async def handle\_file\_created(db: Session, user\_id: uuid.UUID, file\_id: uuid.UUID, event\_data: dict, file\_hash: str = None, resume\_count: int = 0, session\_id: str = None):

    """Обрабатывает создание файла - создает новую сессию"""

    # ПРОВЕРКА ДУБЛИРОВАНИЯ СЕССИЙ

    active\_sessions = crud.get\_active\_sessions\_by\_user\_and\_file(db, user\_id, file\_id)

    if active\_sessions:

        print(f"⚠️ Active session already exists for file, updating instead of creating new")

        session = active\_sessions[0]

        session.last\_activity = datetime.fromisoformat(event\_data.get('event\_timestamp'))

        if file\_hash:

            session.hash\_before = file\_hash

        session.resume\_count = resume\_count

        # СОХРАНЯЕМ ФЛАГ is\_commented ПРИ ОБНОВЛЕНИИ

        is\_commented = session.is\_commented

        db.commit()

        db.refresh(session)

        # ВОССТАНАВЛИВАЕМ ФЛАГ is\_commented

        session.is\_commented = is\_commented

        db.commit()

        # Создаем событие created

        event\_record = schemas.FileEventCreate(

            session\_id=session.id,

            event\_type='created',

            file\_hash=file\_hash,

            event\_timestamp=datetime.fromisoformat(event\_data.get('event\_timestamp'))

        )

        crud.create\_file\_event(db, event\_record)

        print(f"🔄 Updated existing session {session.id} for created file")

        return {"action": "session\_updated", "session\_id": str(session.id), "resumed": False}

    # Проверяем, есть ли недавно закрытая сессия для возобновления

    recent\_session = None

    if resume\_count > 0:

        recent\_session = crud.get\_recent\_closed\_session(db, user\_id, file\_id)

    if recent\_session:

        # Возобновляем существующую сессию

        recent\_session.ended\_at = None

        recent\_session.last\_activity = datetime.fromisoformat(event\_data.get('event\_timestamp'))

        recent\_session.resume\_count = resume\_count

        recent\_session.hash\_before = file\_hash

        # СОХРАНЯЕМ ФЛАГ is\_commented ПРИ ВОЗОБНОВЛЕНИИ

        is\_commented = recent\_session.is\_commented

        db.commit()

        db.refresh(recent\_session)

        # ВОССТАНАВЛИВАЕМ ФЛАГ is\_commented

        recent\_session.is\_commented = is\_commented

        db.commit()

        print(f"🔄 Resumed session {recent\_session.id} (resume count: {resume\_count})")

        session = recent\_session

    else:

        # Создаем новую сессию, но используем session\_id из события если он валидный

        session\_data\_dict = {

            'user\_id': user\_id,

            'file\_id': file\_id,

            'started\_at': datetime.fromisoformat(event\_data.get('event\_timestamp')),

            'last\_activity': datetime.fromisoformat(event\_data.get('event\_timestamp')),

            'hash\_before': file\_hash,

            'resume\_count': resume\_count,

            'is\_commented': False  # НОВАЯ СЕССИЯ - КОММЕНТАРИЕВ ЕЩЕ НЕТ

        }

        # Если session\_id валидный, используем его

        if session\_id:

            try:

                session\_uuid = uuid.UUID(session\_id)

                # Проверяем, не существует ли уже сессия с таким ID

                existing\_session = crud.get\_file\_session(db, session\_uuid)

                if not existing\_session:

                    session\_data\_dict['id'] = session\_uuid

                    print(f"🎯 Using provided session ID for created event: {session\_id}")

            except ValueError:

                print(f"⚠️ Invalid session ID for created event, generating new one")

        session\_data = schemas.FileSessionCreate(\*\*session\_data\_dict)

        session = crud.create\_file\_session\_with\_id(db, session\_data)

        print(f"✅ Created session {session.id} for file {event\_data.get('file\_path')}")

    # Создаем событие файла

    event\_record = schemas.FileEventCreate(

        session\_id=session.id,

        event\_type='created',

        file\_hash=file\_hash,

        event\_timestamp=datetime.fromisoformat(event\_data.get('event\_timestamp'))

    )

    crud.create\_file\_event(db, event\_record)

    return {"action": "session\_created", "session\_id": str(session.id), "resumed": recent\_session is not None}

async def handle\_file\_modified(db: Session, user\_id: uuid.UUID, file\_id: uuid.UUID, event\_data: dict, file\_hash: str = None, session\_id: str = None, resume\_count: int = 0):

    """Обрабатывает изменение файла - обновляет существующую сессию или создает новую"""

    session = None

    if session\_id:

        # Пытаемся найти существующую сессию по ID из события

        try:

            session\_uuid = uuid.UUID(session\_id)

            session = crud.get\_file\_session(db, session\_uuid)

            if session:

                print(f"🔍 Found session by ID: {session.id}")

        except ValueError:

            print(f"⚠️ Invalid session ID format: {session\_id}")

            pass  # Невалидный UUID

    if not session:

        # Ищем активную сессию для этого пользователя и файла

        active\_sessions = crud.get\_active\_sessions\_by\_user\_and\_file(db, user\_id, file\_id)

        if active\_sessions:

            session = active\_sessions[0]

            print(f"🔍 Found active session by user and file: {session.id}")

    if session:

        # СОХРАНЯЕМ ФЛАГ is\_commented ПЕРЕД ОБНОВЛЕНИЕМ

        is\_commented = session.is\_commented

        # Обновляем существующую сессию

        session.last\_activity = datetime.fromisoformat(event\_data.get('event\_timestamp'))

        if file\_hash:

            session.hash\_after = file\_hash

        if resume\_count > session.resume\_count:

            session.resume\_count = resume\_count

        # ВОССТАНАВЛИВАЕМ ФЛАГ is\_commented

        session.is\_commented = is\_commented

        db.commit()

        db.refresh(session)

        print(f"📝 Updated session {session.id} for file {event\_data.get('file\_path')} (resume: {resume\_count}, is\_commented: {session.is\_commented})")

        action = "session\_updated"

    else:

        # Создаем новую сессию, но используем session\_id из события если он валидный

        session\_data\_dict = {

            'user\_id': user\_id,

            'file\_id': file\_id,

            'started\_at': datetime.fromisoformat(event\_data.get('event\_timestamp')),

            'last\_activity': datetime.fromisoformat(event\_data.get('event\_timestamp')),

            'hash\_before': file\_hash,

            'resume\_count': resume\_count,

            'is\_commented': False  # НОВАЯ СЕССИЯ - КОММЕНТАРИЕВ ЕЩЕ НЕТ

        }

        # Если session\_id валидный, используем его

        if session\_id:

            try:

                session\_uuid = uuid.UUID(session\_id)

                # Проверяем, не существует ли уже сессия с таким ID

                existing\_session = crud.get\_file\_session(db, session\_uuid)

                if not existing\_session:

                    session\_data\_dict['id'] = session\_uuid

                    print(f"🎯 Using provided session ID for modified event: {session\_id}")

            except ValueError:

                print(f"⚠️ Invalid session ID for modified event, generating new one")

        session\_data = schemas.FileSessionCreate(\*\*session\_data\_dict)

        session = crud.create\_file\_session\_with\_id(db, session\_data)

        print(f"✅ Created new session {session.id} for modified file {event\_data.get('file\_path')}")

        action = "session\_created"

    # Создаем событие файла

    event\_record = schemas.FileEventCreate(

        session\_id=session.id,

        event\_type='modified',

        file\_hash=file\_hash,

        event\_timestamp=datetime.fromisoformat(event\_data.get('event\_timestamp'))

    )

    crud.create\_file\_event(db, event\_record)

    return {"action": action, "session\_id": str(session.id)}