.append(os.path.abspath(os.path.join(
13     os.path.dirname(__file__), '..', 'app')))
14
15 from app.database import Base
16 from app import models  # Убедитесь, что ваши модели
   ↪  импортируются
17
18 # this is the Alembic Config object, which provides
19 # access to the values within the .ini file in use.
20 config = context.config
21
22 # Interpret the config file for Python logging.
23 # This line sets up loggers basically.
24 fileConfig(config.config_file_name)
```

```
25
26 # Добавьте ваши модели здесь
27 target_metadata = Base.metadata
28
29 def run_migrations_offline():
30     """Run migrations in 'offline' mode."""
31     url = config.get_main_option("sqlalchemy.url")
32     context.configure(
33         url=url, target_metadata=target_metadata,
            ↪ literal_binds=True, dialect_opts={"paramstyle":
            ↪ "named"}
34     )
35
36     with context.begin_transaction():
37         context.run_migrations()
38
39 def run_migrations_online():
40     """Run migrations in 'online' mode."""
41     connectable = engine_from_config(
42         config.get_section(config.config_ini_section),
43         prefix="sqlalchemy.",
44         poolclass=pool.NullPool,
45     )
46
47     with connectable.connect() as connection:
48         context.configure(connection=connection,
            ↪ target_metadata=target_metadata)
49
50         with context.begin_transaction():
51             context.run_migrations()
52
53 if context.is_offline_mode():
54     run_migrations_offline()
55 else:
56     run_migrations_online()
```