

**Автотесты за пределами POM:
строим империю на слоях и DDD**

*«Любой дурак может написать код, который понимает компьютер.
Хороший программист пишет код, который понимает человек.»*

Мартин Фаулер, «Рефакторинг»

Шалом



SDET, QA Auto



Артём Францев

Все идет по плану

1. Почему код автотестов превращается в хаос

Все идет по плану

1. Почему код автотестов превращается в хаос
2. Что мы можем взять у разработки

Все идет по плану

1. Почему код автотестов превращается в хаос
2. Что мы можем взять у разработки
3. Луковая архитектура для тестового фреймворка

Все идет по плану

1. Почему код автотестов превращается в хаос
2. Что мы можем взять у разработки
3. Луковая архитектура для тестового фреймворка
4. Домен, репозитории, сервисы — на примерах

Все идет по плану

1. Почему код автотестов превращается в хаос
2. Что мы можем взять у разработки
3. Луковая архитектура для тестового фреймворка
4. Домен, репозитории, сервисы — на примерах
5. Управление состоянием: `ArtifactStorage`

Все идет по плану

1. Почему код автотестов превращается в хаос
2. Что мы можем взять у разработки
3. Луковая архитектура для тестового фреймворка
4. Домен, репозитории, сервисы — на примерах
5. Управление состоянием: `ArtifactStorage`
6. Ловкие моки

Как обычно выглядит код автотестов

I. Проблема и мотивация

```
def test_order_rejected_when_limit_exceeded(db_cursor: Cursor, grpc_stub: LimitServiceStub) -> None:
    # Шаг 1: лезем в базу за инструментом
    db_cursor.execute(
        "SELECT symbol_id, symbol_name, board, lot_size, min_price_step "
        "FROM instruments.symbols "
        "WHERE symbol_name = 'Common1' AND board = 'TQBR'"
    )
    row = cursor.fetchone()
    symbol_id = row[0]
    symbol_name = row[1]
    board = row[2]
    lot_size = row[3]
    min_price_step = row[4]

    # Шаг 2: формируем gRPC-структуру и устанавливаем лимит

    limit_request = limit_pb2.SetLimitRequest(
        symbol_id=symbol_id,
        symbol_name=symbol_name,
        board=board,
        limit_type=limit_pb2.DAILY,
        amount=100_000_000,
        currency="RUB",
        effective_from="2024-01-01T00:00:00Z",
        effective_to="2024-12-31T23:59:59Z",
        created_by="autotest",
    )
    limit_response = grpc_stub.SetLimit(limit_request, timeout=5)
    assert limit_response.status == limit_pb2.SUCCESS
```

Как обычно выглядит код автотестов

I. Проблема и мотивация

```
def test_order_rejected_when_limit_exceeded(db_cursor: Cursor, grpc_stub: LimitServiceStub) -> None:
    # Шаг 1: лезем в базу за инструментом
    db_cursor.execute(
        "SELECT symbol_id, symbol_name, board, lot_size, min_price_step "
        "FROM instruments.symbols "
        "WHERE symbol_name = 'Common1' AND board = 'TQBR'"
    )
    row = cursor.fetchone()
    symbol_id = row[0]
    symbol_name = row[1]
    board = row[2]
    lot_size = row[3]
    min_price_step = row[4]

    # Шаг 2: формируем gRPC-структуру и устанавливаем лимит

    limit_request = limit_pb2.SetLimitRequest(
        symbol_id=symbol_id,
        symbol_name=symbol_name,
        board=board,
        limit_type=limit_pb2.DAILY,
        amount=100_000_000,
        currency="RUB",
        effective_from="2024-01-01T00:00:00Z",
        effective_to="2024-12-31T23:59:59Z",
        created_by="autotest",
    )
    limit_response = grpc_stub.SetLimit(limit_request, timeout=5)
    assert limit_response.status == limit_pb2.SUCCESS
```

Как обычно выглядит код автотестов

I. Проблема и мотивация

```
def test_order_rejected_when_limit_exceeded(db_cursor: Cursor, grpc_stub: LimitServiceStub) -> None:
    # Шаг 1: лезем в базу за инструментом
    db_cursor.execute(
        "SELECT symbol_id, symbol_name, board, lot_size, min_price_step "
        "FROM instruments.symbols "
        "WHERE symbol_name = 'Common1' AND board = 'TQBR'"
    )
    row = cursor.fetchone()
    symbol_id = row[0]
    symbol_name = row[1]
    board = row[2]
    lot_size = row[3]
    min_price_step = row[4]

    # Шаг 2: формируем gRPC-структуру и устанавливаем лимит

    limit_request = limit_pb2.SetLimitRequest(
        symbol_id=symbol_id,
        symbol_name=symbol_name,
        board=board,
        limit_type=limit_pb2.DAILY,
        amount=100_000_000,
        currency="RUB",
        effective_from="2024-01-01T00:00:00Z",
        effective_to="2024-12-31T23:59:59Z",
        created_by="autotest",
    )
    limit_response = grpc_stub.SetLimit(limit_request, timeout=5)
    assert limit_response.status == limit_pb2.SUCCESS
```

Как обычно выглядит код автотестов

I. Проблема и мотивация

```
def test_order_rejected_when_limit_exceeded(db_cursor: Cursor, grpc_stub: LimitServiceStub) -> None:
    # Шаг 1: лезем в базу за инструментом
    db_cursor.execute(
        "SELECT symbol_id, symbol_name, board, lot_size, min_price_step "
        "FROM instruments.symbols "
        "WHERE symbol_name = 'Common1' AND board = 'TQBR'"
    )
    row = cursor.fetchone()
    symbol_id = row[0]
    symbol_name = row[1]
    board = row[2]
    lot_size = row[3]
    min_price_step = row[4]

    # Шаг 2: формируем gRPC-структуру и устанавливаем лимит

    limit_request = limit_pb2.SetLimitRequest(
        symbol_id=symbol_id,
        symbol_name=symbol_name,
        board=board,
        limit_type=limit_pb2.DAILY,
        amount=100_000_000,
        currency="RUB",
        effective_from="2024-01-01T00:00:00Z",
        effective_to="2024-12-31T23:59:59Z",
        created_by="autotest",
    )
    limit_response = grpc_stub.SetLimit(limit_request, timeout=5)
    assert limit_response.status == limit_pb2.SUCCESS
```

Как обычно выглядит код автотестов

I. Проблема и мотивация

```
def test_order_rejected_when_limit_exceeded(db_cursor: Cursor, grpc_stub: LimitServiceStub) -> None:
    # Шаг 1: лезем в базу за инструментом
    db_cursor.execute(
        "SELECT symbol_id, symbol_name, board, lot_size, min_price_step "
        "FROM instruments.symbols "
        "WHERE symbol_name = 'Common1' AND board = 'TQBR'"
    )
    row = cursor.fetchone()
    symbol_id = row[0]
    symbol_name = row[1]
    board = row[2]
    lot_size = row[3]
    min_price_step = row[4]

    # Шаг 2: формируем gRPC-структуру и устанавливаем лимит

    limit_request = limit_pb2.SetLimitRequest(
        symbol_id=symbol_id,
        symbol_name=symbol_name,
        board=board,
        limit_type=limit_pb2.DAILY,
        amount=100_000_000,
        currency="RUB",
        effective_from="2024-01-01T00:00:00Z",
        effective_to="2024-12-31T23:59:59Z",
        created_by="autotest",
    )
    limit_response = grpc_stub.SetLimit(limit_request, timeout=5)
    assert limit_response.status == limit_pb2.SUCCESS
```

Как обычно выглядит код автотестов

I. Проблема и мотивация

```
def test_order_rejected_when_limit_exceeded(db_cursor: Cursor, grpc_stub: LimitServiceStub) -> None:
    # Шаг 1: лезем в базу за инструментом
    db_cursor.execute(
        "SELECT symbol_id, symbol_name, board, lot_size, min_price_step "
        "FROM instruments.symbols "
        "WHERE symbol_name = 'Common1' AND board = 'TQBR'"
    )
    row = cursor.fetchone()
    symbol_id = row[0]
    symbol_name = row[1]
    board = row[2]
    lot_size = row[3]
    min_price_step = row[4]

    # Шаг 2: формируем gRPC-структуру и устанавливаем лимит

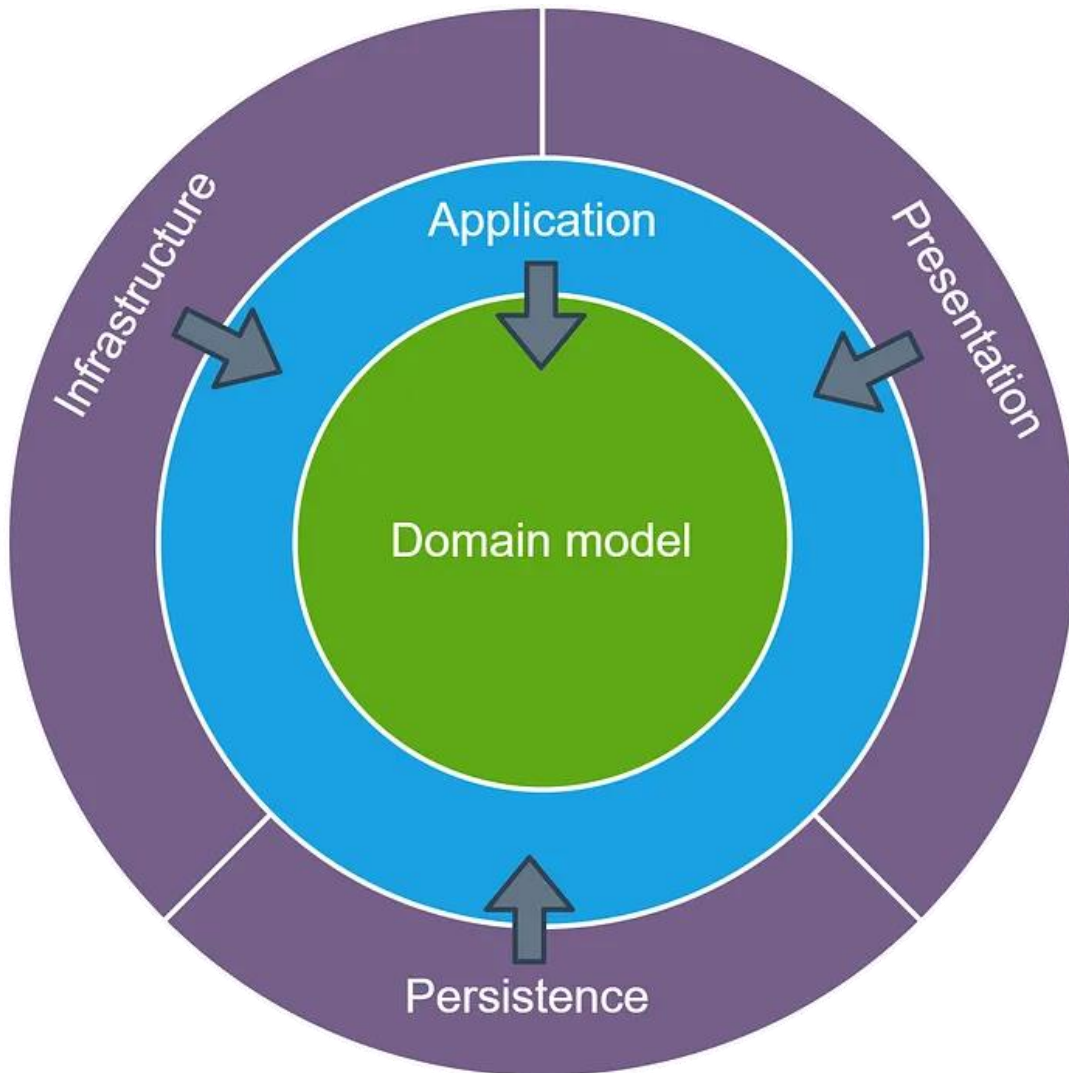
    limit_request = limit_pb2.SetLimitRequest(
        symbol_id=symbol_id,
        symbol_name=symbol_name,
        board=board,
        limit_type=limit_pb2.DAILY,
        amount=100_000_000,
        currency="RUB",
        effective_from="2024-01-01T00:00:00Z",
        effective_to="2024-12-31T23:59:59Z",
        created_by="autotest",
    )
    limit_response = grpc_stub.SetLimit(limit_request, timeout=5)
    assert limit_response.status == limit_pb2.SUCCESS
```

Инженерный подход

I. Проблема и мотивация

У разработки есть:

- Луковая архитектура;

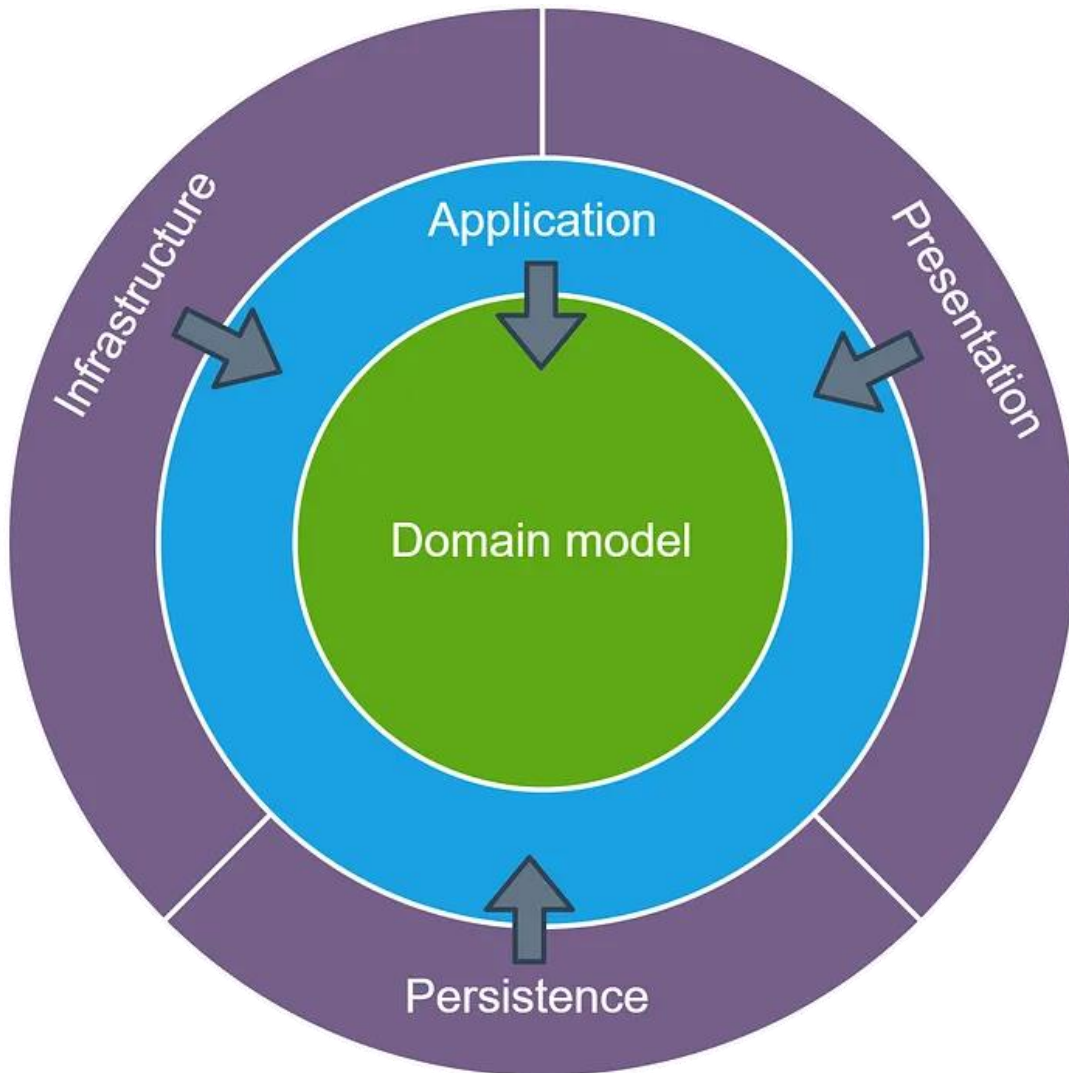


Инженерный подход

I. Проблема и мотивация

У разработки есть:

- Луковая архитектура;
- SOLID принципы;

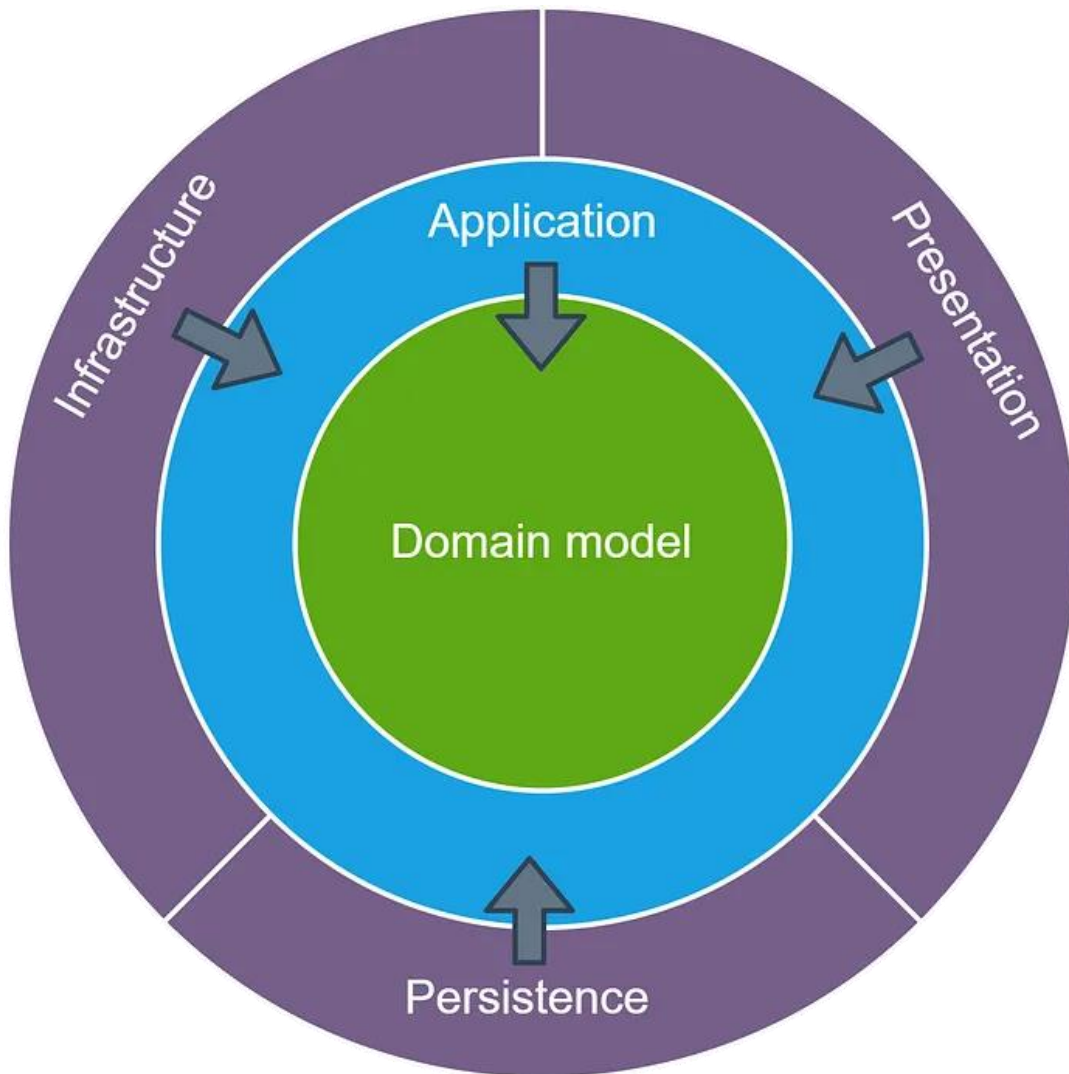


Инженерный подход

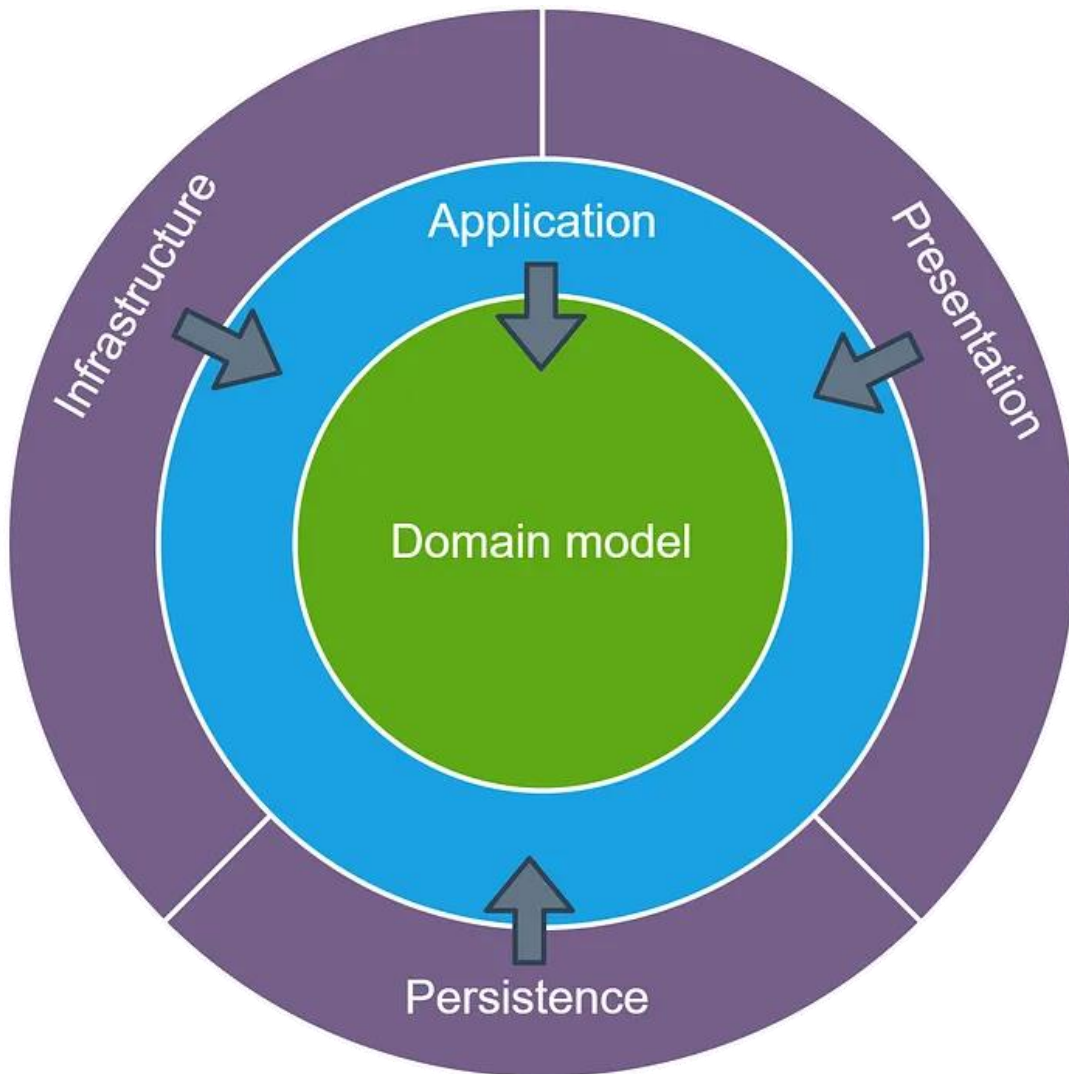
I. Проблема и мотивация

У разработки есть:

- Луковая архитектура;
- SOLID принципы;
- Domain Driven Design (DDD)



Инженерный подход



I. Проблема и мотивация

У разработки есть:

- Луковая архитектура;
- SOLID принципы;
- Domain Driven Design (DDD)

А у нас... POM?

Мы можем подходить к написанию кодовой базы фреймворков с той же инженерной точки зрения, что и разработчики!

Главная мысль

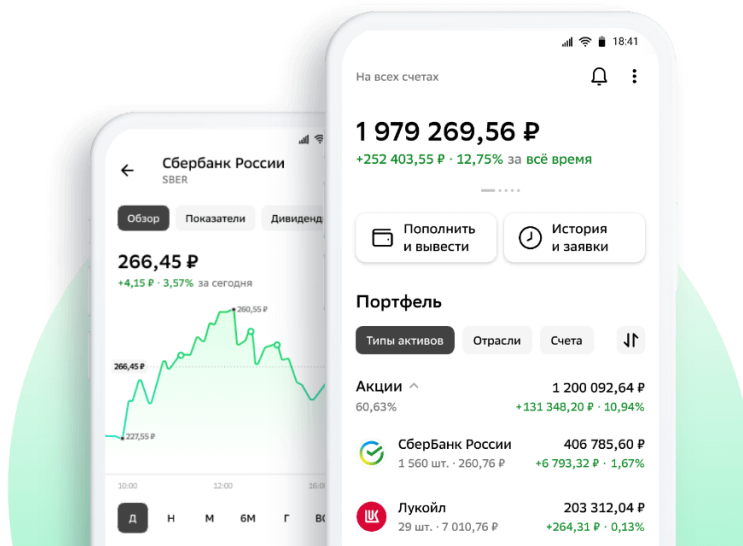
Тестовая функция должна читаться как ручной тест-кейс

Подытожим: Код автотестов — такой же продукт. Он заслуживает продвинутой архитектуры на больших и долгих проектах.

Давайте пофантазируем

II. Контекст продукта

Наш продукт — торговый клиент, система, через которую можно покупать и продавать финансовые инструменты

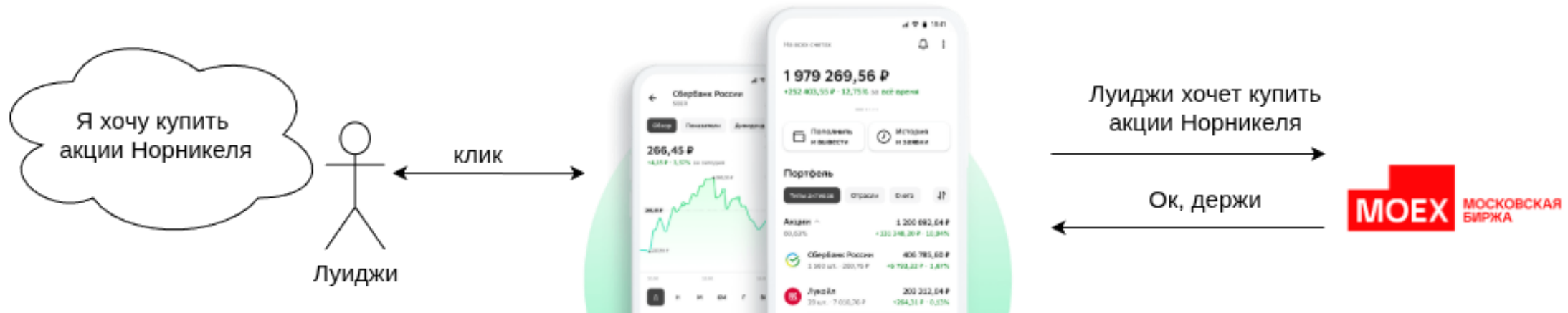


↔ Наш зоопарк систем ↔



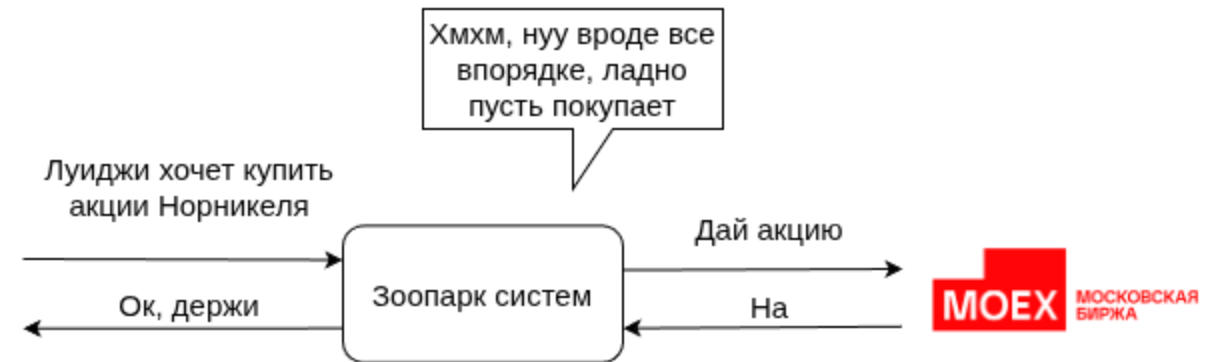
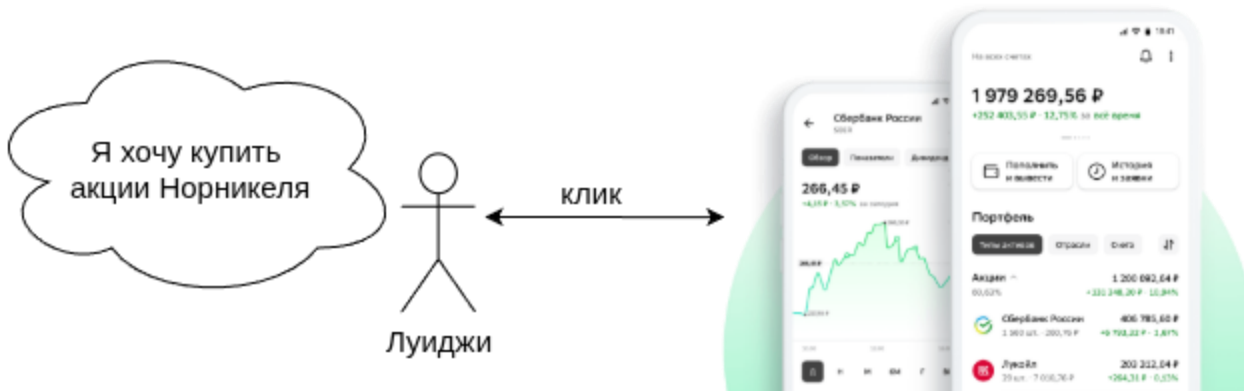
Давайте пофантазируем

II. Контекст продукта



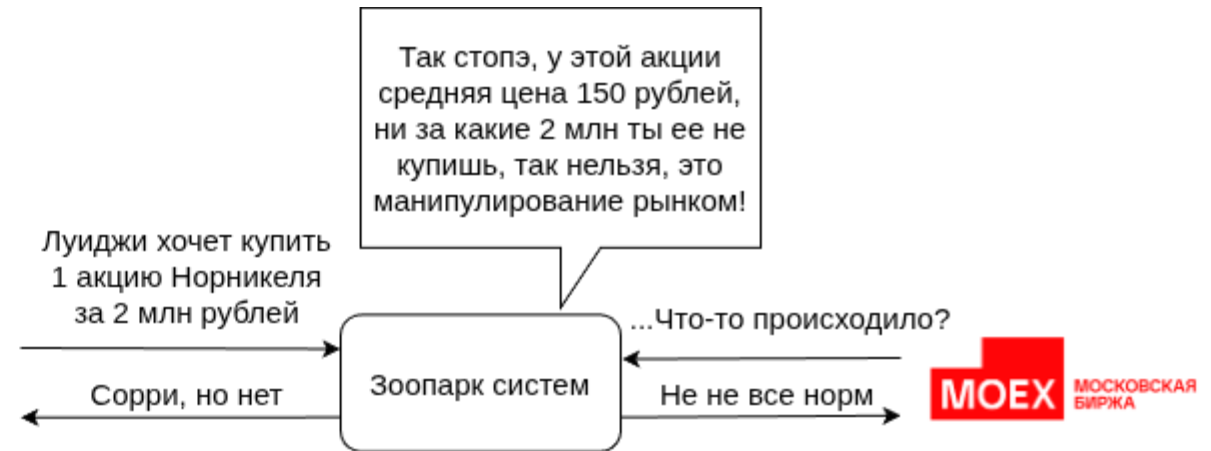
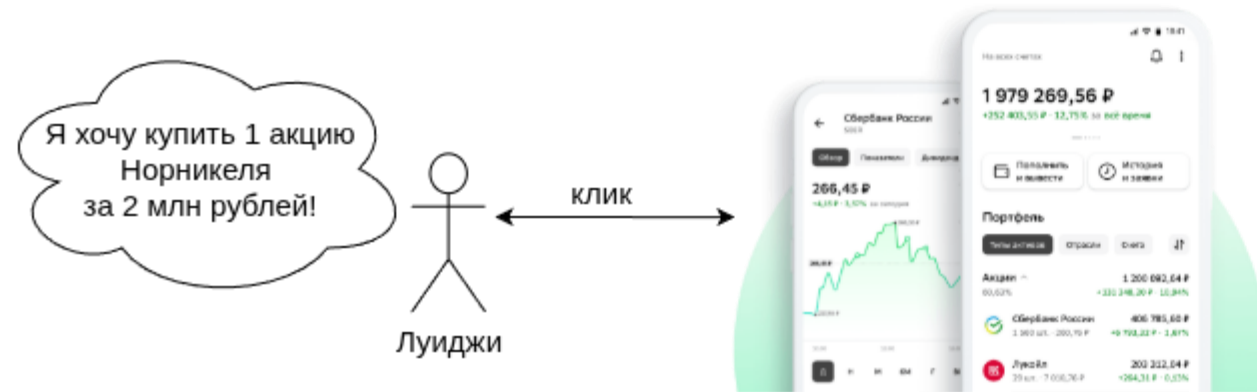
Давайте пофантазируем

II. Контекст продукта



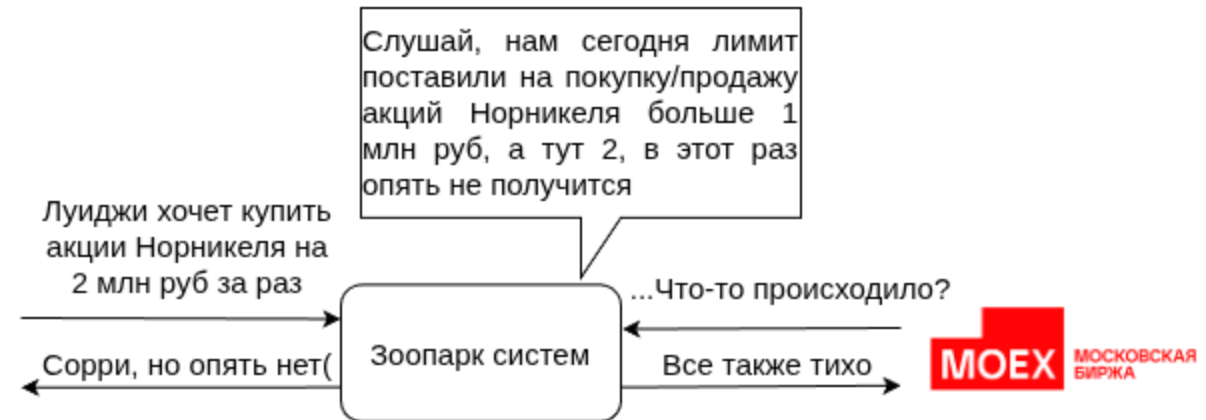
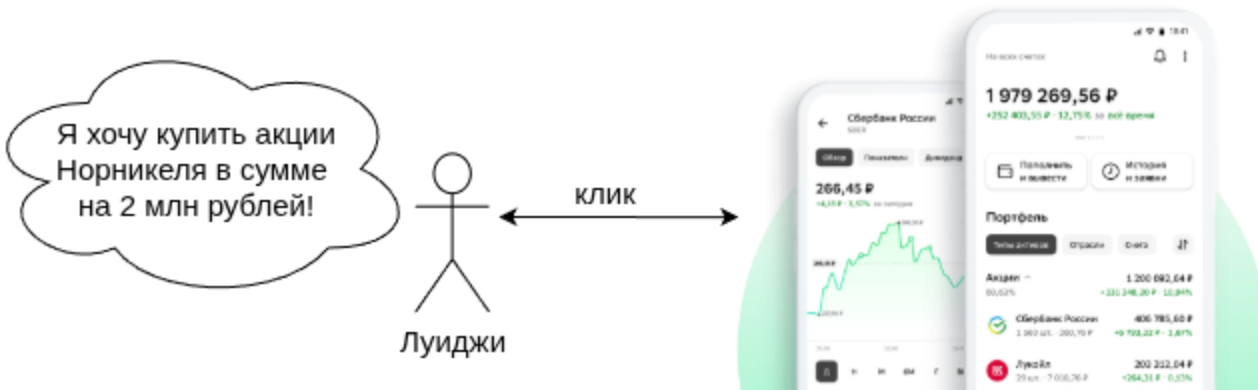
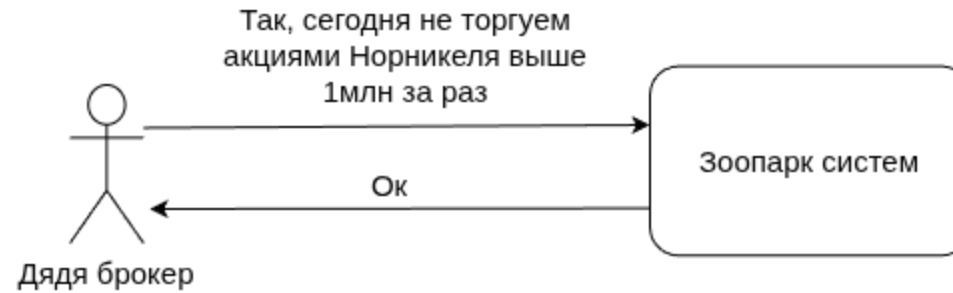
Давайте пофантазируем

II. Контекст продукта



Давайте пофантазируем

II. Контекст продукта



Ручной E2E сценарий

II. Контекст продукта

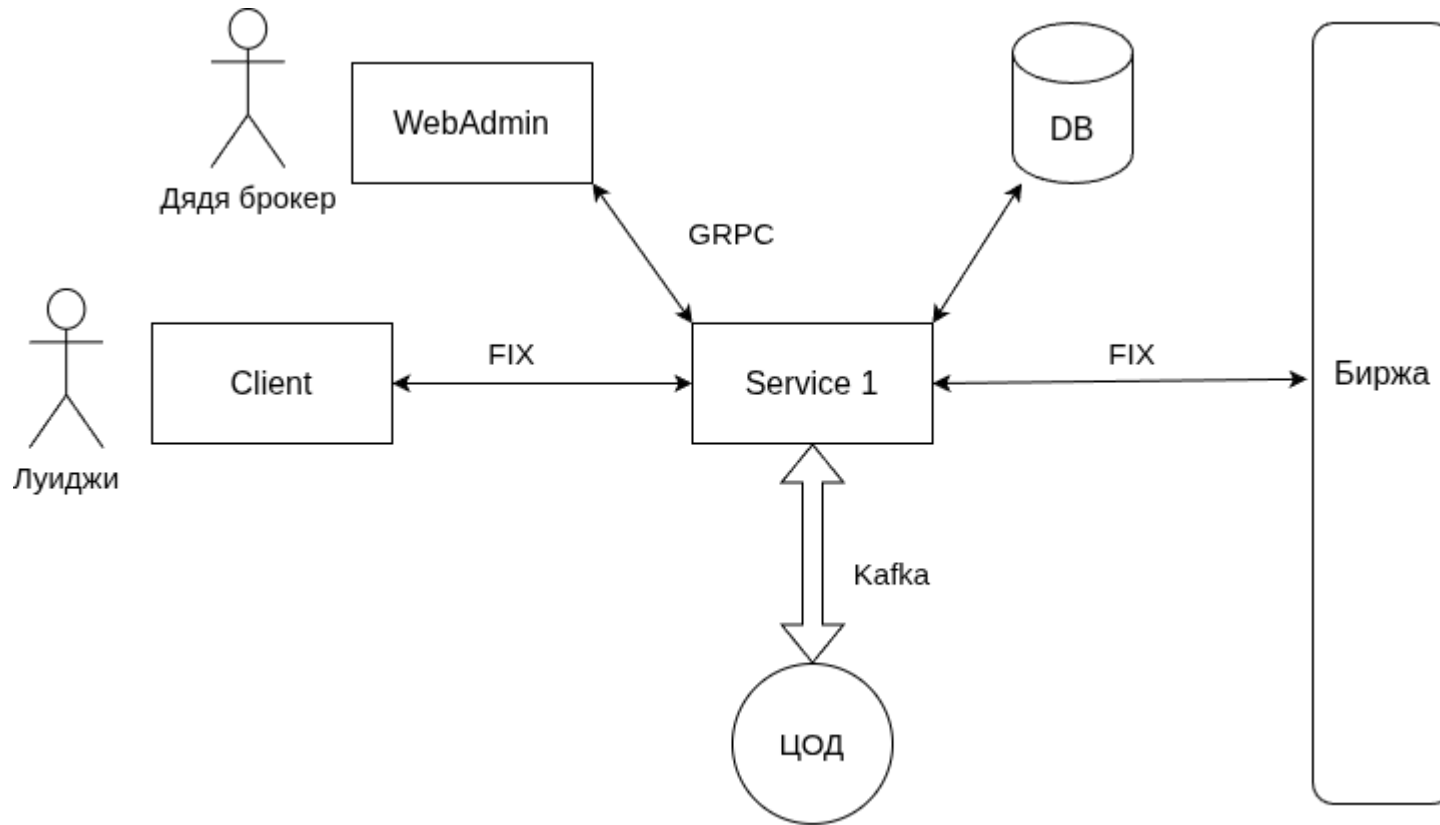
1. Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.
2. Отправить заявку на покупку акций больше чем 1 миллион рублей.
3. Ожидаемый результат: система принимает заявку, видит превышение лимита, возвращает отказ.

Автоматизируем м?



Схема системы

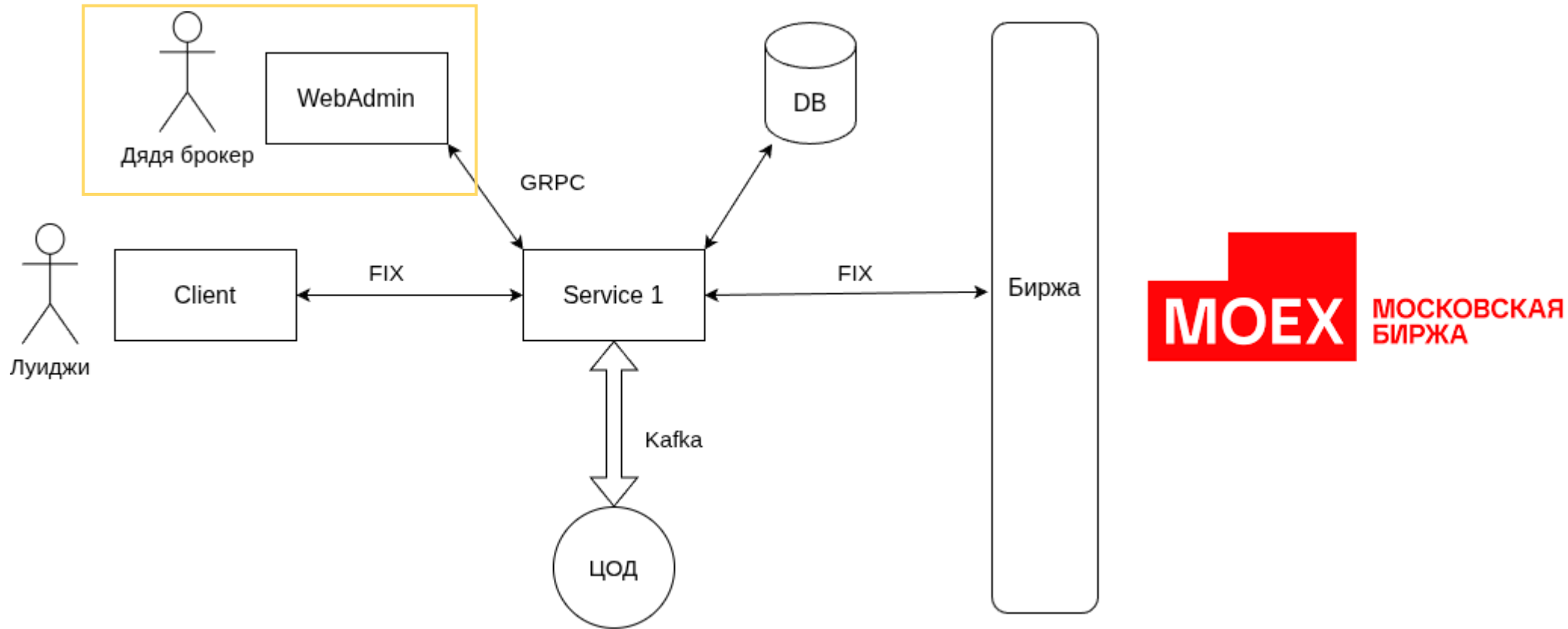
II. Контекст продукта



- FIX-протокол — отправка/получение ордеров;
- gRPC — управление настройками, установка лимитов на оборот фин инструментов;
- PostgreSQL — хранение данных об инструментах, лимитах;
- Kafka — аудит, нотификации.

Схема системы

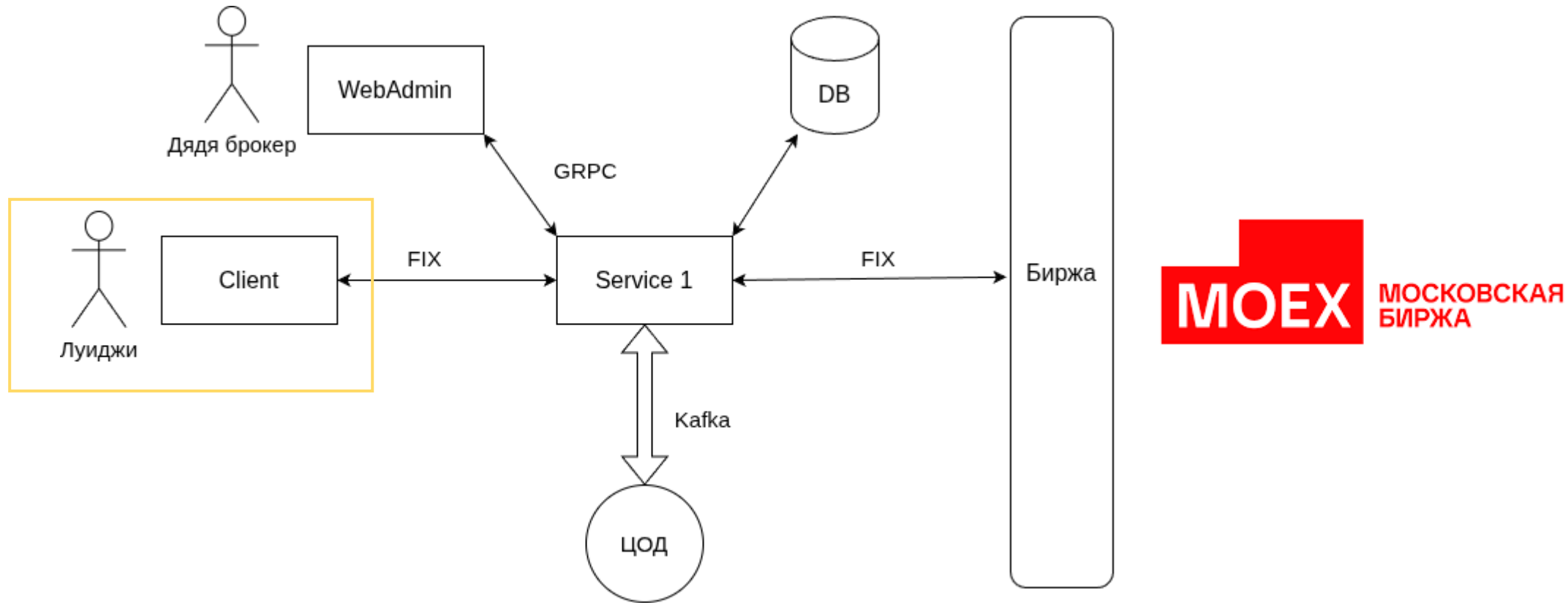
II. Контекст продукта



- FIX-протокол — отправка/получение ордеров;
- gRPC — управление настройками, установка лимитов на оборот фин инструментов;
- PostgreSQL — хранение данных об инструментах, лимитах;
- Kafka — аудит, нотификации.

Схема системы

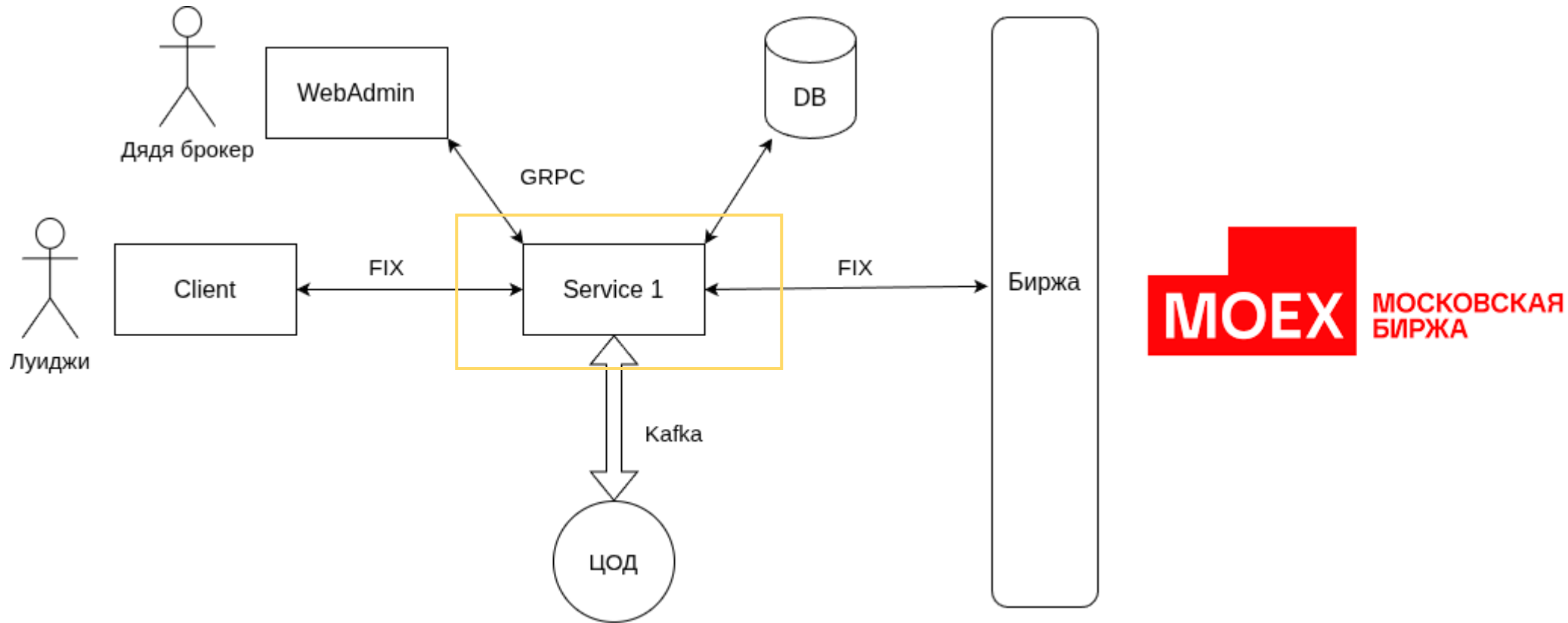
II. Контекст продукта



- FIX-протокол — отправка/получение ордеров;
- gRPC — управление настройками, установка лимитов на оборот фин инструментов;
- PostgreSQL — хранение данных об инструментах, лимитах;
- Kafka — аудит, нотификации.

Схема системы

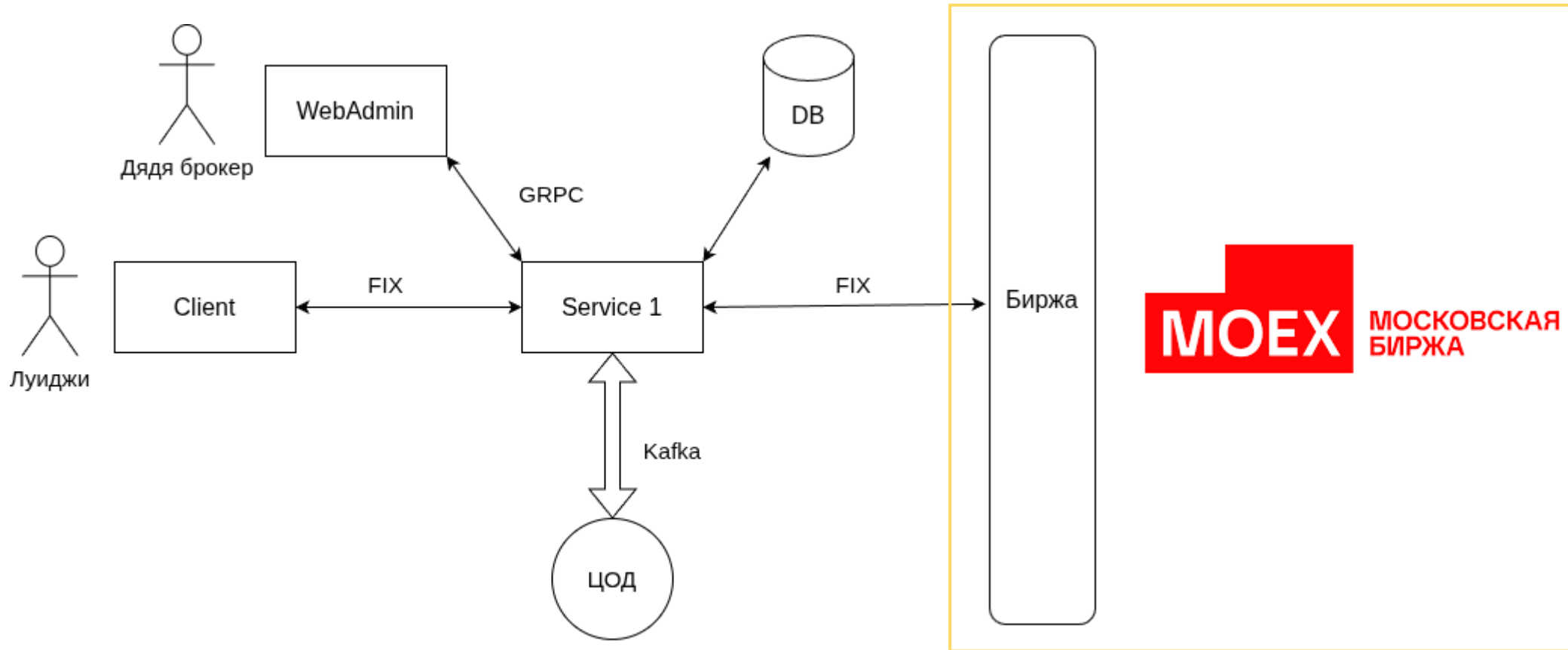
II. Контекст продукта



- FIX-протокол — отправка/получение ордеров;
- gRPC — управление настройками, установка лимитов на оборот фин инструментов;
- PostgreSQL — хранение данных об инструментах, лимитах;
- Kafka — аудит, нотификации.

Схема системы

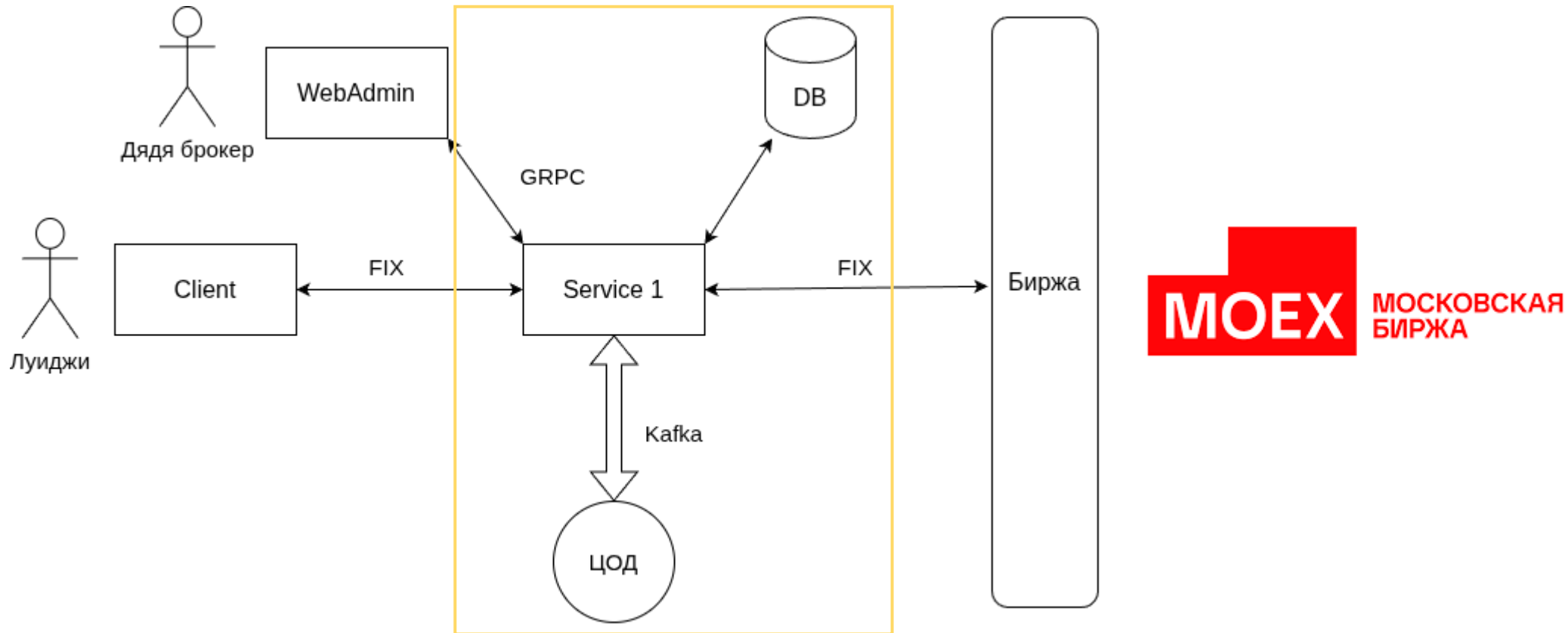
II. Контекст продукта



- FIX-протокол — отправка/получение ордеров;
- gRPC — управление настройками, установка лимитов на оборот фин инструментов;
- PostgreSQL — хранение данных об инструментах, лимитах;
- Kafka — аудит, нотификации.

Схема системы

II. Контекст продукта

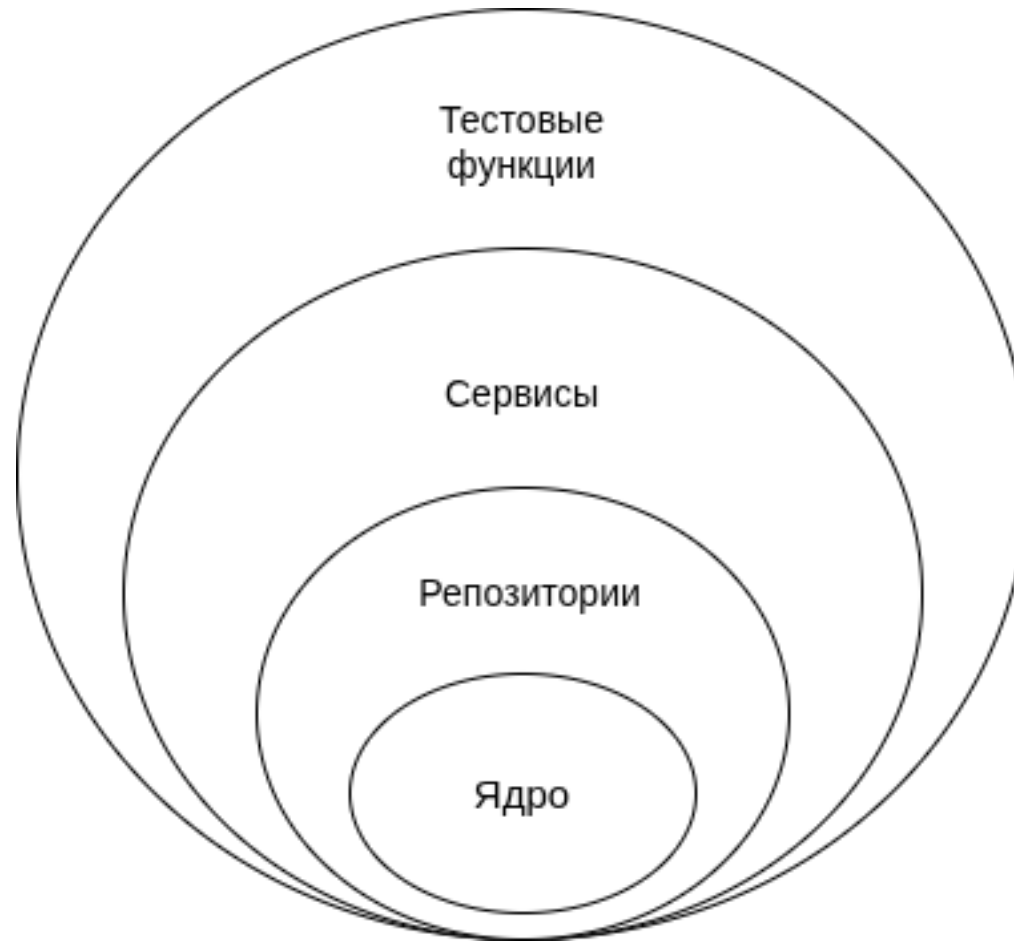


- FIX-протокол — отправка/получение ордеров;
- gRPC — управление настройками, установка лимитов на оборот фин инструментов;
- PostgreSQL — хранение данных об инструментах, лимитах;
- Kafka — аудит, нотификации.

Подытожим: Архитектура фреймворка — зеркало архитектуры продукта.

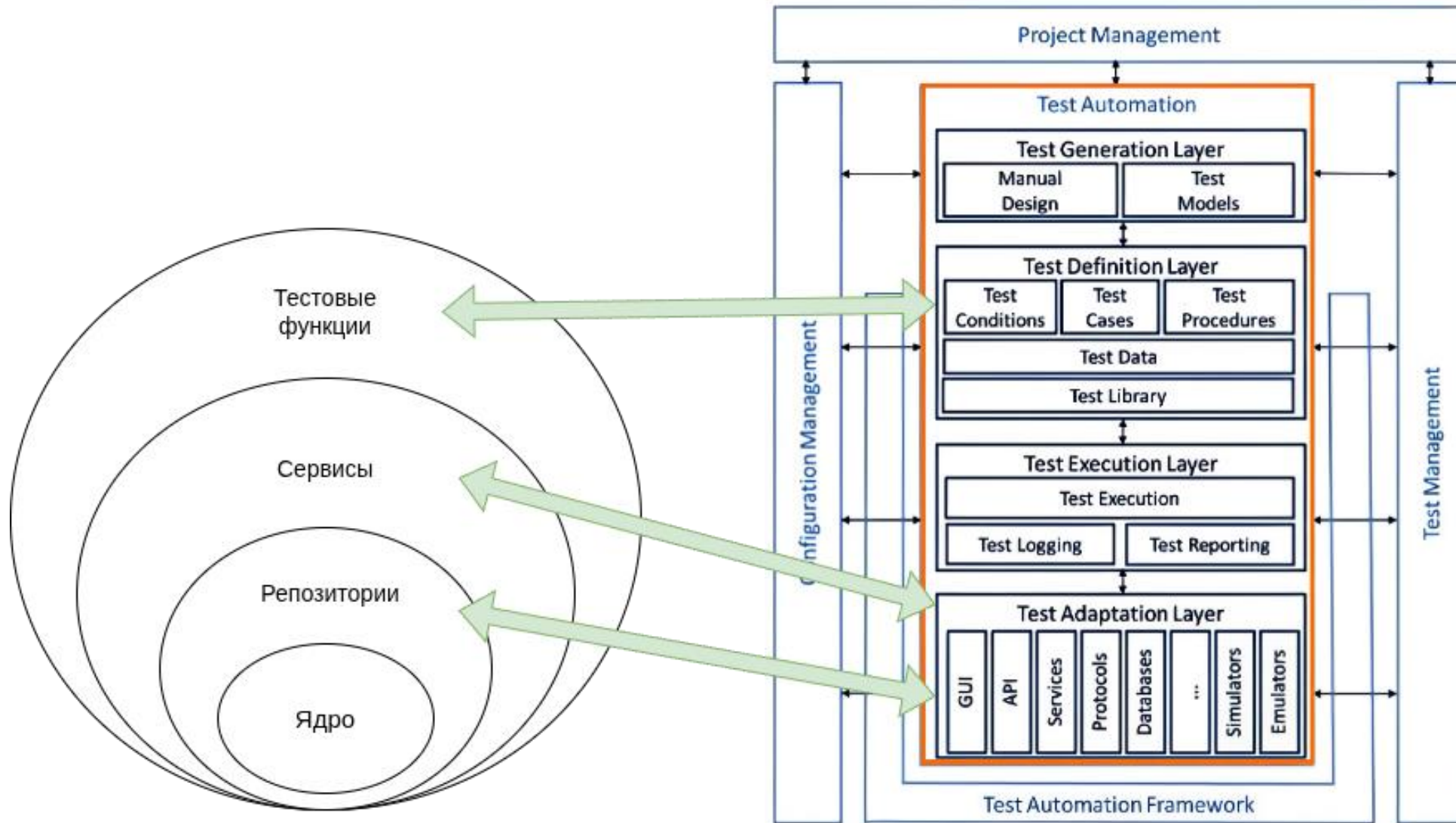
Луковая архитектура для тестового фреймворка

III. Архитектура: общая картина



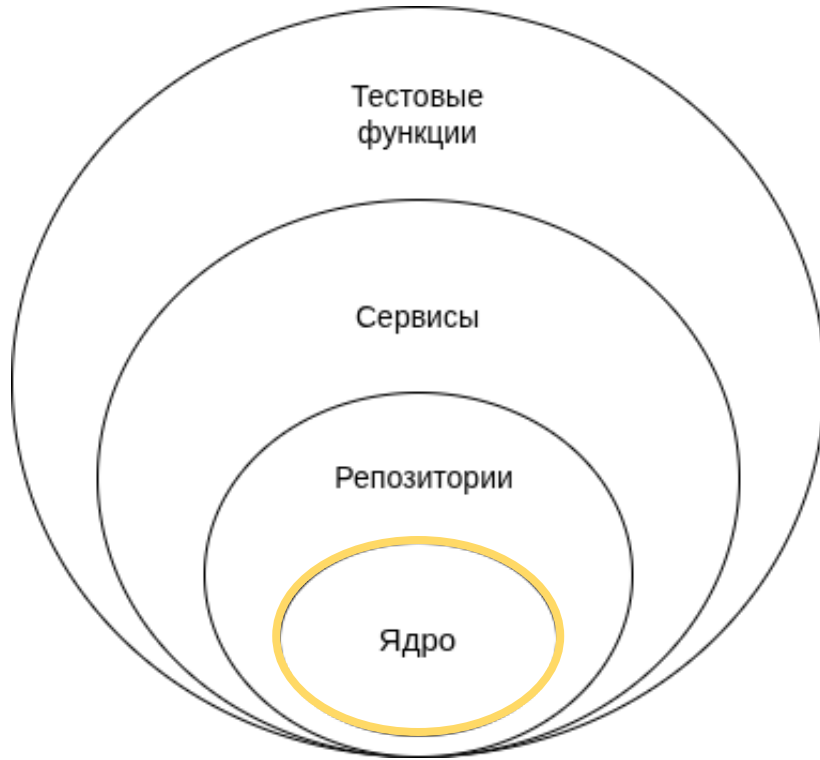
Структура фреймворка по ISTQB

III. Архитектура: общая картина



Луковая архитектура для тестового фреймворка

IV. Ядро - Домен

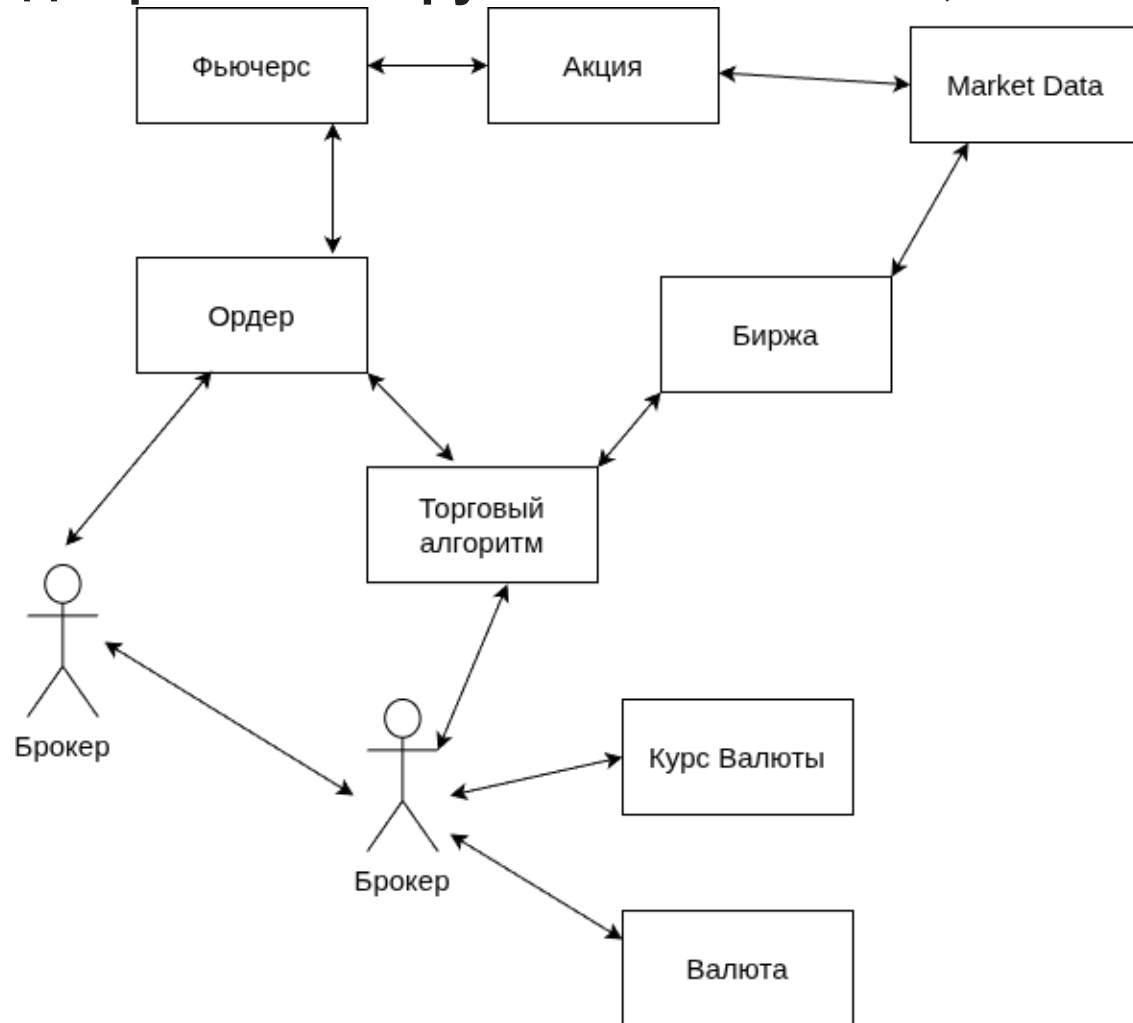


– Ядро — доменные модели, доменные ошибки

DDD за 60 секунд

IV. Ядро - Домен

DDD (Domain-Driven Design, предметно-ориентированное проектирование) — подход, где **код строится вокруг бизнес-понятий**, а не вокруг технологий.



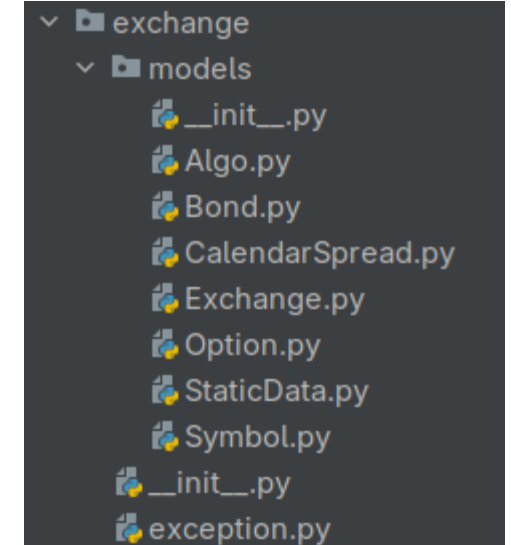
- Доменная модель - «Акция», «Ордер», «Лимит» а не Row, dict, protobuf-message;
- Единый язык - Все слои фреймворка говорят одними терминами;
- Слои - Домен в центре, всё остальное адаптируется к нему

Ядро — сердце фреймворка

IV. Ядро - Домен

```
class Symbol(BaseModel):
    system_id: t.Optional[int] = None
    system_name: t.Optional[str] = None
    isin: t.Optional[str] = None
    cfi_code: t.Optional[str] = None
    name: t.Optional[str] = None
    class_: t.Optional[str] = None
    lot_size: t.Optional[int] = None
    currency: t.Optional[str] = None
    settl_type: t.Optional[int] = None
    contract_size: t.Optional[int] = None
    adt: t.Optional[int] = None
    instr_type: t.Optional[str] = None
    tick_size: t.Optional[float] = None
    tick_value: t.Optional[float] = None
    tick_value_curr: t.Optional[float] = None
    # this field is required for the repository
    # not every instrument can be identified by
    test_type: t.Optional[str] = None
```

```
class Option(Symbol):
    base_symbol: t.Optional[Symbol] = None
    put_call: t.Optional[str] = None
    base_contract: t.Optional[str] = None
    maturity_date: t.Optional[date] = None
    option_type: t.Optional[str] = None
    strike: t.Optional[float] = None
```



Подход из DDD: у нас есть объекты доменной области, с которыми мы работаем по всей структуре фреймворка.

Объект `Symbol` — финансовый инструмент, у которого есть свойства, присущие доменной модели из продукта

Не путать с моделью из ORM!

```
class Symbol(BaseModel):
    system_id: t.Optional[int] = None
    system_name: t.Optional[str] = None
    isin: t.Optional[str] = None
    cfi_code: t.Optional[str] = None
    name: t.Optional[str] = None
    class_: t.Optional[str] = None
    lot_size: t.Optional[int] = None
    currency: t.Optional[str] = None
    settl_type: t.Optional[int] = None
    contract_size: t.Optional[int] = None
    adt: t.Optional[int] = None
    instr_type: t.Optional[str] = None
    tick_size: t.Optional[float] = None
    tick_value: t.Optional[float] = None
    tick_value_curr: t.Optional[float] = None
    # this field is required for the repository
    # not every instrument can be identified by
    test_type: t.Optional[str] = None
```

Доменная модель и модель из базы данных — это две разные вещи

```
class PSymbol(DeclarativeBase):
    __tablename__ = "symbol"

    system_id: Mapped[int] = mapped_column(Integer, primary_key=True)
    isin: Mapped[str] = mapped_column(VARCHAR, default=None)
    cfi_code: Mapped[str] = mapped_column(VARCHAR)
    name: Mapped[str] = mapped_column(VARCHAR)
    class_: Mapped[str] = mapped_column(VARCHAR)
    lot_size: Mapped[int] = mapped_column(Integer)
    currency: Mapped[str] = mapped_column(VARCHAR)
    settl_type: Mapped[int] = mapped_column(Integer)
    contract_size: Mapped[int] = mapped_column(Integer)
    adt: Mapped[int] = mapped_column(BIGINT, default=None)
    instr_type: Mapped[str] = mapped_column(Text, default=None)
```

Не путать с моделью из ORM!

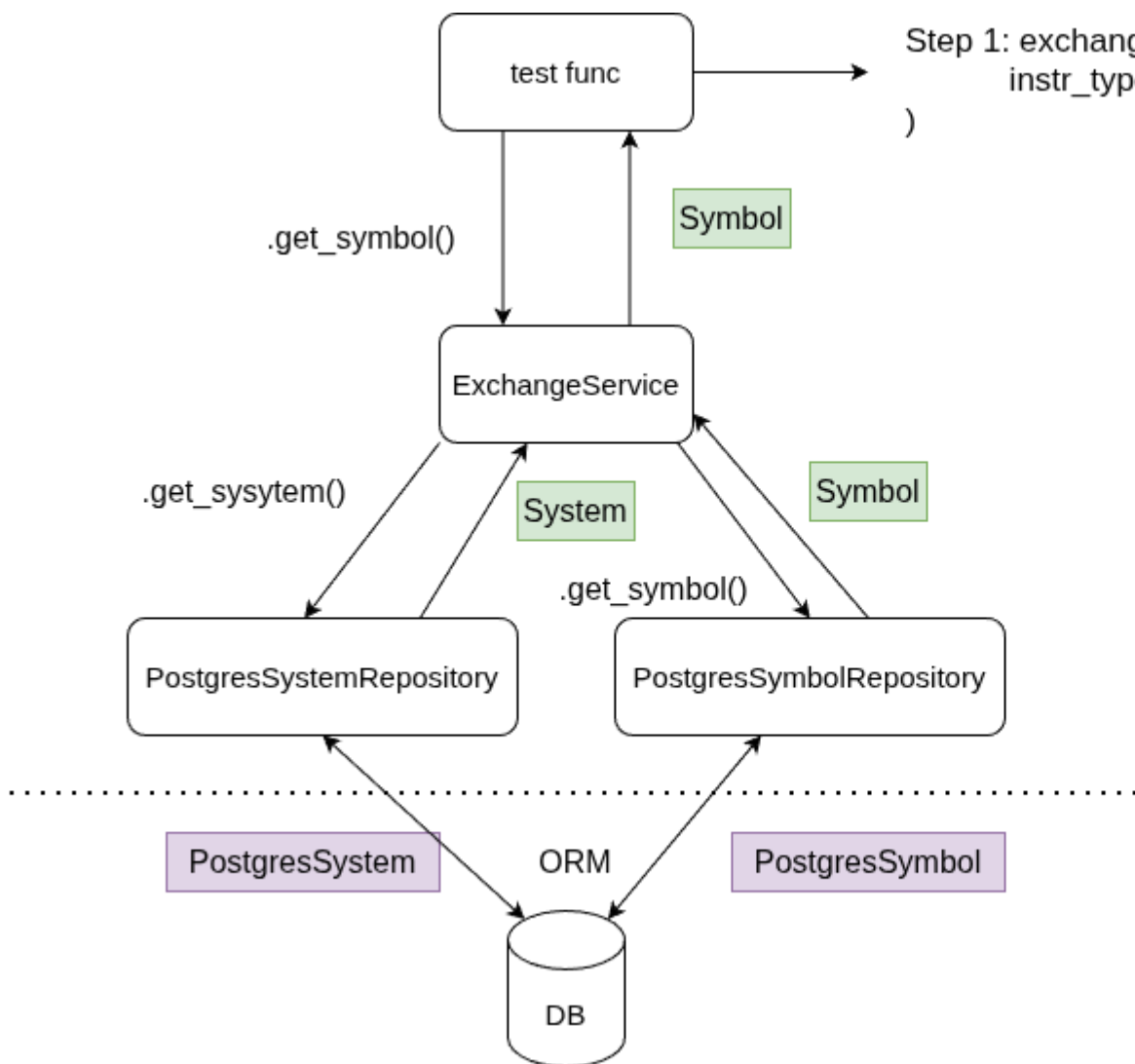
IV. Ядро - Домен

Доменная модель и модель из базы данных — это две разные вещи

```
class Symbol(BaseModel):
    system_id: t.Optional[int] = None
    system_name: t.Optional[str] = None
    isin: t.Optional[str] = None
    cfi_code: t.Optional[str] = None
    name: t.Optional[str] = None
    class_: t.Optional[str] = None
    lot_size: t.Optional[int] = None
    currency: t.Optional[str] = None
    settl_type: t.Optional[int] = None
    contract_size: t.Optional[int] = None
    adt: t.Optional[int] = None
    instr_type: t.Optional[str] = None
    tick_size: t.Optional[float] = None
    tick_value: t.Optional[float] = None
    tick_value_curr: t.Optional[float] = None
    # this field is required for the repository
    # not every instrument can be identified by
    test_type: t.Optional[str] = None
```

```
class PSymbol(DeclarativeBase):
    __tablename__ = "symbol"

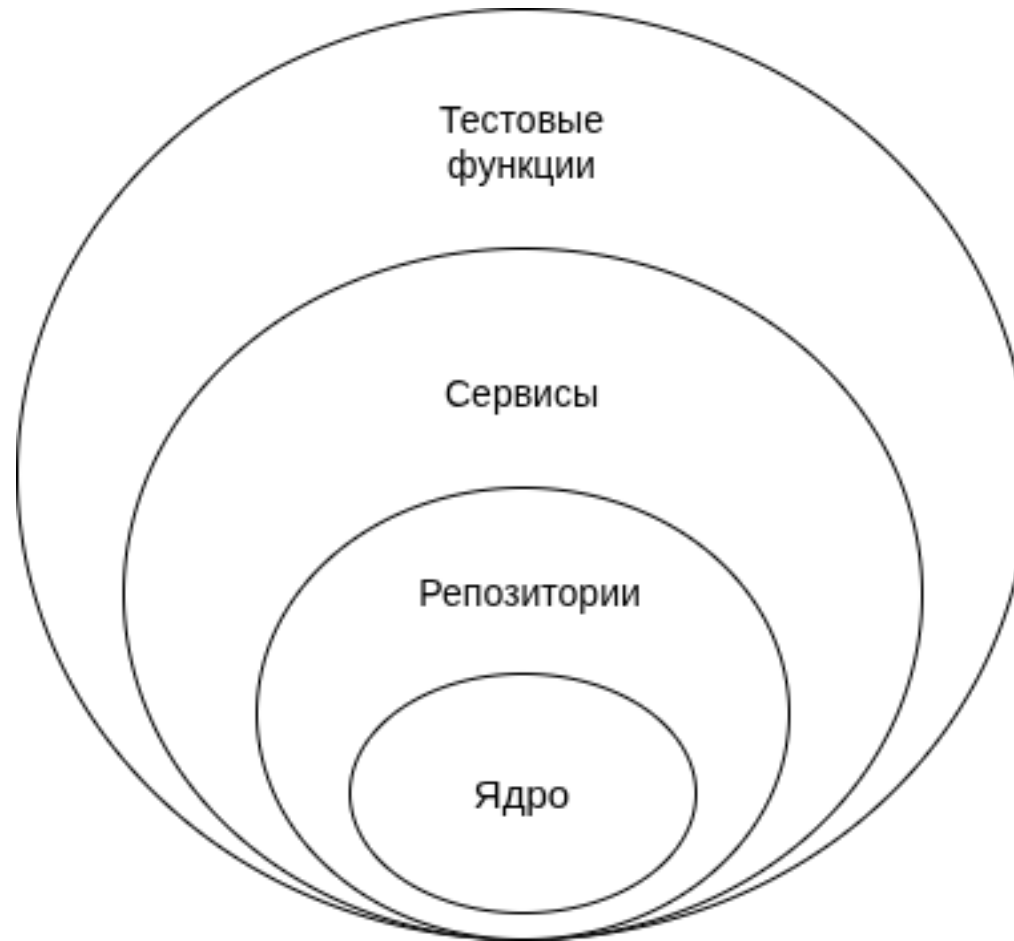
    system_id: Mapped[int] = mapped_column(Integer, primary_key=True)
    isin: Mapped[str] = mapped_column(VARCHAR, default=None)
    cfi_code: Mapped[str] = mapped_column(VARCHAR)
    name: Mapped[str] = mapped_column(VARCHAR)
    class_: Mapped[str] = mapped_column(VARCHAR)
    lot_size: Mapped[int] = mapped_column(Integer)
    currency: Mapped[str] = mapped_column(VARCHAR)
    settl_type: Mapped[int] = mapped_column(Integer)
    contract_size: Mapped[int] = mapped_column(Integer)
    adt: Mapped[int] = mapped_column(BIGINT, default=None)
    instr_type: Mapped[str] = mapped_column(Text, default=None)
```

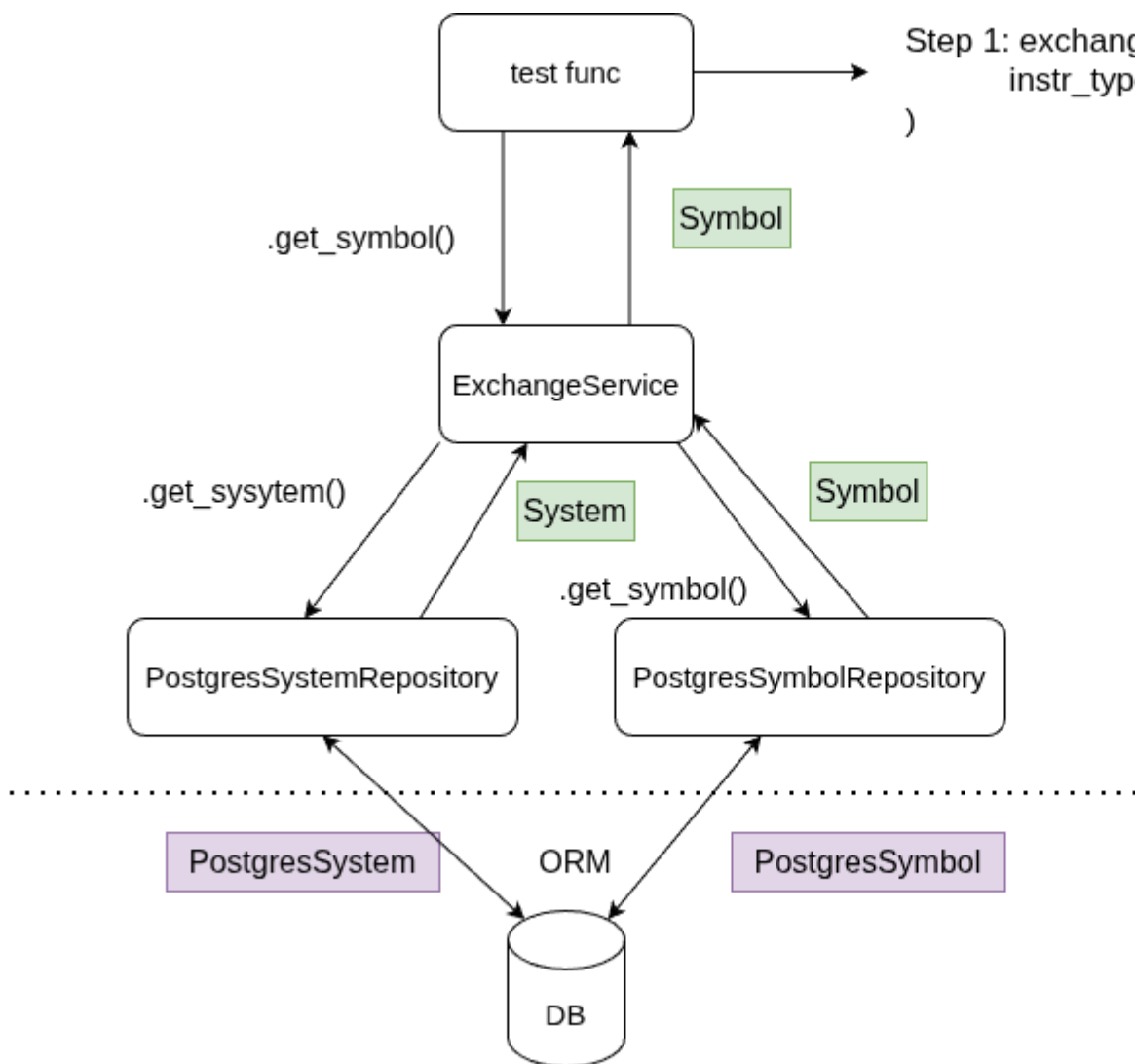


Весь фреймворк использует только доменную модель. Про ORM модель знает только PostgresSymbolRepository и на выход также возвращает доменную.

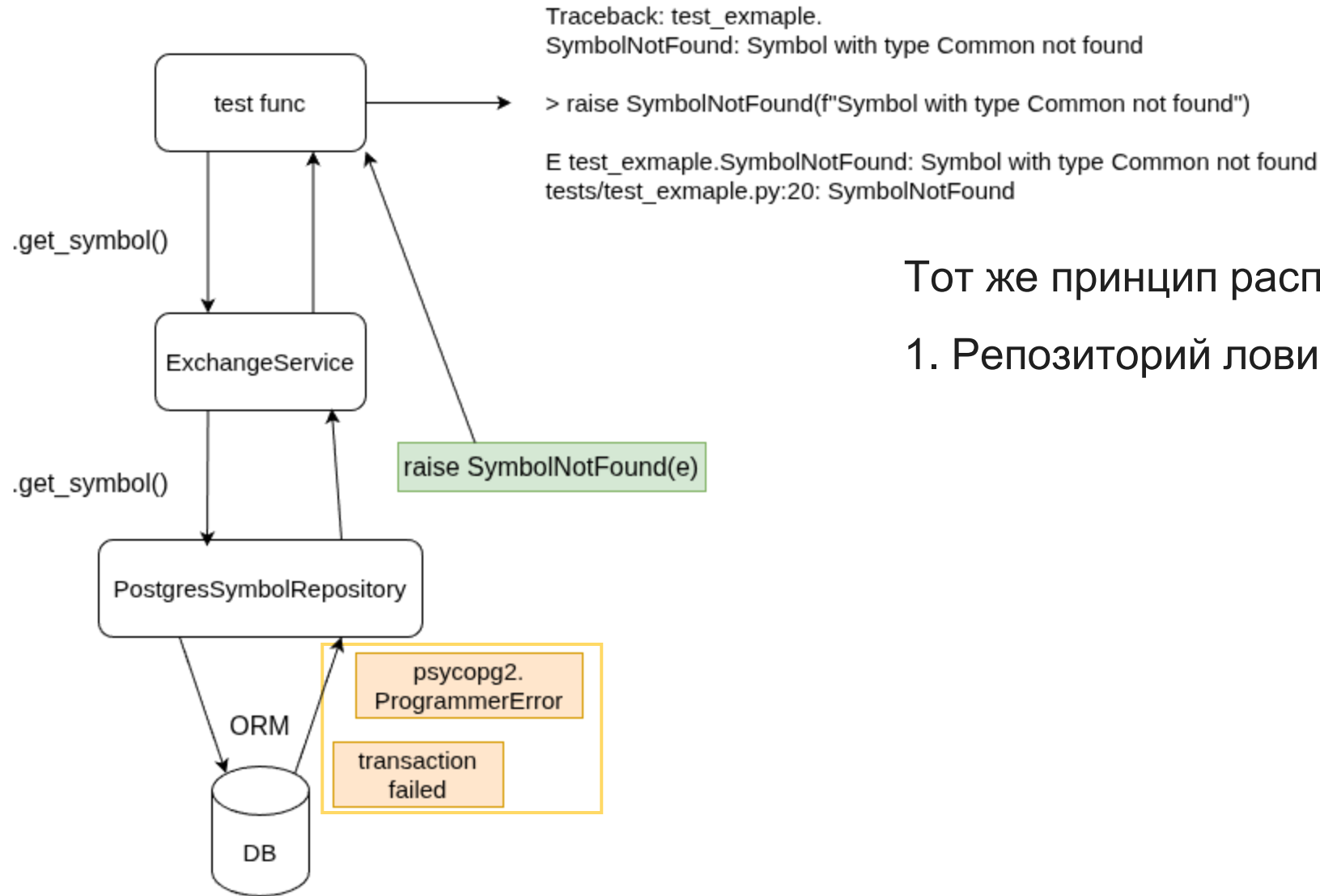
Луковая архитектура для тестового фреймворка

III. Архитектура: общая картина



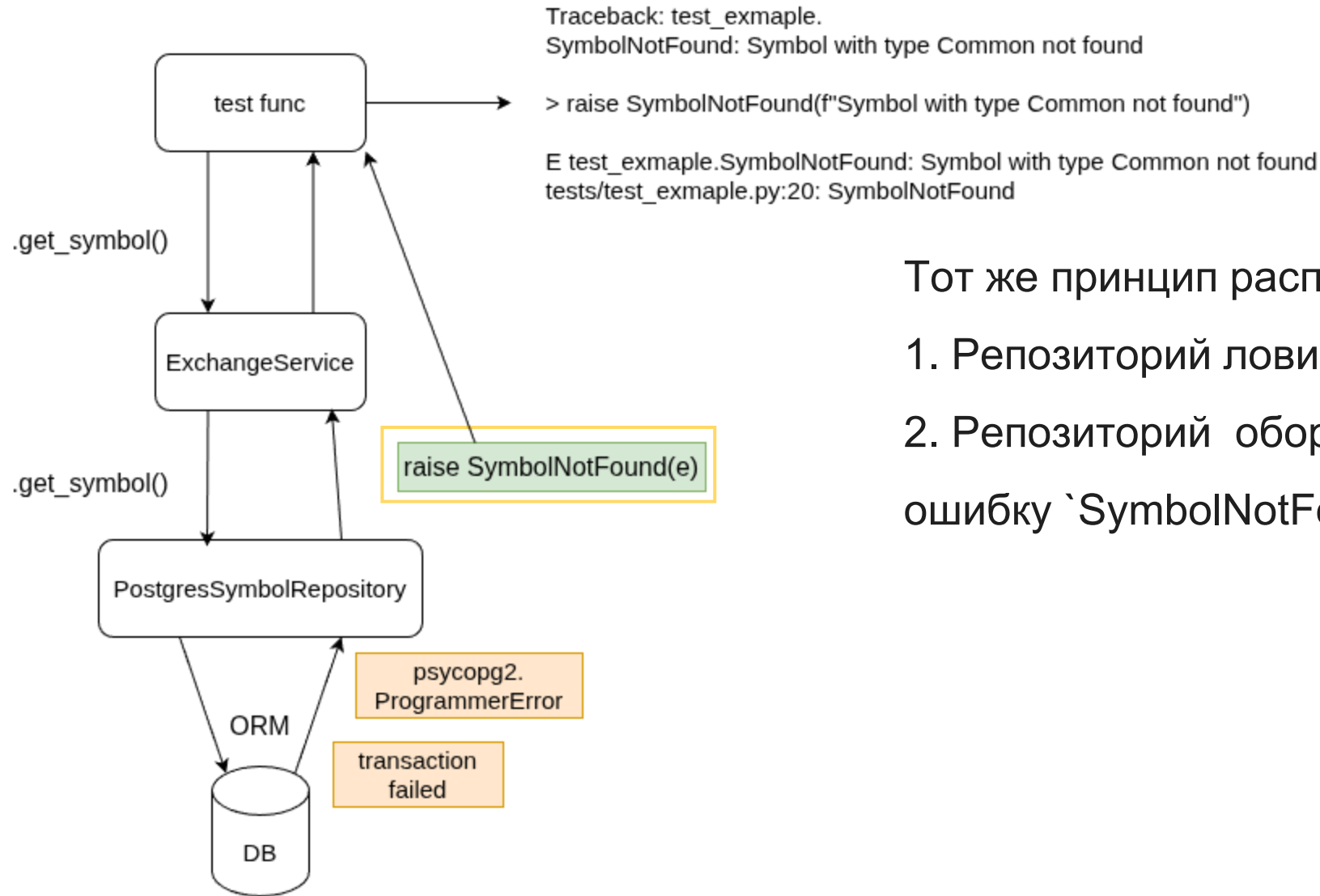


Весь фреймворк использует только доменную модель. Про ORM модель знает только PostgresSymbolRepository и на выход также возвращает доменную.



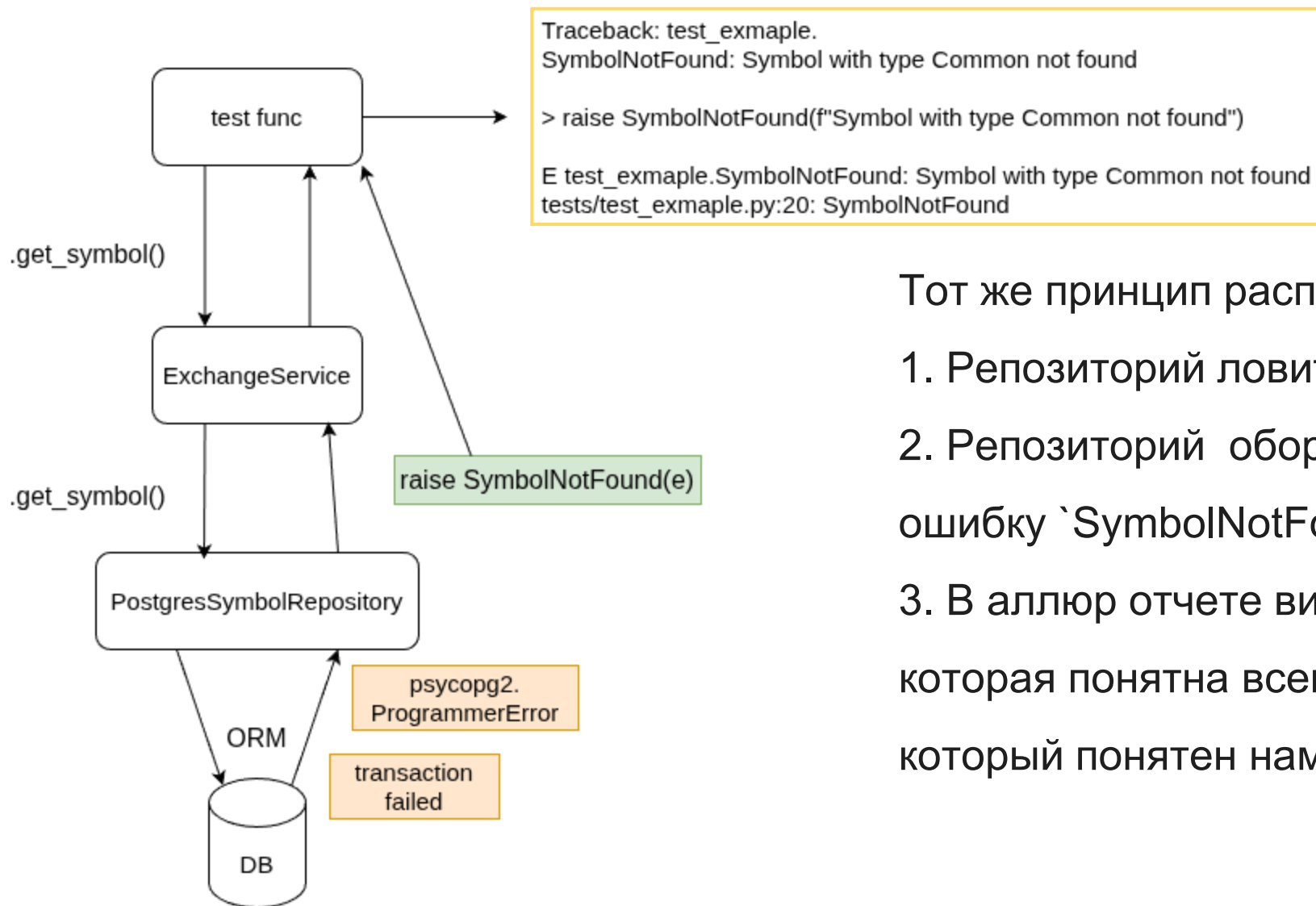
Тот же принцип распространяется на ошибки.

1. Репозиторий ловит свою техническую ошибку



Тот же принцип распространяется на ошибки.

1. Репозиторий ловит свою техническую ошибку
2. Репозиторий оборачивает её в доменную ошибку `SymbolNotFound`



Тот же принцип распространяется на ошибки.

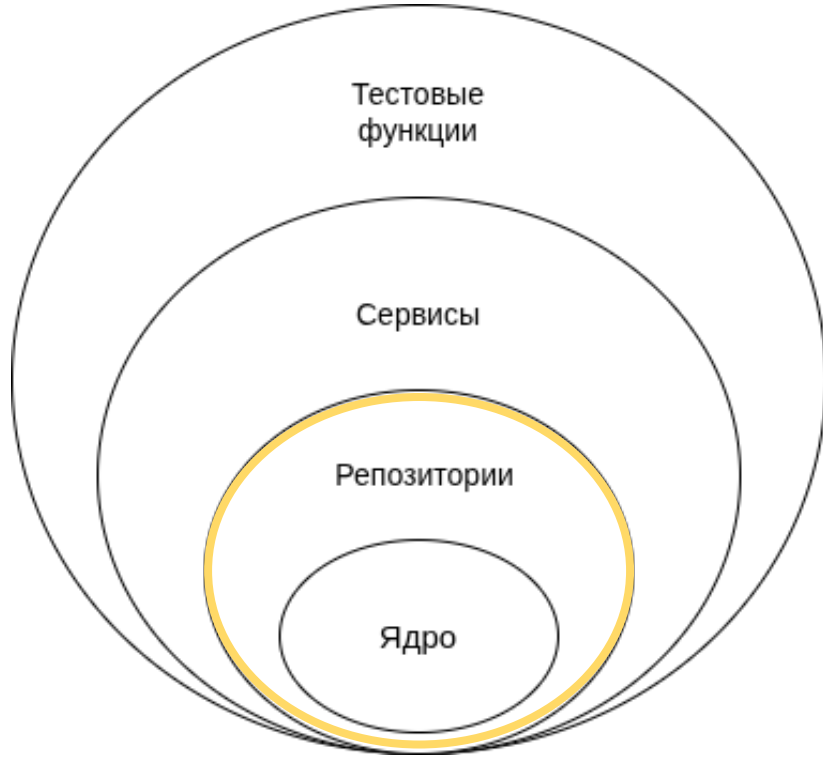
1. Репозиторий ловит свою техническую ошибку
2. Репозиторий оборачивает её в доменную ошибку `SymbolNotFound`
3. В аллюр отчете видим доменную ошибку, которая понятна всей команде и стектрейс, который понятен нам

Подытожим: Доменная модель — единый язык всего фреймворка.

Всё остальное конвертируется в неё.

Репозитории — уровень протоколов

V. Репозитории



- Ядро — доменные модели, доменные ошибки
- Репозитории — конкретные реализации протоколов

Репозитории — уровень протоколов

V. Репозитории

```
class ISymbolRepository:
    8 usages
    def get_symbol(self, count: int = 1, with_base_symbol_board_list: bool = False, **conditions_and) -> t.List[Symbol]:
        """
        Additional conditions for Options and legs for CalSpread:
        - with_base_symbol_board_list (if true repository find symbol only with board in BASE_SYMBOL_BOARDS_LIST)
        """
        raise NotImplementedError

    2 usages
    def get_option(self, count: int = 1, **conditions_and) -> t.List[Option]:
        raise NotImplementedError

    2 usages
    def get_bond(self, count: int = 1, with_accrued: bool = False, **conditions_and) -> t.List[Symbol]:
        """
        Additional conditions for Bonds:
        - with_accrued (if true repository find symbol where accrued is not None)
        """
        raise NotImplementedError

    2 usages
    def get_legs_calendar_spread(self, instrument: Symbol) -> t.Tuple[str, str]:
        raise NotImplementedError
```

Репозитории — уровень протоколов

V. Репозитории

```
# When using an ORM model on this table, the same objects are returned as a result.
# Therefore we use sql query
def get_symbol(self, count: int = 1, with_base_symbol_board_list: bool = False, **conditions_and) -> t.List[Symbol]:
    conditions_query = _get_conditions_and_psymbol_sql_string(**conditions_and)
    if with_base_symbol_board_list:
        base_symbol_board_sql_list = str(BASE_SYMBOL_BOARDS_LIST).replace('[', '(').replace(']', ')')
        conditions_query += f" AND class IN {base_symbol_board_sql_list}"

    sql_query = f"""SELECT * FROM {PSymbol.__tablename__} WHERE
                    {conditions_query} LIMIT {str(self.limit)};"""
    try:
        response = self.session.execute(text(sql_query)).all()
        symbol_list = [self._tuple_to_domain_symbol(i) for i in response]
        random.shuffle(symbol_list)
    except Exception as e:
        raise PostgresRepositoryError(e)
    if len(symbol_list) > count:
        symbol_list = symbol_list[:count]
    elif len(symbol_list) < count:
        raise SymbolNotFound(f"Symbol with conditions {conditions_and} not found")
    return symbol_list
```

```
# When using an ORM model on this table, the same objects are returned as a result.
# Therefore we use sql query
def get_symbol(self, count: int = 1, with_base_symbol_board_list: bool = False, **conditions_and) -> t.List[Symbol]:
    conditions_query = _get_conditions_and_psymbol_sql_string(**conditions_and)
    if with_base_symbol_board_list:
        base_symbol_board_sql_list = str(BASE_SYMBOL_BOARDS_LIST).replace('[', '(').replace(']', ')')
        conditions_query += f" AND class IN {base_symbol_board_sql_list}"

    sql_query = f"""SELECT * FROM {PSymbol.__tablename__} WHERE
        {conditions_query} LIMIT {str(self.limit)};"""
    try:
        response = self.session.execute(text(sql_query)).all()
        symbol_list = [self._tuple_to_domain_symbol(i) for i in response]
        random.shuffle(symbol_list)
    except Exception as e:
        raise PostgresRepositoryError(e)
    if len(symbol_list) > count:
        symbol_list = symbol_list[:count]
    elif len(symbol_list) < count:
        raise SymbolNotFound(f"Symbol with conditions {conditions_and} not found")
    return symbol_list
```

Репозитории — уровень протоколов

V. Репозитории

```
def _db_to_domain_symbol(self, psymbol: PSymbol) -> Symbol:
    symbol = Symbol(
        system_id=psymbol.system_id,
        isin=psymbol.isin,
        cfi_code=psymbol.cfi_code,
        name=psymbol.name,
        class_=psymbol.class_,
        lot_size=int(psymbol.lot_size),
        currency=str(psymbol.currency),
        settl_type=int(psymbol.settl_type),
        contract_size=int(psymbol.contract_size),
        adt=psymbol.adt,
        instr_type=psymbol.instr_type,
        test_type=psymbol.instr_type
    )

    return symbol

def _domain_to_db_symbol(self, symbol: Symbol) -> PSymbol:
    return PSymbol(
        system_id=symbol.system_id,
        isin=symbol.isin,
        cfi_code=symbol.cfi_code,
        name=symbol.name,
        class_=symbol.class_,
        lot_size=symbol.lot_size,
        currency=symbol.currency,
        settl_type=symbol.settl_type,
        contract_size=symbol.contract_size,
        adt=symbol.adt,
        instr_type=symbol.instr_type,
    )
```

Репозиторий принимает и возвращает доменные модели. Внутри он конвертирует их как ему нужно.

```
def _db_to_domain_system(self, psystem: PSystem) -> Exchange:
    return Exchange(
        id=int(psystem.id),
        name=str(psystem.name),
        is_tradable=int(psystem.is_tradable)
    )

1 usage
def _domain_to_db_system(self, system: Exchange) -> PSystem:
    return PSystem(
        name=system.name,
        is_tradable=system.is_tradable
    )
```

Подытожим: Репозиторий — адаптер. Завтра поменяли базу — тесты не узнали.



- Ядро — доменные модели, доменные ошибки
- Репозитории — конкретные реализации протоколов
- Сервисы — бизнес-логика фреймворка



- Ядро — доменные модели, доменные ошибки
- Репозитории — конкретные реализации протоколов
- Сервисы — бизнес-логика фреймворка

Правило взаимодействия: верхние слои знают про нижние, нижние не знают про верхние

```
class IExchangeService:
    def get_instrument(self, system_name: t.Optional[str] = None, with_accrued: bool = False, **conditions) -> Symbol:
        """If you want to get instrument not from a selected exchange,
        indicate the required exchange in the parameters, if not, then just skip it.
```

```
class IExchangeService:
    def get_instrument(self, system_name: t.Optional[str] = None, with_accrued: bool = False, **conditions) -> Symbol:
        """If you want to get instrument not from a selected exchange,
        indicate the required exchange in the parameters, if not, then just skip it."""
```

```
class ExchangeServiceImpl(IExchangeService):
    def __init__(self, system_repository: ISystemRepository,
                 symbol_repository: ISymbolRepository,
                 static_data_repository: IStaticDataRepository) -> None:
        self.system_repository = system_repository
        self.symbol_repository = symbol_repository
        self.static_data_repository = static_data_repository
        self.system = None
```

```
def get_instrument(self, system_name: Optional[str] = None, **conditions) -> Symbol:
    """
    Service Layer: Получение инструмента с автоматической загрузкой
    всех зависимостей (статические данные, маркет данные, базовые инструменты)
    """
    system = self.system_repository.get_system(system_name)
    symbol = self.symbol_repository.get_symbol(system_id=system.id, **conditions)[0]

    # Загрузка статических данных
    static_data = self.static_data_repository.reload_symbol_static_data(symbol)
    symbol = type(symbol)(
        **{**symbol.model_dump(exclude_none=True), **static_data.model_dump(exclude_none=True)}
    )

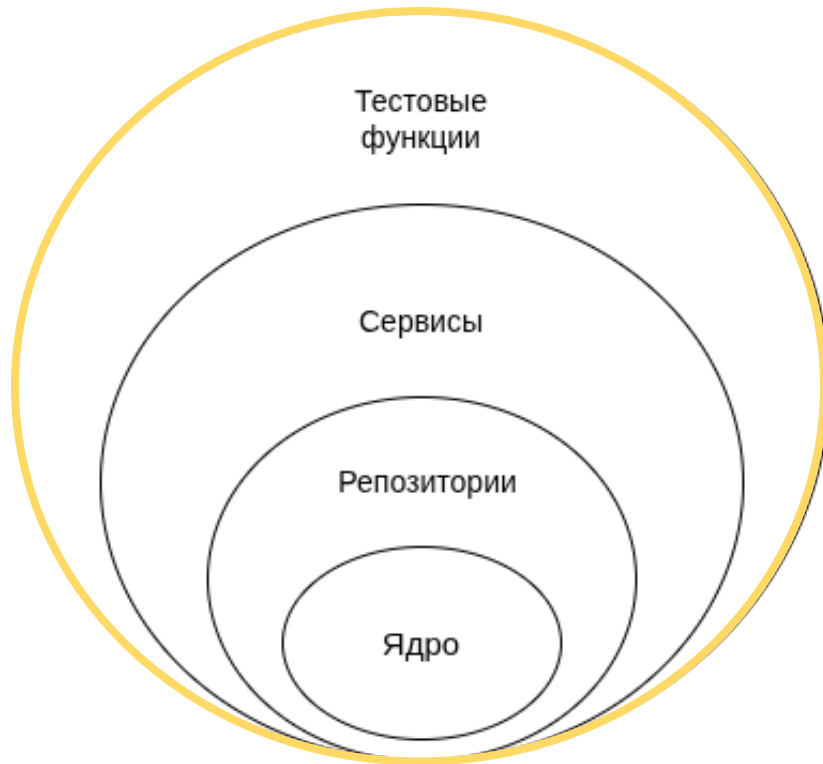
    return symbol
```

ExchangeService — управляет финансовыми Инструментами и всеми сущностями, связанными с Биржей.

Подытожим: Сервис отвечает на вопрос «что делаем», а не «как технически»

Луковая архитектура для тестового фреймворка

III. Архитектура: общая картина



- Ядро — доменные модели, доменные ошибки
- Репозитории — конкретные реализации протоколов
- Сервисы — бизнес-логика тестов
- Тестовые функции — верхний уровень, точка входа

Ручной E2E сценарий

II. Контекст продукта

1. Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.
2. Отправить заявку на покупку акций больше чем 1 миллион рублей.
3. Ожидаемый результат: система принимает заявку, видит превышение лимита, возвращает отказ.



Пример из начала

II. Контекст продукта

```
def test_order_rejected_when_limit_exceeded(db_cursor: Cursor, grpc_stub: LimitServiceStub) -> None:
    # Шаг 1: лезем в базу за инструментом
    db_cursor.execute(
        "SELECT symbol_id, symbol_name, board, lot_size, min_price_step "
        "FROM instruments.symbols "
        "WHERE symbol_name = 'Common1' AND board = 'TQBR'"
    )
    row = cursor.fetchone()
    symbol_id = row[0]
    symbol_name = row[1]
    board = row[2]
    lot_size = row[3]
    min_price_step = row[4]

    # Шаг 2: формируем gRPC-структуру и устанавливаем лимит
    limit_request = limit_pb2.SetLimitRequest(
        symbol_id=symbol_id,
        symbol_name=symbol_name,
        board=board,
        limit_type=limit_pb2.DAILY,
        amount=100_000_000,
        currency="RUB",
        effective_from="2024-01-01T00:00:00Z",
        effective_to="2024-12-31T23:59:59Z",
        created_by="autotest",
    )
    limit_response = grpc_stub.SetLimit(limit_request, timeout=5)
    assert limit_response.status == limit_pb2.SUCCESS
```

```

def test_order_rejected_when_limit_exceeded(db_cursor: Cursor, grpc_stub: LimitServiceStub,
fix_app: FixApplication) -> None:

    db_cursor.execute(
        "SELECT symbol_id, symbol_name, board, lot_size, min_price_step "
        "FROM instruments.symbols "
        "WHERE symbol_name = 'Common1' AND board = 'TOBR'"
    )
    row = cursor.fetchone()
    symbol_id = row[0]
    symbol_name = row[1]
    board = row[2]

    limit_request = limit_pb2.SetLimitRequest(
        symbol_id=symbol_id,
        symbol_name=symbol_name,
        board=board,
        limit_type=limit_pb2.DAILY,
        amount=1_000_000,
        currency="RUB",
        effective_from="2024-01-01T00:00:00Z",
        effective_to="2024-12-31T23:59:59Z",
        created_by="autotest",
    )
    limit_response = grpc_stub.SetLimit(limit_request, timeout=5)
    assert limit_response.status == limit_pb2.SUCCESS

    order = fix.Message()
    order.setField(fix.ClOrdID("order-001"))
    order.setField(fix.Symbol(symbol_name))
    order.setField(fix.SecurityExchange(board))
    order.setField(fix.Side(fix.Side_BUY))
    order.setField(fix.OrderQty(10_000))
    order.setField(fix.Price(268.53)) # volume: Price * OrderQty= 10_000 * 268.53 = 2_685_300.0

    order.setField(fix.OrdType(fix.OrdType_LIMIT))
    order.setField(fix.TransactTime())
    order.setField(fix.HandlInst("1"))
    order.setField(fix.Account("TEST_ACCOUNT"))

    fix_app.send_message(order)
    time.sleep(3)

    message = fix_app.pop_recieved_message(1)

    assert message.getField(fix.OrdStatus) == '8'
    assert message.getField(fix.ExecType) == '8'

    assert float(message.getField(fix.OrderVolume)) == 2_685_300.0
    assert "limit exceeded" in message.getField(fix.TEXT)

```

Было:

VII. От автотеста к автотесту

```

def test_order_rejected_when_limit_exceeded(db_cursor: Cursor, grpc_stub: LimitServiceStub,
                                           fix_app: FixApplication) -> None:
    db_cursor.execute(
        "SELECT symbol_id, symbol_name, board, lot_size, min_price_step "
        "FROM instruments.symbols "
        "WHERE symbol_name = 'Common1' AND board = 'TOBR'"
    )
    row = cursor.fetchone()
    symbol_id = row[0]
    symbol_name = row[1]
    board = row[2]

    limit_request = limit_pb2.SetLimitRequest(
        symbol_id=symbol_id,
        symbol_name=symbol_name,
        board=board,
        limit_type=limit_pb2.DAILY,
        amount=1_000_000,
        currency="RUB",
        effective_from="2024-01-01T00:00:00Z",
        effective_to="2024-12-31T23:59:59Z",
        created_by="autotest",
    )
    limit_response = grpc_stub.SetLimit(limit_request, timeout=5)
    assert limit_response.status == limit_pb2.SUCCESS

    order = fix.Message()
    order.setField(fix.ClOrdID("order-001"))
    order.setField(fix.Symbol(symbol_name))
    order.setField(fix.SecurityExchange(board))
    order.setField(fix.Side(fix.Side_BUY))
    order.setField(fix.OrderQty(10_000))
    order.setField(fix.Price(268.53)) # volume: Price * OrderQty= 10_000 * 268.53 = 2_685_300.0

    order.setField(fix.OrdType(fix.OrdType_LIMIT))
    order.setField(fix.TransactTime())
    order.setField(fix.HandlInst("1"))
    order.setField(fix.Account("TEST_ACCOUNT"))

    fix_app.send_message(order)
    time.sleep(3)

    message = fix_app.pop_received_message(1)

    assert message.getField(fix.OrdStatus) == '8'
    assert message.getField(fix.ExecType) == '8'

    assert float(message.getField(fix.OrderVolume)) == 2_685_300.0
    assert "limit exceeded" in message.getField(fix.TEXT)

```

Было:

VII. От автотеста к автотесту

0) Получить акцию.

```

def test_order_rejected_when_limit_exceeded(db_cursor: Cursor, grpc_stub: LimitServiceStub,
                                           fix_app: FixApplication) -> None:
    db_cursor.execute(
        "SELECT symbol_id, symbol_name, board, lot_size, min_price_step "
        "FROM instruments.symbols "
        "WHERE symbol_name = 'Common1' AND board = 'TQBR'"
    )
    row = cursor.fetchone()
    symbol_id = row[0]
    symbol_name = row[1]
    board = row[2]

    limit_request = limit_pb2.SetLimitRequest(
        symbol_id=symbol_id,
        symbol_name=symbol_name,
        board=board,
        limit_type=limit_pb2.DAILY,
        amount=1_000_000,
        currency="RUB",
        effective_from="2024-01-01T00:00:00Z",
        effective_to="2024-12-31T23:59:59Z",
        created_by="autotest",
    )
    limit_response = grpc_stub.SetLimit(limit_request, timeout=5)
    assert limit_response.status == limit_pb2.SUCCESS

    order = fix.Message()
    order.setField(fix.ClOrdID("order-001"))
    order.setField(fix.Symbol(symbol_name))
    order.setField(fix.SecurityExchange(board))
    order.setField(fix.Side(fix.Side_BUY))
    order.setField(fix.OrderQty(10_000))
    order.setField(fix.Price(268.53)) # volume: Price * OrderQty= 10_000 * 268.53 = 2_685_300.0

    order.setField(fix.OrdType(fix.OrdType_LIMIT))
    order.setField(fix.TransactTime())
    order.setField(fix.HandlInst("1"))
    order.setField(fix.Account("TEST_ACCOUNT"))

    fix_app.send_message(order)
    time.sleep(3)

    message = fix_app.pop_recieved_message(1)

    assert message.getField(fix.OrdStatus) == '8'
    assert message.getField(fix.ExecType) == '8'

    assert float(message.getField(fix.OrderVolume)) == 2_685_300.0
    assert "limit exceeded" in message.getField(fix.TEXT)

```

Было:

VII. От автотеста к автотесту

0) Получить акцию.

1) Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.

```

def test_order_rejected_when_limit_exceeded(db_cursor: Cursor, grpc_stub: LimitServiceStub,
                                           fix_app: FixApplication) -> None:
    db_cursor.execute(
        "SELECT symbol_id, symbol_name, board, lot_size, min_price_step "
        "FROM instruments.symbols "
        "WHERE symbol_name = 'Common1' AND board = 'TOBR'"
    )
    row = cursor.fetchone()
    symbol_id = row[0]
    symbol_name = row[1]
    board = row[2]

    limit_request = limit_pb2.SetLimitRequest(
        symbol_id=symbol_id,
        symbol_name=symbol_name,
        board=board,
        limit_type=limit_pb2.DAILY,
        amount=1_000_000,
        currency="RUB",
        effective_from="2024-01-01T00:00:00Z",
        effective_to="2024-12-31T23:59:59Z",
        created_by="autotest",
    )
    limit_response = grpc_stub.SetLimit(limit_request, timeout=5)
    assert limit_response.status == limit_pb2.SUCCESS

    order = fix.Message()
    order.setField(fix.ClOrdID("order-001"))
    order.setField(fix.Symbol(symbol_name))
    order.setField(fix.SecurityExchange(board))
    order.setField(fix.Side(fix.Side_BUY))
    order.setField(fix.OrderQty(10_000))
    order.setField(fix.Price(268.53)) # volume: Price * OrderQty= 10_000 * 268.53 = 2_685_300.0

    order.setField(fix.OrdType(fix.OrdType_LIMIT))
    order.setField(fix.TransactTime())
    order.setField(fix.HandlInst("1"))
    order.setField(fix.Account("TEST_ACCOUNT"))

    fix_app.send_message(order)
    time.sleep(3)

    message = fix_app.pop_recieved_message(1)

    assert message.getField(fix.OrdStatus) == '8'
    assert message.getField(fix.ExecType) == '8'

    assert float(message.getField(fix.OrderVolume)) == 2_685_300.0
    assert "limit exceeded" in message.getField(fix.TEXT)

```

Было:

VII. От автотеста к автотесту

0) Получить акцию.

1) Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.

2) Отправить заявку на покупку акций больше чем 1 миллион рублей.

```

def test_order_rejected_when_limit_exceeded(db_cursor: Cursor, grpc_stub: LimitServiceStub,
                                           fix_app: FixApplication) -> None:
    db_cursor.execute(
        "SELECT symbol_id, symbol_name, board, lot_size, min_price_step "
        "FROM instruments.symbols "
        "WHERE symbol_name = 'Common1' AND board = 'TQBR'"
    )
    row = cursor.fetchone()
    symbol_id = row[0]
    symbol_name = row[1]
    board = row[2]

    limit_request = limit_pb2.SetLimitRequest(
        symbol_id=symbol_id,
        symbol_name=symbol_name,
        board=board,
        limit_type=limit_pb2.DAILY,
        amount=1_000_000,
        currency="RUB",
        effective_from="2024-01-01T00:00:00Z",
        effective_to="2024-12-31T23:59:59Z",
        created_by="autotest",
    )
    limit_response = grpc_stub.SetLimit(limit_request, timeout=5)
    assert limit_response.status == limit_pb2.SUCCESS

    order = fix.Message()
    order.setField(fix.ClOrdID("order-001"))
    order.setField(fix.Symbol(symbol_name))
    order.setField(fix.SecurityExchange(board))
    order.setField(fix.Side(fix.Side_BUY))
    order.setField(fix.OrderQty(10_000))
    order.setField(fix.Price(268.53)) # volume: Price * OrderQty= 10_000 * 268.53 = 2_685_300.0

    order.setField(fix.OrdType(fix.OrdType_LIMIT))
    order.setField(fix.TransactTime())
    order.setField(fix.HandlInst("1"))
    order.setField(fix.Account("TEST_ACCOUNT"))

    fix_app.send_message(order)
    time.sleep(3)

    message = fix_app.pop_recieved_message(1)

    assert message.getField(fix.OrdStatus) == '8'
    assert message.getField(fix.ExecType) == '8'

    assert float(message.getField(fix.OrderVolume)) == 2_685_300.0
    assert "limit exceeded" in message.getField(fix.TEXT)

```

Было:

VII. От автотеста к автотесту

0) Получить акцию.

1) Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.

2) Отправить заявку на покупку акций больше чем 1 миллион рублей.

Ожидаемый результат: система принимает заявку, видит превышение лимита, возвращает отказ.

Стало:

VII. От автотеста к автотесту

```
def test_common_order_value_with_symbol_limit(order_limit_service: IOrderLimitService,
                                              order_service: IOrderService,
                                              micex_service: ExchangeServiceImpl) -> None:

    instrument = micex_service.get_instrument(
        currency='RUR',
        instr_type='Common',
        class_='TQBR'
    )

    order_limit_service.set_symbol_limit(
        symbol=instrument,
        max_volume=1_000_000,
        currency="RUB"
    )

    check_request = build_default_check_request(
        instrument=instrument,
        algo=Algo(),
        order_qty=10_000,
        price=268.53
    )

    response = order_service.send_order_exchange(check_request, execution_report=True)
    order_volume = OrderValueEquationSolver.solve_stock_order_value(
        order_qty=check_request.OrderQty,
        price=check_request.Price
    )

    assert_check_response(
        response,
        OrderVolume=order_volume,
        text=["OrderLimit error", "limit exceeded"]
    )
```

```

def test_common_order_value_with_symbol_limit(order_limit_service: IOrderLimitService,
                                              order_service: IOrderService,
                                              micex_service: ExchangeServiceImpl) -> None:
    instrument = micex_service.get_instrument(
        currency='RUR',
        instr_type='Common',
        class_='TQBR'
    )

    order_limit_service.set_symbol_limit(
        symbol=instrument,
        max_volume=1_000_000,
        currency="RUB"
    )

    check_request = build_default_check_request(
        instrument=instrument,
        algo=Algo(),
        order_qty=10_000,
        price=268.53
    )

    response = order_service.send_order_exchange(check_request, execution_report=True)
    order_volume = OrderValueEquationSolver.solve_stock_order_value(
        order_qty=check_request.OrderQty,
        price=check_request.Price
    )

    assert_check_response(
        response,
        OrderVolume=order_volume,
        text=["OrderLimit error", "limit exceeded"]
    )

```

Стало:

VII. От автотеста к автотесту

0) Получить акцию.

```
def test_common_order_value_with_symbol_limit(order_limit_service: IOrderLimitService,
                                              order_service: IOrderService,
                                              micex_service: ExchangeServiceImpl) -> None:

    instrument = micex_service.get_instrument(
        currency='RUR',
        instr_type='Common',
        class_='TQBR'
    )

    order_limit_service.set_symbol_limit(
        symbol=instrument,
        max_volume=1_000_000,
        currency="RUB"
    )

    check_request = build_default_check_request(
        instrument=instrument,
        algo=Algo(),
        order_qty=10_000,
        price=268.53
    )

    response = order_service.send_order_exchange(check_request, execution_report=True)
    order_volume = OrderValueEquationSolver.solve_stock_order_value(
        order_qty=check_request.OrderQty,
        price=check_request.Price
    )

    assert_check_response(
        response,
        OrderVolume=order_volume,
        text=["OrderLimit error", "limit exceeded"]
    )
```

Стало:

VII. От автотеста к автотесту

0) Получить акцию.

1) Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.

```
def test_common_order_value_with_symbol_limit(order_limit_service: IOrderLimitService,
                                              order_service: IOrderService,
                                              micex_service: ExchangeServiceImpl) -> None:

    instrument = micex_service.get_instrument(
        currency='RUR',
        instr_type='Common',
        class_='TQBR'
    )

    order_limit_service.set_symbol_limit(
        symbol=instrument,
        max_volume=1_000_000,
        currency="RUB"
    )

    check_request = build_default_check_request(
        instrument=instrument,
        algo=Algo(),
        order_qty=10_000,
        price=268.53
    )
    response = order_service.send_order_exchange(check_request, execution_report=True)
    order_volume = OrderValueEquationSolver.solve_stock_order_value(
        order_qty=check_request.OrderQty,
        price=check_request.Price
    )
    assert_check_response(
        response,
        OrderVolume=order_volume,
        text=["OrderLimit error", "limit exceeded"]
    )
```

Стало:

VII. От автотеста к автотесту

0) Получить акцию.

1) Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.

2) Отправить заявку на покупку акций больше чем 1 миллион рублей.

```
def test_common_order_value_with_symbol_limit(order_limit_service: IOrderLimitService,
                                             order_service: IOrderService,
                                             micex_service: ExchangeServiceImpl) -> None:

    instrument = micex_service.get_instrument(
        currency='RUR',
        instr_type='Common',
        class_='TQBR'
    )

    order_limit_service.set_symbol_limit(
        symbol=instrument,
        max_volume=1_000_000,
        currency="RUB"
    )

    check_request = build_default_check_request(
        instrument=instrument,
        algo=Algo(),
        order_qty=10_000,
        price=268.53
    )

    response = order_service.send_order_exchange(check_request, execution_report=True)

    order_volume = OrderValueEquationSolver.solve_stock_order_value(
        order_qty=check_request.OrderQty,
        price=check_request.Price
    )

    assert_check_response(
        response,
        OrderVolume=order_volume,
        text=["OrderLimit error", "limit exceeded"]
    )
```

Стало:

VII. От автотеста к автотесту

0) Получить акцию.

1) Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.

2) Отправить заявку на покупку акций больше чем 1 миллион рублей.

Ожидаемый результат: система принимает заявку, видит превышение лимита, возвращает отказ.

```

@pytest.mark.parametrize("limit_currency", ["RUB", "USD", "CNY"])
def test_common_order_value_with_symbol_limit(order_limit_service: IOrderLimitService,
                                              order_service: IOrderService,
                                              micex_service: ExchangeServiceImpl,
                                              limit_currency: str) -> None:

    instrument = micex_service.get_instrument(
        currency='RUR',
        instr_type='Common',
        class_='TOBR'
    )

    order_limit_service.set_symbol_limit(
        symbol=instrument,
        max_volume=1_000_000,
        currency=limit_currency
    )

    check_request = build_default_check_request(
        instrument=instrument,
        algo=Algo(),
        order_qty=10_000,
        price=268.53
    )
    response = order_service.send_order_exchange(check_request, execution_report=True)
    order_volume = OrderValueEquationSolver.solve_stock_order_value(
        order_qty=check_request.OrderQty,
        price=check_request.Price
    )
    assert_check_response(
        response,
        OrderVolume=order_volume,
        text=["OrderLimit error", "limit exceeded"]
    )

```

VII. От автотеста к
автотесту

Что это дает

- Быстрая параметризация;

Что это дает

- Быстрая параметризация;
- Переиспользование сервисов для интеграционного и функционального тестирования;

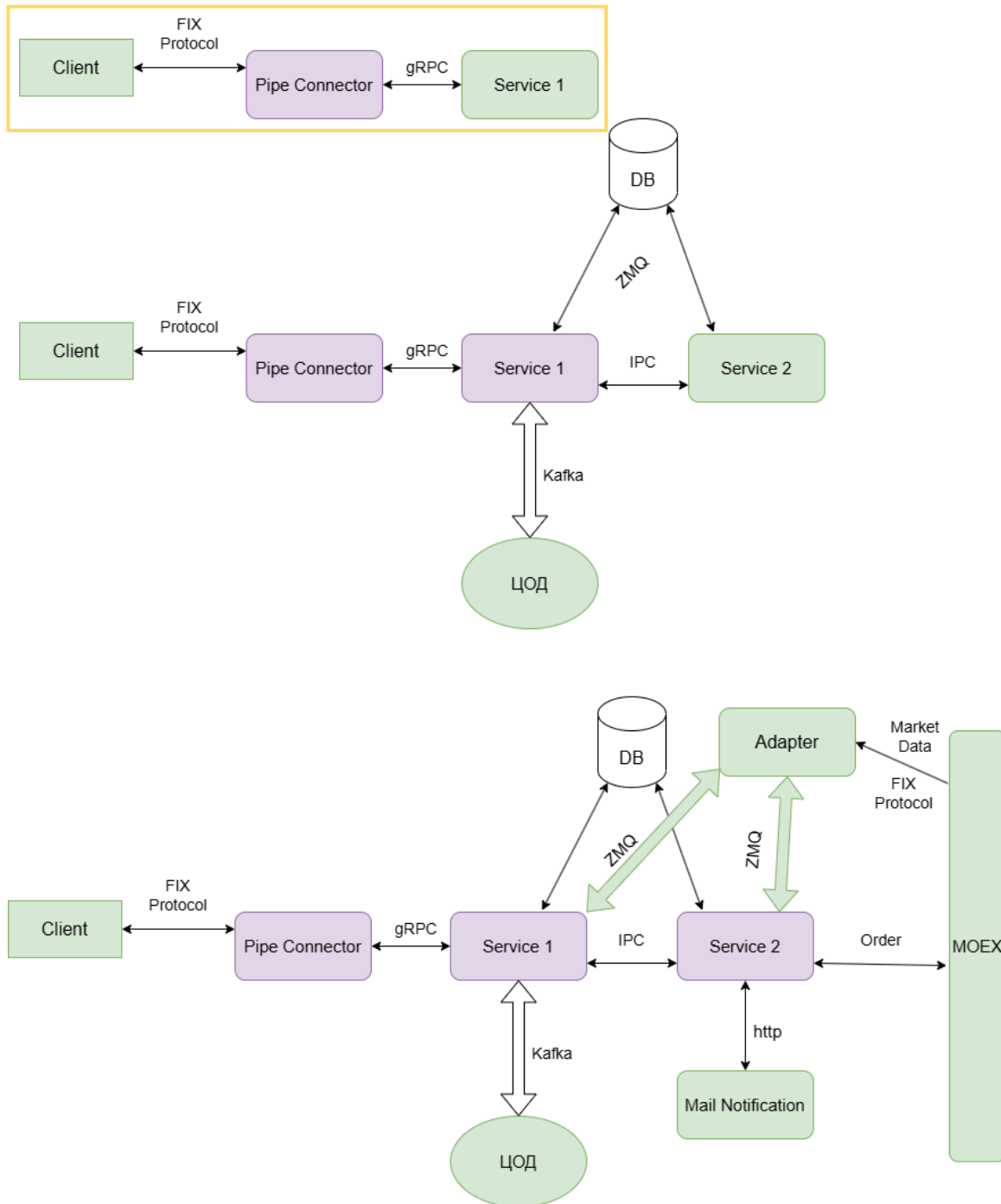
```
@pytest.fixture(scope="module")
def instrument(micex_service: ExchangeServiceImpl) -> Symbol:
    instrument = micex_service.get_instrument(
        currency='RUR',
        instr_type='Common',
        lot_size=10,
        class_='TOBR'
    )

    return instrument
```

```
@allure.title("Проверка AuditCheckRequest ордера, исполненного в полном объеме")
def test_send_check_request_check_audit(fix_pipe_initiator: Initiator, exchange_acceptor: Acceptor, instrument: Symbol,
                                         audit_check_request_repository: IAuditCheckRequestRepository, algo: Algo,
                                         trader: Trader) -> None:
```

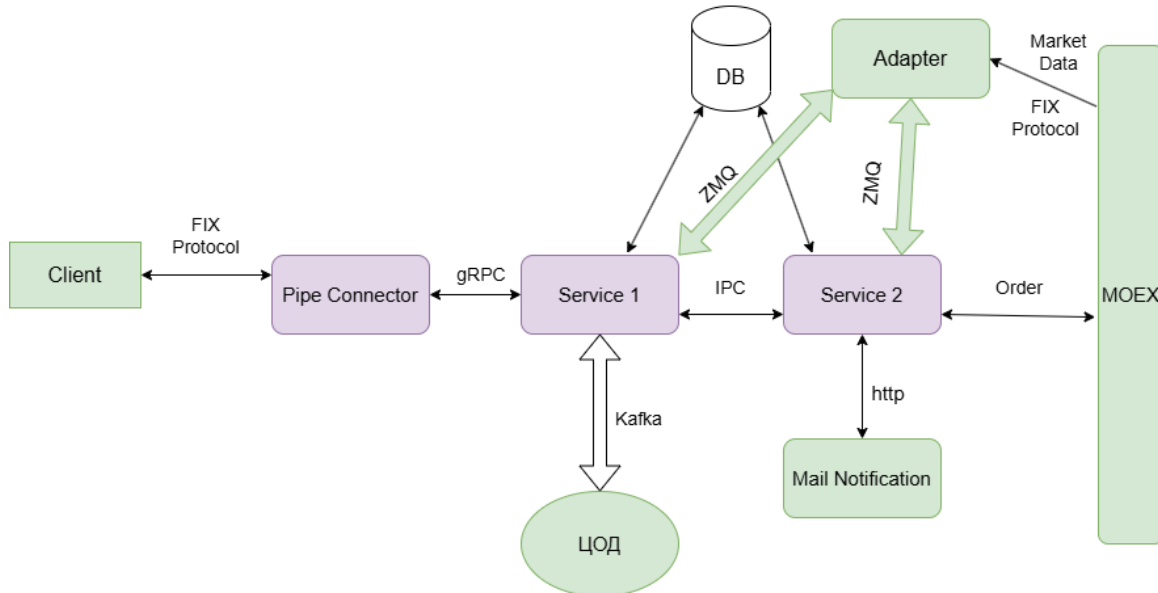
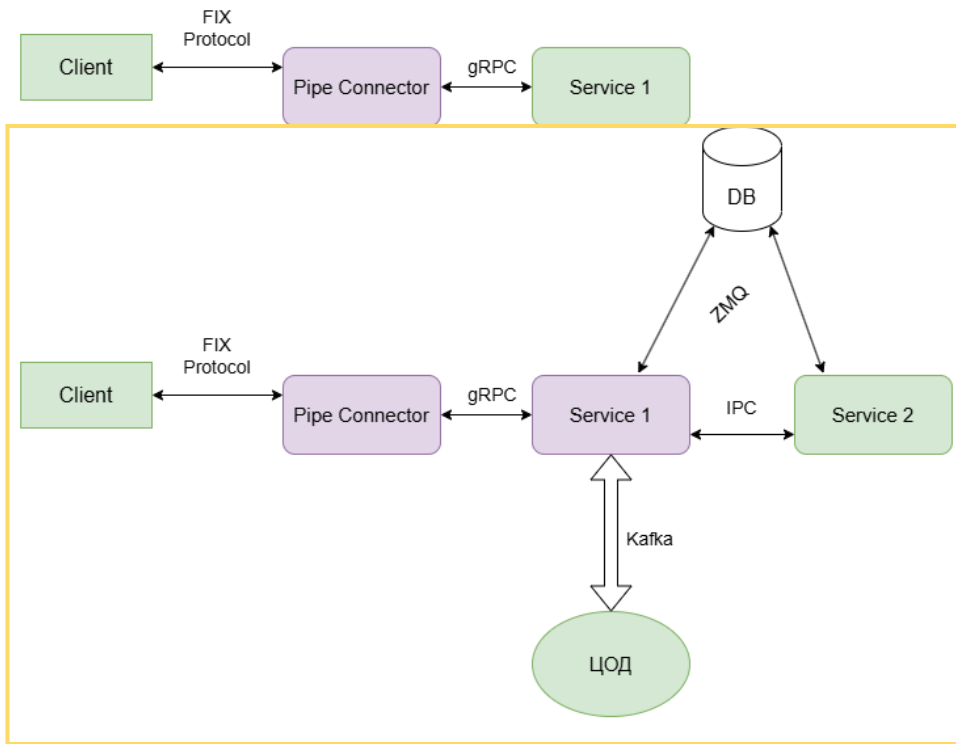
Что это дает

- Быстрая параметризация;
- Переиспользование сервисов для интеграционного и функционального тестирования;
- Большое количество проверок;



Что это дает

- Быстрая параметризация;
- Переиспользование сервисов для интеграционного и функционального тестирования;
- Большое количество проверок;

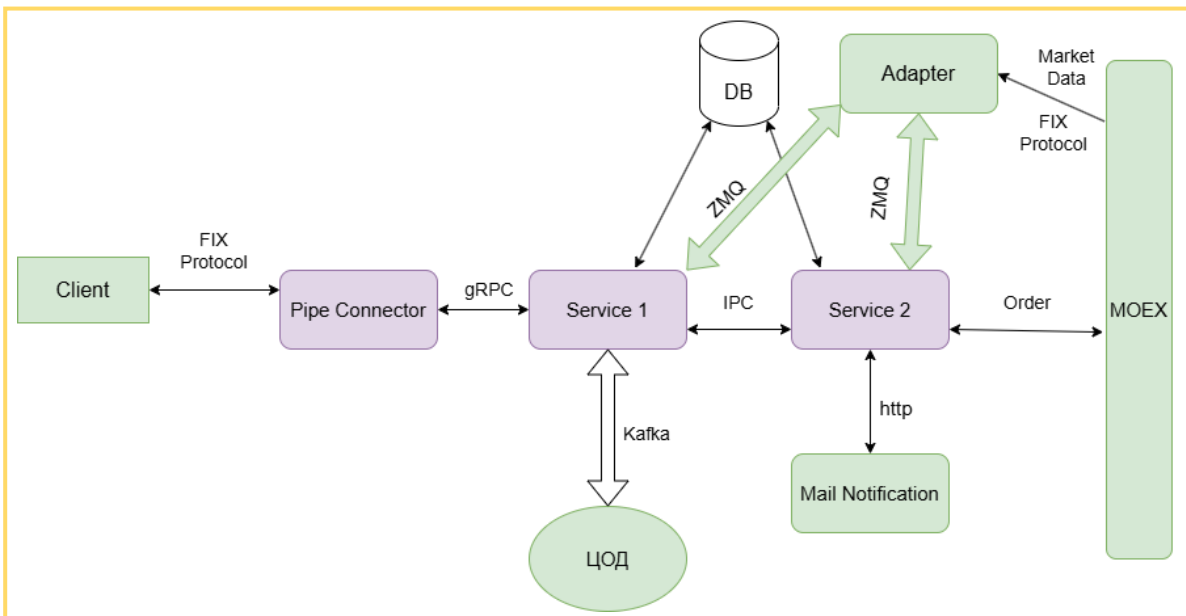
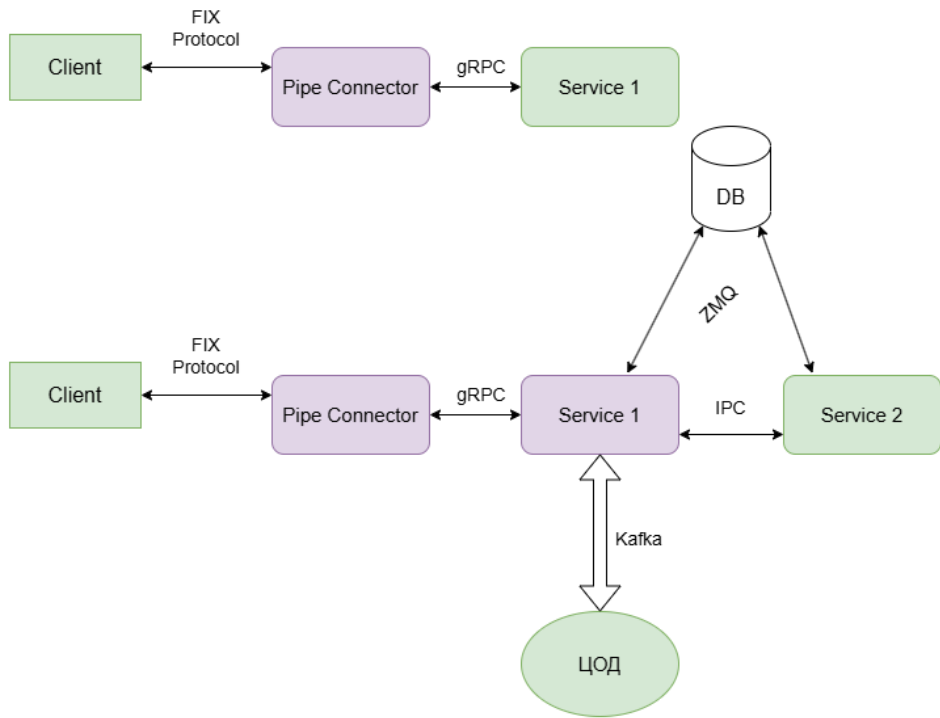


Зеленый - что мокаем

Фиолетовый - что тестируем

Что это дает

- Быстрая параметризация;
- Переиспользование сервисов для интеграционного и функционального тестирования;
- Большое количество проверок;



Зеленый - что мокаем

Фиолетовый - что тестируем

Луковая архитектура — это общее понятие, в отличие от разработки наш слой представления (тестовые функции) более гибкий и чаще всего имеет разный контекст разного уровня.

Соответственно, взаимодействие с тестом может быть менее изолировано от протокола.

Если наш тест будет выглядеть так:

1. Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.
2. Отправить по **FIX протоколу** ордер на покупку акций больше чем 1 миллион рублей.
3. Ожидаемый результат: система принимает заявку, видит превышение лимита, возвращает отказ.

Всё зависит от контекста

То наш автотест может звучать так:

```
def test_common_order_value_with_symbol_limit_2(order_limit_service: IOrderLimitService, fix_pipe_initiator: Initiator, micex_service: ExchangeServiceImpl) -> None:
    instrument = micex_service.get_instrument(currency='RUR', instr_type='Common', class_='IQBR')

    order_limit_service.set_symbol_limit(symbol=instrument, max_volume=1_000_000, currency="RUB")

    fix_message = FixMessageCheckRequestContext(
        build_default_fix_message(
            instrument=instrument,
            trader=trader,
            algo=algo
        )
    )
    fix_message.set_parties()

    fix_pipe_initiator.send(fix_message.new())
    response = fix_pipe_initiator.wait_message(
        1, timeout=15, condition=lambda x: x[0].OrdStatus == "8" and x[0].ExecType == "8"
    )

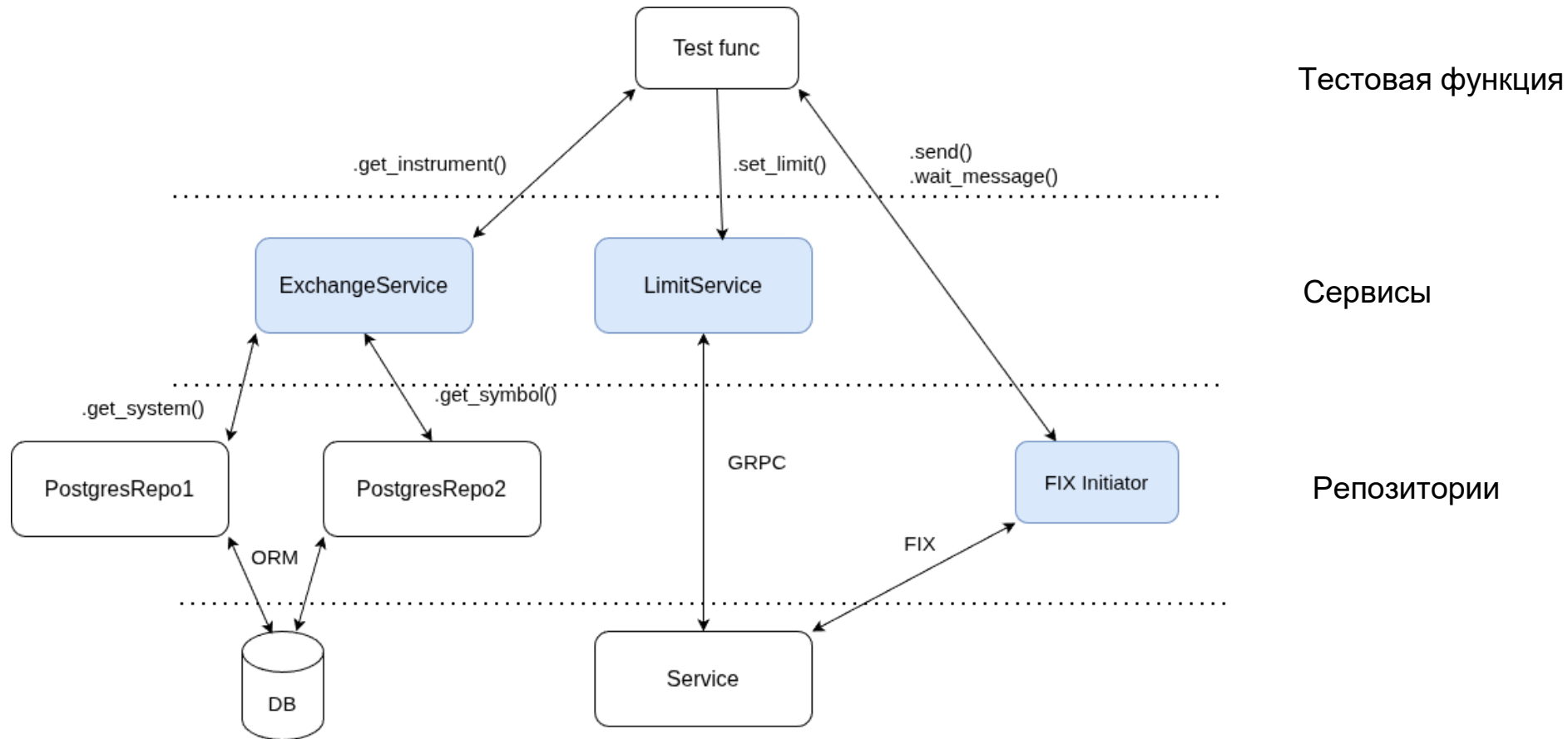
    order_volume = OrderValueEquationSolver.solve_stock_order_value(
        order_qty=check_request.OrderQty,
        price=check_request.Price
    )
    assert_check_response(response, OrderVolume=order_volume, text=["OrderLimit error", "limit exceeded"])
```

VIII. Гибкость контекста

1. Настроить систему:
установить лимит на
покупку акций за день — 1
миллион рублей.
2. Отправить по **FIX**
протоколу ордер на
покупку акций больше чем
1 миллион рублей.
3. Ожидаемый результат:
система принимает заявку,
видит превышение
лимита, возвращает отказ.

Всё зависит от контекста

VIII. Гибкость контекста



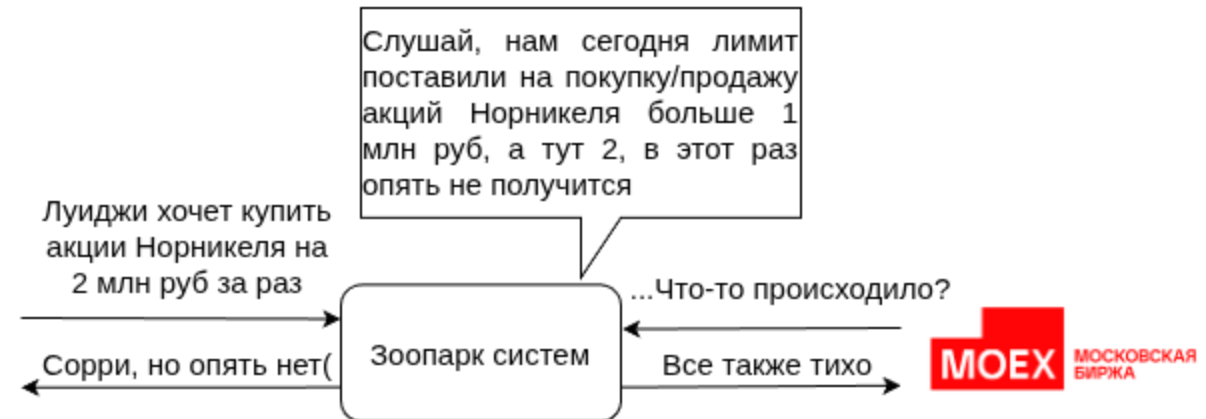
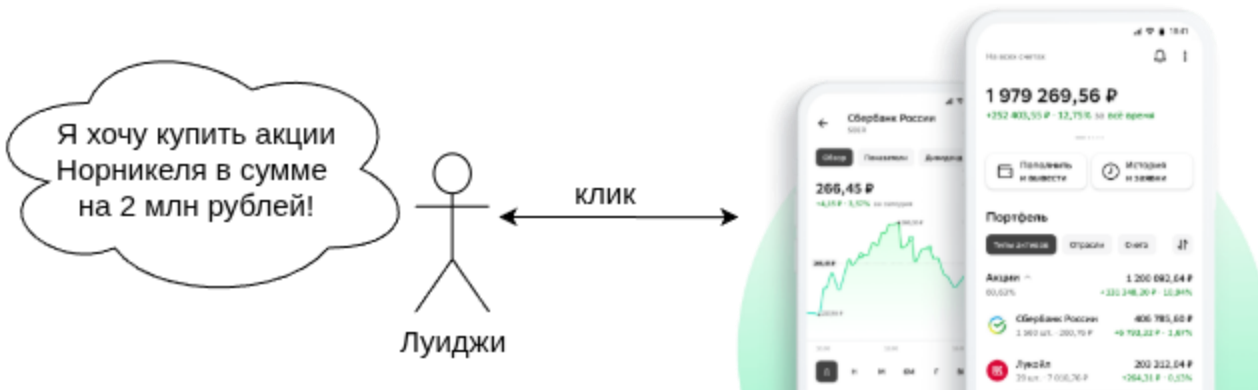
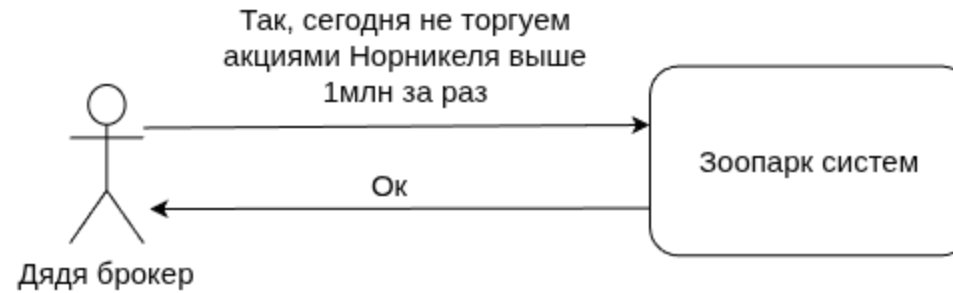
В одной тестовой функции и высокоуровневый `ExchangeService`, `LimitService` и низкоуровневый отправитель сообщений (`Initiator`) по FIX протоколу.

Подытожим: Луковая архитектура — руководство, не догма.

Количество слоёв определяет контекст теста.

Давайте пофантазируем

II. Контекст продукта



Как чистить за собой между тестами? IX. ArtifactStorage

Если первый тест установил лимит на акцию, а второй тест взял эту же акцию из базы и также работает с лимитами - результат может быть непредсказуем.

```
instrument = micex_service.get_instrument(currency='RUR', instr_type='Common', class_='TQBR')  
order_limit_service.set_symbol_limit(symbol=instrument, max_volume=1_000_000, currency="RUB")
```

Как чистить за собой между тестами? IX. ArtifactStorage

Если первый тест установил лимит на акцию, а второй тест взял эту же акцию из базы и также работает с лимитами - результат может быть непредсказуем.

```
instrument = micex_service.get_instrument(currency='RUR', instr_type='Common', class_='TQBR')  
order_limit_service.set_symbol_limit(symbol=instrument, max_volume=1_000_000, currency="RUB")
```

```
instrument = micex_service.get_instrument(currency='RUR', instr_type='Common', class_='TQBR')  
order_limit_service.set_symbol_limit(symbol=instrument, max_volume=1_000_000, currency="RUB")
```

Привет flaky тесты)

Как чистить за собой между тестами?

IX. ArtifactStorage

```
def test_common_order_value_with_symbol_limit_2(order_limit_service: IOrderLimitService, fix_pipe_initiator: Initiator, micex_service: ExchangeServiceImpl) -> None:
    instrument = micex_service.get_instrument(currency='RUR', instr_type='Common', class_='TQBR')
    order_limit_service.set_symbol_limit(symbol=instrument, max_volume=1_000_000, currency="RUB")

    fix_message = FixMessageCheckRequestContext(
        build_default_fix_message(
            instrument=instrument,
            trader=trader,
            algo=algo
        )
    )
    fix_message.set_parties()

    fix_pipe_initiator.send(fix_message.new())
    response = fix_pipe_initiator.wait_message(
        1, timeout=15, condition=lambda x: x[0].OrdStatus == "8" and x[0].ExecType == "8"
    )

    order_volume = OrderValueEquationSolver.solve_stock_order_value(
        order_qty=check_request.OrderQty,
        price=check_request.Price
    )
    assert_check_response(response, OrderVolume=order_volume, text=["OrderLimit error", "limit exceeded"])
```

Как teardown-фикстура узнает, что именно нужно почистить?

Artifact Storage

```
import typing as t

T = t.TypeVar('T')

__all__ = [
    "IocContainerImpl"
]

4 usages
class IocContainerImpl:
    __slots__ = "__instances"

    def __init__(self) -> None:
        self.__instances: t.MutableMapping[t.Type[T]][t.List[T]] = {}

6 usages (6 dynamic)
    def add_instances(self, key: t.Type[T], *values: T) -> None:
        if self.__instances.get(key) is None:
            self.__instances[key] = list(values)
        else:
            for value in values:
                self.__instances.get(key).append(value)

    def pop_instances(self, key: t.Type[T]) -> t.List[T]:
        try:
            return self.__instances.pop(key)
        except KeyError as e:
            return []
```

IX. ArtifactStorage

IoC-контейнер (ArtifactStorage) - синглтон со словарём:

- Ключ — интерфейс (тип артефакта)
- Значение — конкретный объект

Объект не создает нужные ему инструменты сам, а получает их извне в готовом виде.

Чтобы получить эти данные, другой заинтересованный объект должен их заранее положить - Dependency Injection (Внедрение зависимостей)

В наших тестах:

IX. ArtifactStorage

```
def set_symbol_limit(self, symbol: Symbol, max_volume: int, currency: str,
                    auction_coeff: float = 1.0, is_algo: int = 0) -> None:
    with allure.step(f"Установка лимита Инструмента {symbol.name}"):
        allure.attach(str(symbol), name="Инструмент", attachment_type=allure.attachment_type.TEXT)
        symbol_limit = TOrderValuePerSymbolSettings(
            initiator_username=self.initiator_username,
            system_name=symbol.system_name,
            symbol_name=symbol.name,
            symbol_class=symbol.class_,
            max_volume=max_volume,
            auction_factor=auction_coeff,
            currency=currency,
            is_algo=is_algo
        )
        allure.attach(str(symbol_limit), name="Лимит Инструмента", attachment_type=allure.attachment_type.TEXT)
        symbol_limit_with_addr = OrderValuePerSymbolSettingsAddr(
            orderValuePerSymbolSettings=symbol_limit,
            address=self.address,
        )
        try:
            logging_request_rpc("processOrderValueCheckerPerSymbol", symbol_limit_with_addr)
            self.grpc_stub.processOrderValueCheckerPerSymbol(symbol_limit_with_addr, timeout=self.default_timeout)
            artifact_storage.add_instances(OrderArtifact[Symbol], OrderArtifact(symbol=symbol))
        except Exception as e:
            raise GrpcServiceError(e)
```

```
class OrderArtifact(t.Generic[T]):
    __slots__ = ("exchange", "algo", "symbol")

    def __init__(self, exchange: t.Optional[Exchange] = None,
                 algo: t.Optional[Algo] = None,
                 symbol: t.Optional[Symbol] = None):
        self.exchange = exchange
        self.algo = algo
        self.symbol = symbol
```

```
artifact_storage.add_instances(OrderArtifact[Symbol], OrderArtifact(symbol=symbol))
```

При каждой установке лимита сервис кладет в ioc контейнер OrderArtifact с объектом Акции, на которую и установили лимит

У каждого свой тип артефакта в зависимости от контекста

В ExchangeService .get_instrument() перед возвращением объекта Symbol:

```
artifact_storage.add_instances[OrderArtifact[Symbol]], OrderArtifact(symbol=symbol))
```

```
class ExchangeArtifact(t.Generic[T]):  
    __slots__ = ("exchange", "symbol")  
  
    def __init__(self, exchange: t.Optional[Exchange] = None, symbol: t.Optional[Symbol] = None) -> None:  
        self.exchange = exchange  
        self.symbol = symbol
```

Перед и после выполнения каждого теста:

```
@pytest.fixture(autouse=True)
def clear_order_limit(order_limit_service: IOrderLimitService) -> None:
    exchange_artifacts = artifact_storage.pop_instances(ExchangeArtifact[Exchange])
    for exchange_artifact in exchange_artifacts:
        order_limit_service.clear_limit_per_exchange(exchange_artifact.exchange)

    exchange_artifact = artifact_storage.pop_instances(OrderArtifact[Exchange])
    for exchange_artifact in exchange_artifact:
        order_limit_service.clear_limit_per_exchange(exchange_artifact.exchange)

    order_limit_service.clear_default_order_limit()

    yield

    symbol_artifacts = artifact_storage.pop_instances(ExchangeArtifact[Symbol])
    for symbol_artifact in symbol_artifacts:
        order_limit_service.clear_limit_per_symbol(symbol_artifact.symbol)

    symbol_artifacts = artifact_storage.pop_instances(OrderArtifact[Symbol])
    for symbol_artifact in symbol_artifacts:
        order_limit_service.clear_limit_per_symbol(symbol_artifact.symbol)
```

Фикстура:

1. Получает артефакты из ioc-контейнера;
2. Подчищает лимит по соответствующим артефактам;
3. Возвращает систему в изначальное состояние, используя те же самые сервисы, что и сам тест.

Перед и после выполнения каждого теста:

```
@pytest.fixture(autouse=True)
def clear_order_limit(order_limit_service: IOrderLimitService) -> None:
    exchange_artifacts = artifact_storage.pop_instances(ExchangeArtifact[Exchange])
    for exchange_artifact in exchange_artifacts:
        order_limit_service.clear_limit_per_exchange(exchange_artifact.exchange)

    exchange_artifact = artifact_storage.pop_instances(OrderArtifact[Exchange])
    for exchange_artifact in exchange_artifact:
        order_limit_service.clear_limit_per_exchange(exchange_artifact.exchange)

    order_limit_service.clear_default_order_limit()

    yield

    symbol_artifacts = artifact_storage.pop_instances(ExchangeArtifact[Symbol])
    for symbol_artifact in symbol_artifacts:
        order_limit_service.clear_limit_per_symbol(symbol_artifact.symbol)

    symbol_artifacts = artifact_storage.pop_instances(OrderArtifact[Symbol])
    for symbol_artifact in symbol_artifacts:
        order_limit_service.clear_limit_per_symbol(symbol_artifact.symbol)
```

Фикстура:

1. Получает артефакты из ioc-контейнера;
2. Подчищает лимит по соответствующим артефактам;
3. Возвращает систему в изначальное состояние, используя те же самые сервисы, что и сам тест.

Перед и после выполнения каждого теста:

```
@pytest.fixture(autouse=True)
def clear_order_limit(order_limit_service: IOrderLimitService) -> None:
    exchange_artifacts = artifact_storage.pop_instances(ExchangeArtifact[Exchange])
    for exchange_artifact in exchange_artifacts:
        order_limit_service.clear_limit_per_exchange(exchange_artifact.exchange)

    exchange_artifact = artifact_storage.pop_instances(OrderArtifact[Exchange])
    for exchange_artifact in exchange_artifact:
        order_limit_service.clear_limit_per_exchange(exchange_artifact.exchange)

    order_limit_service.clear_default_order_limit()

    yield

    symbol_artifacts = artifact_storage.pop_instances(ExchangeArtifact[Symbol])
    for symbol_artifact in symbol_artifacts:
        order_limit_service.clear_limit_per_symbol(symbol_artifact.symbol)

    symbol_artifacts = artifact_storage.pop_instances(OrderArtifact[Symbol])
    for symbol_artifact in symbol_artifacts:
        order_limit_service.clear_limit_per_symbol(symbol_artifact.symbol)
```

Фикстура:

1. Получает артефакты из ioc-контейнера;
2. Подчищает лимит по соответствующим артефактам;
3. Возвращает систему в изначальное состояние, используя те же самые сервисы, что и сам тест.

Перед и после выполнения каждого теста:

```
@pytest.fixture(autouse=True)
def clear_order_limit(order_limit_service: IOrderLimitService) -> None:
    exchange_artifacts = artifact_storage.pop_instances(ExchangeArtifact[Exchange])
    for exchange_artifact in exchange_artifacts:
        order_limit_service.clear_limit_per_exchange(exchange_artifact.exchange)

    exchange_artifact = artifact_storage.pop_instances(OrderArtifact[Exchange])
    for exchange_artifact in exchange_artifact:
        order_limit_service.clear_limit_per_exchange(exchange_artifact.exchange)

    order_limit_service.clear_default_order_limit()

    yield

    symbol_artifacts = artifact_storage.pop_instances(ExchangeArtifact[Symbol])
    for symbol_artifact in symbol_artifacts:
        order_limit_service.clear_limit_per_symbol(symbol_artifact.symbol)

    symbol_artifacts = artifact_storage.pop_instances(OrderArtifact[Symbol])
    for symbol_artifact in symbol_artifacts:
        order_limit_service.clear_limit_per_symbol(symbol_artifact.symbol)
```

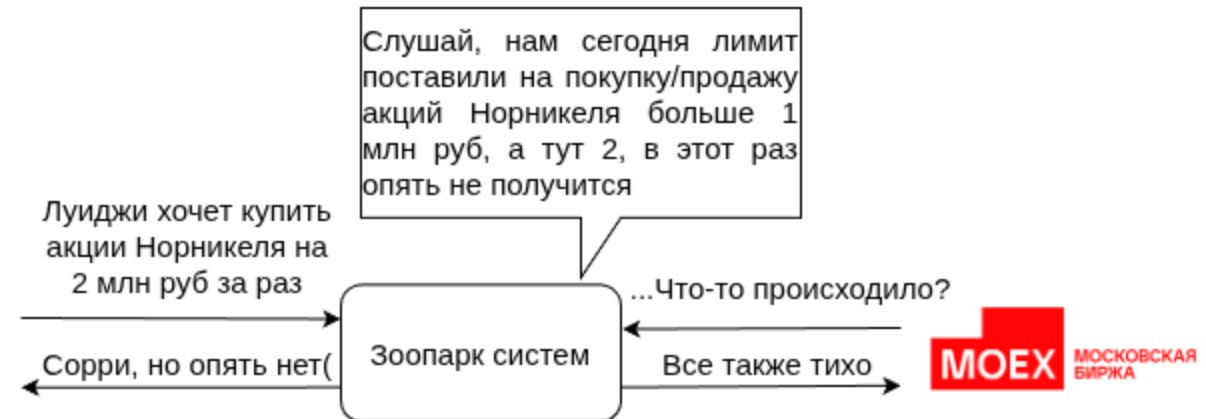
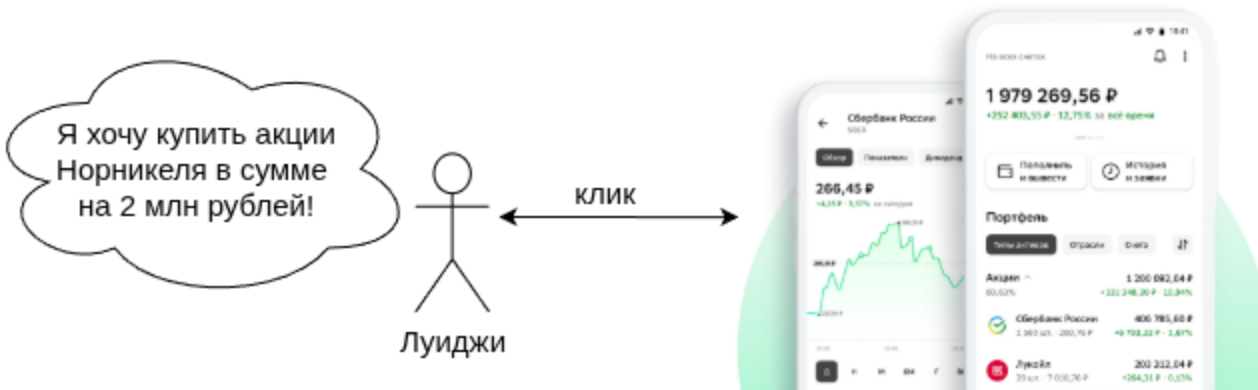
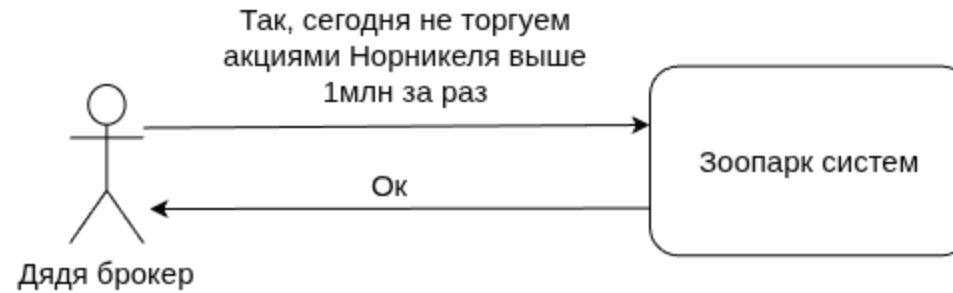
Фикстура:

1. Получает артефакты из ioc-контейнера;
2. Подчищает лимит по соответствующим артефактам;
3. Возвращает систему в изначальное состояние, используя те же самые сервисы, что и сам тест.

Подытожим: Сервис создал — сервис зарегистрировал —
фикстура почистила. Минус Flaky.

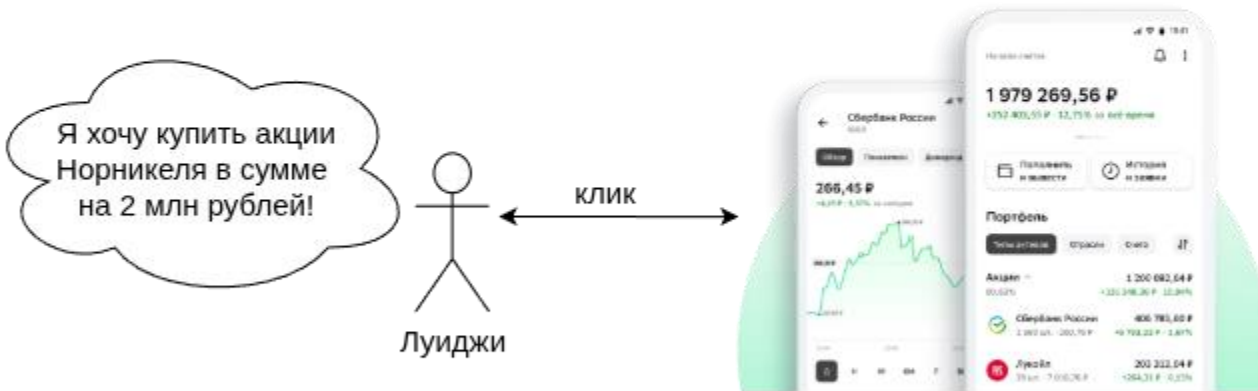
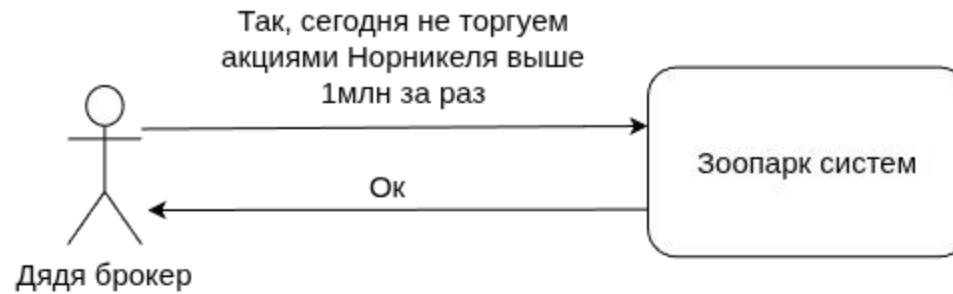
Давайте пофантазируем

II. Контекст продукта



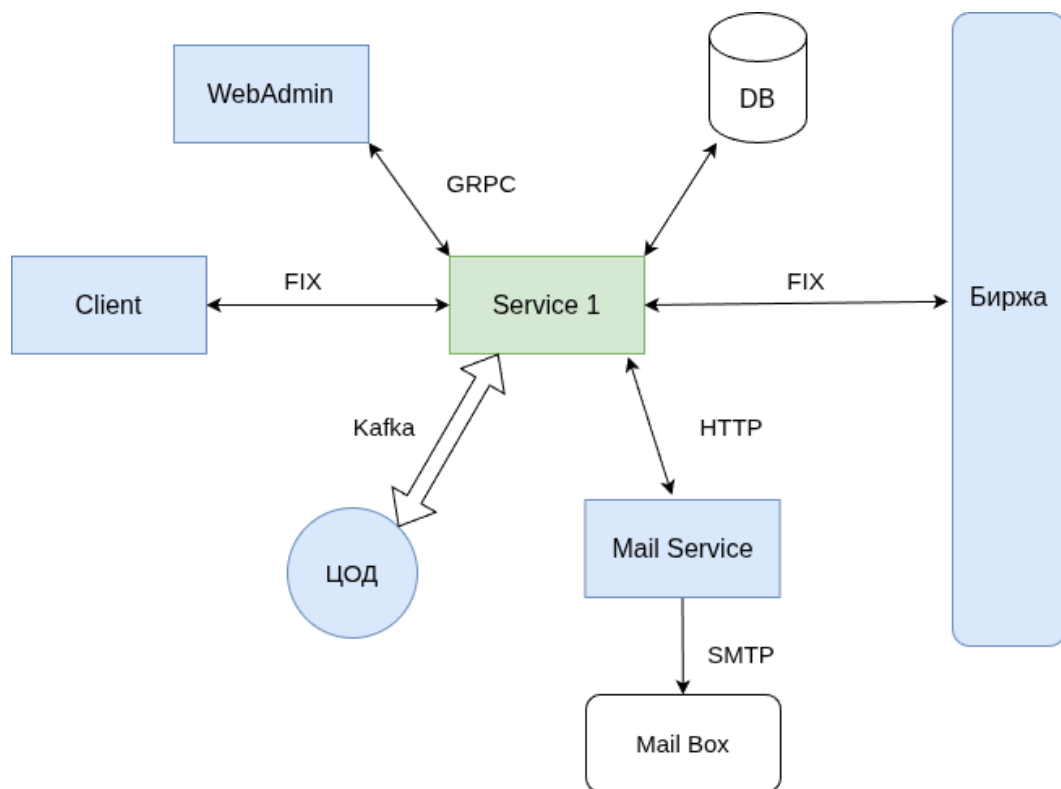
Давайте пофантазируем

II. Контекст продукта



Что можно замокать

Х. Моки



Зеленый - что тестируем

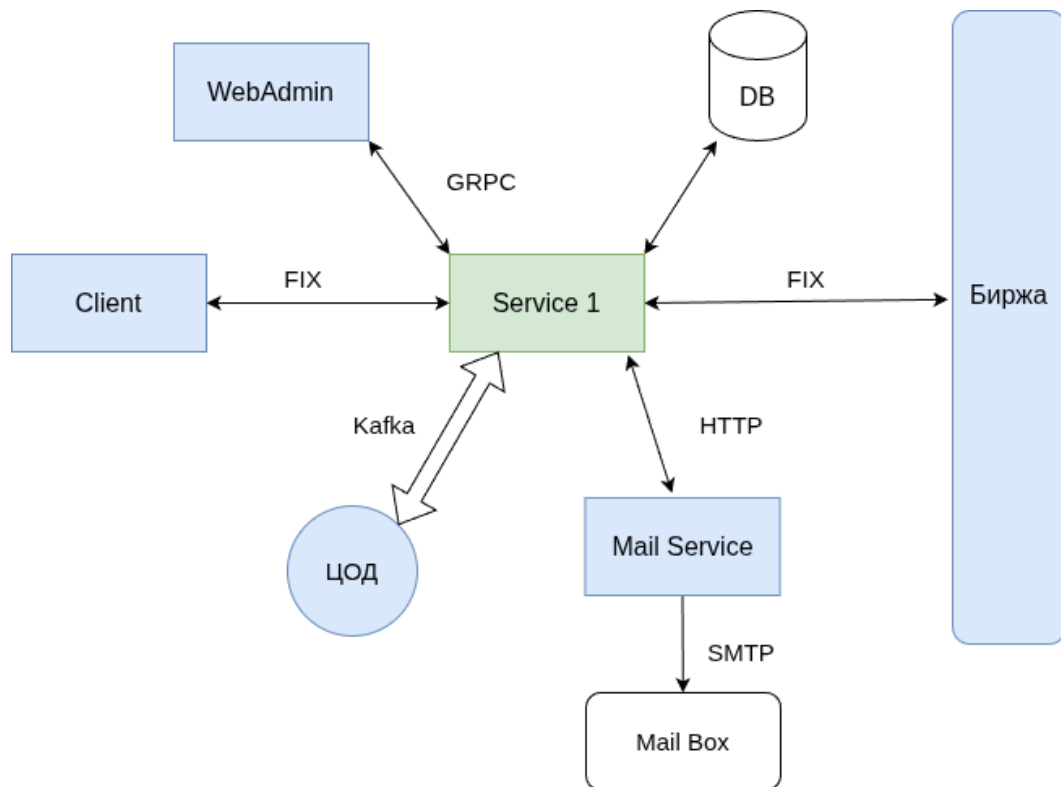
Синий - что мокаем

Что можно замочать

Х. Моки

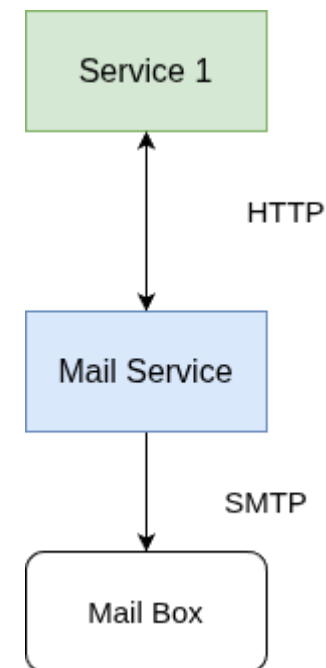
Зеленый - что тестируем

Синий - что мокаем



Мы можем ставить заглушки на разных уровнях и разных протоколах.

Это позволяет тестировать систему на разных уровнях, переиспользуя одну и ту же кодовую базу.



Ручной E2E сценарий

II. Контекст продукта

1. Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.
2. Отправить заявку на покупку акций больше чем 1 миллион рублей.
3. Ожидаемый результат: система принимает заявку, видит превышение лимита, возвращает отказ, отправляет на почтовый сервис сообщение о событии.



HTTP-МОК ДЛЯ ЛОВКИХ ТЕСТОВ

```
class ServerApplication:
    __slots__ = ("_app", "port", "response_method", "server", "run_thread", "request_history")

    def __init__(self, port: int = 8080) -> None:
        self._app = Flask(__name__)
        self._app.config.update(
            TEMPLATES_AUTO_RELOAD=True
        )
        self.port = port
        self.response_method: t.Callable[[t.Any], t.Tuple[t.Any, HTTPStatus]] | None = None
        self.server = waitress.create_server(self._app, host="0.0.0.0", port=self.port)
        self.run_thread = threading.Thread(target=self.server.run)
        self.request_history: t.List[t.Tuple[Request, Response]] = []

    def create_route(self, path: str, method: t.Literal['POST', 'PUT', 'GET', 'DELETE']) -> None:
        with allure.step(f"Создание {method} роута :{self.port}/{path}"):
            self._app.add_url_rule(path, view_func=self._index, methods=[method])

    def clear_request_history(self) -> None:
        self.request_history.clear()

    def set_response_method(self, response_method: t.Callable[[t.Any], t.Tuple[t.Any, HTTPStatus]]) -> None:
        func_ = inspect.getsource(response_method)
        with allure.step("Установка метода ответа"):
            self.response_method = response_method
            allure.attach(func_, name="Функция", attachment_type=allure.attachment_type.TEXT)
```

HTTP-МОК ДЛЯ ЛОВКИХ ТЕСТОВ

```
def wait_request(self, count: int = 1, timeout: int = 5) -> t.Tuple[Request, Response] | \
    t.List[t.Tuple[Request, Response]]:

    sleeping_second = 0.1
    wait_request_iteration = timeout / sleeping_second
    count_iter = itertools.count()
    requests = []
    with allure.step(f"Ожидание запроса"):
        while next(count_iter) <= wait_request_iteration:
            try:
                requests.append(self.request_history.pop())
                if len(requests) == count:
                    if count == 1:
                        allure.attach(str(requests[0][0]), name='Запрос')
                        allure.attach(str(requests[0][1]), name='Ответ')
                        return requests[0]
                    else:
                        for i in range(len(requests)):
                            with allure.step(f":{str(i+1)} запрос"):
                                allure.attach(str(requests[i][0]), name='Запрос')
                                allure.attach(str(requests[i][1]), name='Ответ')
                        return requests
            except IndexError:
                time.sleep(sleeping_second)
        raise RequestNotFound
```

Возможности:

- Создание роута на ходу;
- Мутация статус-кода ответа;
- Имитация задержки (time.sleep);
- Отправка произвольного тела ответа.

Сервис поверх мока

Х. Моки

Можно добавить еще один уровень абстракции и написать сервис поверх мока:

```
def mutable_response(self, status_code: int = None, body: str = None, delay: int = 0) -> None:
    with allure.step(f"Установка мутированного ответа на запрос"):
        attach_body = "" \
            + (f"status_code: {status_code}\n" if status_code is not None else "") \
            + (f"body: {body}\n" if body is not None else "") \
            + (f"delay: {delay}\n" if delay != 0 else "")
        allure.attach(body=attach_body, name="Параметры ответа на запрос",
            attachment_type=allure.attachment_type.TEXT)

    def response_method(*args):
        status = HTTPStatus.OK if status_code is None else status_code
        response_body = self._get_books() if body is None else body
        if delay != 0:
            time.sleep(delay)
        return response_body, status
    self._app.set_response_method(response_method=response_method)

def unmute_response(self) -> None:
    with allure.step("Установка валидного ответа на запрос"):
        self._app.set_response_method(response_method=self._get_books)

def add_trade_book(self, book: BookContract) -> None:
    with allure.step(f"Добавление BookContract {book.name}"):
        allure.attach(book, name="BookContract", attachment_type=allure.attachment_type.TEXT)
        self.books.append(book)

def update_trade_book(self, book_name: str, book: BookContract) -> None:
    with allure.step(f"Обновление BookContract {book_name}"):
        allure.attach(book, name="BookContract", attachment_type=allure.attachment_type.TEXT)
        for i in range(len(self.books)):
            if self.books[i].name == book_name:
                self.books[i] = book
                break
```

```
class BrdServiceImpl:
    __slots__ = ("_app", "books")

    def __init__(self, port: int) -> None:
        self._app = ServerApplication(port=port)
        self._app.create_route(path="/books/api/v1/integrations/algo/books", method='GET')
        self._app.set_response_method(response_method=self._get_books)
        self.books = []
        self._app.run()
```

Фикстура создания мока

Х. Моки

Или создать фикстуру, которая создает и возвращает мок, если не требуются сильные изменения состояния находу:

```
@allure.title(f"Инициализация Mail Mock")
@pytest.fixture(scope='session')
def mail_api_mock(mail_mock_port: int) -> ServerApplication:
    mail_mock = ServerApplication(port=mail_mock_port)
    mail_mock.create_route('/send', method='POST')

    mail_mock.set_response_method(lambda *x: ({}, HTTPStatus.OK))

    mail_mock.run()

    yield mail_mock

    mail_mock.stop()
```

Доп проверка к тесту Луиджи

1. Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.
2. Отправить заявку на покупку акций больше чем 1 миллион рублей.
3. Ожидаемый результат: система принимает заявку, видит превышение лимита, возвращает отказ, отправляет на почтовый сервис сообщение о событии.

```
def test_common_order_value_with_symbol_limit(order_limit_service: IOrderLimitService,
                                              order_service: IOrderService,
                                              micex_service: ExchangeServiceImpl,
                                              mail_api_mock: ServerApplication) -> None:

    instrument = micex_service.get_instrument(
        currency='RUR',
        instr_type='Common',
        class_='TOBR'
    )

    order_limit_service.set_symbol_limit(
        symbol=instrument,
        max_volume=1_000_000,
        currency="RUB"
    )

    check_request = build_default_check_request(
        instrument=instrument,
        algo=Algo(),
        order_qty=10_000,
        price=268.53
    )

    response = order_service.send_order_exchange(check_request, execution_report=True)
    order_volume = OrderValueEquationSolver.solve_stock_order_value(
        order_qty=check_request.OrderQty,
        price=check_request.Price
    )

    assert_check_response(
        response,
        OrderVolume=order_volume,
        text=["OrderLimit error", "limit exceeded"]
    )

    request, response = mail_api_mock.wait_request(timeout=30)
    assert_mail_message(
        mailer_node_config=mailer_node_config,
        request=request,
        template_name="checker_notification"
    )
```

```
def test_common_order_value_with_symbol_limit(order_limit_service: IOrderLimitService,
                                              order_service: IOrderService,
                                              micex_service: ExchangeServiceImpl,
                                              mail_api_mock: ServerApplication) -> None:

    instrument = micex_service.get_instrument(
        currency='RUR',
        instr_type='Common',
        class_='TOBR'
    )

    order_limit_service.set_symbol_limit(
        symbol=instrument,
        max_volume=1_000_000,
        currency="RUB"
    )

    check_request = build_default_check_request(
        instrument=instrument,
        algo=Algo(),
        order_qty=10_000,
        price=268.53
    )

    response = order_service.send_order_exchange(check_request, execution_report=True)
    order_volume = OrderValueEquationSolver.solve_stock_order_value(
        order_qty=check_request.OrderQty,
        price=check_request.Price
    )

    assert_check_response(
        response,
        OrderVolume=order_volume,
        text=["OrderLimit error", "limit exceeded"]
    )

    request, response = mail_api_mock.wait_request(timeout=30)
    assert_mail_message(
        mailer_node_config=mailer_node_config,
        request=request,
        template_name="checker_notification"
    )
```

Х. Моки

Доп проверка к тесту Луиджи

1. Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.
2. Отправить заявку на покупку акций больше чем 1 миллион рублей.
3. Ожидаемый результат: система принимает заявку, видит превышение лимита, возвращает отказ, отправляет на почтовый сервис сообщение о событии.

Доп проверка к тесту Луиджи

1. Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.
2. Отправить заявку на покупку акций больше чем 1 миллион рублей.
3. Ожидаемый результат: система принимает заявку, видит превышение лимита, возвращает отказ, отправляет на почтовый сервис сообщение о событии.

```
def test_common_order_value_with_symbol_limit(order_limit_service: IOrderLimitService,
                                              order_service: IOrderService,
                                              micex_service: ExchangeServiceImpl,
                                              mail_api_mock: ServerApplication) -> None:

    instrument = micex_service.get_instrument(
        currency='RUR',
        instr_type='Common',
        class_='TOBR'
    )

    order_limit_service.set_symbol_limit(
        symbol=instrument,
        max_volume=1_000_000,
        currency="RUB"
    )

    check_request = build_default_check_request(
        instrument=instrument,
        algo=Algo(),
        order_qty=10_000,
        price=268.53
    )

    response = order_service.send_order_exchange(check_request, execution_report=True)
    order_volume = OrderValueEquationSolver.solve_stock_order_value(
        order_qty=check_request.OrderQty,
        price=check_request.Price
    )

    assert_check_response(
        response,
        OrderVolume=order_volume,
        text=["OrderLimit error", "limit exceeded"]
    )

    request, response = mail_api_mock.wait_request(timeout=30)
    assert_mail_message(
        mailer_node_config=mailer_node_config,
        request=request,
        template_name="checker_notification"
    )
```

Доп проверка к тесту Луиджи

1. Настроить систему: установить лимит на покупку акций за день — 1 миллион рублей.
2. Отправить заявку на покупку акций больше чем 1 миллион рублей.
3. Ожидаемый результат: система принимает заявку, видит превышение лимита, возвращает отказ, отправляет на почтовый сервис сообщение о событии.

```
def test_common_order_value_with_symbol_limit(order_limit_service: IOrderLimitService,
                                              order_service: IOrderService,
                                              micex_service: ExchangeServiceImpl,
                                              mail_api_mock: ServerApplication) -> None:

    instrument = micex_service.get_instrument(
        currency='RUR',
        instr_type='Common',
        class_='TOBR'
    )

    order_limit_service.set_symbol_limit(
        symbol=instrument,
        max_volume=1_000_000,
        currency="RUB"
    )

    check_request = build_default_check_request(
        instrument=instrument,
        algo=Algo(),
        order_qty=10_000,
        price=268.53
    )

    response = order_service.send_order_exchange(check_request, execution_report=True)
    order_volume = OrderValueEquationSolver.solve_stock_order_value(
        order_qty=check_request.OrderQty,
        price=check_request.Price
    )

    assert_check_response(
        response,
        OrderVolume=order_volume,
        text=["OrderLimit error", "limit exceeded"]
    )

    request, response = mail_api_mock.wait_request(timeout=30)
    assert_mail_message(
        mailer_node_config=mailer_node_config,
        request=request,
        template_name="checker_notification"
    )
```

Подытожим: Мок с доменным интерфейсом = читаемый тест

Итоги

1. Код автоматизации заслуживает продвинутой архитектуры;
2. Тестовая функция $\sim$ ручной тест-кейс;
3. Доменные модели — единый язык фреймворка.
4. Количество слоев зависит от потребностей.
5. Умные моки с доменным интерфейсом как пример подхода.

Результат: фреймворк, который легко расширять, поддерживать и читать. Когда меняется реализация продукта — тесты остаются прежними.

Жду вопросы, спасибо!



Источники

- Pink Floyd: “Wish You Were Here”;
- <https://istqb.org/>: BENEFITS OF A GENERIC TEST AUTOMATION ARCHITECTURE;
- <https://www.moex.com/>: Московская Биржа — котировки, акции, облигации;
- <https://www.sberbank.ru/>: Сбербанк;
- <https://ibs.ru/>: IBS — ведущая российская IT-сервисная компания

V. Репозитории

```
def reload_symbol_static_data(self, symbol: Symbol) -> StaticData:
    instrument = Instrument(
        Symbol=symbol.name,
        Board=symbol.class_
    )
    try:
        response = self.grpc_stub.getStaticData(instrument, timeout=self.default_timeout)
    except Exception as e:
        raise GrpcRepositoryError(e)

    static_data = self._grpc_to_domain_static_data(response)
    return static_data
```

```
def _grpc_to_domain_static_data(self, static_data: GrpcStaticData) -> StaticData:
    static_data = message_to_dict(static_data)
    domain_static_data = StaticData().model_dump()

    repeated_static_data = {i: static_data[i] for i in static_data.keys()
                             if i in domain_static_data.keys()}
    domain_static_data.update(
        **repeated_static_data,
        currency=static_data.get("instrument_currency"),
        instr_type=static_data.get("type")
    )
    domain_static_data = StaticData(**domain_static_data)
    if static_data.get("accrued_info") is not None:
        accrued_info = [Accrued(
            offset=i.offset,
            amount=i.accrued
        ) for i in message_to_dict(static_data.get("accrued_info")).get("accrued")]
        domain_static_data.accrueds = accrued_info
    return domain_static_data
```

```
class StaticData(BaseModel):
    base_symbol: t.Optional[Symbol] = None |
    nominal_currency: t.Optional[str] = None
    accrueds: t.Optional[t.List[Accrued]] = None
    face_value: t.Optional[float] = None
    is_eurobond: t.Optional[float] = None
    currency: t.Optional[str] = None
    instr_type: t.Optional[str] = None
    lot_size: t.Optional[int] = None
    tick_size: t.Optional[float] = None
    tick_value: t.Optional[float] = None
    tick_value_curr: t.Optional[float] = None
    contract_size: t.Optional[int] = None
```

```
from shared_proto.guardian.grpc_for_qa import accrued_pb2 as

DESCRIPTOR = _descriptor_pool.Default().AddSerializedFile(b'\

_globals = globals()
_builder.BuildMessageAndEnumDescriptors(DESCRIPTOR, _globals)
_builder.BuildTopDescriptorsAndMessages(DESCRIPTOR, 'shared_p
if not _descriptor._USE_C_DESCRIPTORS:
    DESCRIPTOR._loaded_options = None
    _globals['_STATICDATA']._serialized_start=105
    _globals['_STATICDATA']._serialized_end=390
# @@protoc_insertion_point(module_scope)
```

Сервисы — уровень контроллеров

VI. Сервисы

OrderLimitService —
ответственен за установку
лимитов. Принимает на
вход доменные понятия:
модель и простые
переменные. Формирует
gRPC-структуру и
отправляет запрос.

```
class IOrderLimitService:
    def set_exchange_limit(self, exchange: Exchange, max_volume: int, currency: str) -> None:
        raise NotImplementedError

    def set_algo_limit(self, algo: Algo, exchange: Exchange, max_volume: int, currency: str, is_algo: int = 1) -> None:
        raise NotImplementedError

    def set_symbol_limit(self, symbol: Symbol, max_volume: int, currency: str,
                        auction_coeff: float = 1.0, is_algo: int = 0) -> None:
        raise NotImplementedError

    def set_default_order_limit(self, max_volume: int, is_algo: int = 0) -> None:
        raise NotImplementedError

    2 usages
    def clear_limit_per_exchange(self, exchange: Exchange, is_algo: int = 1) -> None:
        raise NotImplementedError

    1 usage
    def clear_limit_per_algo(self, algo: Algo, exchange: Exchange, is_algo: int = 1) -> None:
        raise NotImplementedError

    2 usages
    def clear_limit_per_symbol(self, symbol: Symbol, is_algo: int = 1) -> None:
        raise NotImplementedError

    1 usage
    def clear_default_order_limit(self, is_algo: int = 0) -> None:
        raise NotImplementedError
```

Сервисы — уровень контроллеров

VI. Сервисы

```
def set_symbol_limit(self, symbol: Symbol, max_volume: int, currency: str,
                    auction_coeff: float = 1.0, is_algo: int = 0) -> None:
    with allure.step(f"Установка лимита Инструмента {symbol.name}"):
        allure.attach(str(symbol), name="Инструмент", attachment_type=allure.attachment_type.TEXT)
        symbol_limit = TOrderValuePerSymbolSettings(
            initiator_username=self.initiator_username,
            system_name=symbol.system_name,
            symbol_name=symbol.name,
            symbol_class=symbol.class_,
            max_volume=max_volume,
            auction_factor=auction_coeff,
            currency=currency,
            is_algo=is_algo
        )
        allure.attach(str(symbol_limit), name="Лимит Инструмента", attachment_type=allure.attachment_type.TEXT)
        symbol_limit_with_addr = OrderValuePerSymbolSettingsAddr(
            orderValuePerSymbolSettings=symbol_limit,
            address=self.address,
        )
        try:
            logging_request_rpc("processOrderValueCheckerPerSymbol", symbol_limit_with_addr)
            self.grpc_stub.processOrderValueCheckerPerSymbol(symbol_limit_with_addr, timeout=self.default_timeout)
            artifact_storage.add_instances(OrderArtifact[Symbol], OrderArtifact(symbol=symbol))
        except Exception as e:
            raise GrpcServiceError(e)
```

Сервис — это объект, который инкапсулирует бизнес-логику и использует внутри необходимые источники данных.

Когда написание целого сервиса нецелесообразно из-за простой логики, тесты могут использовать классы-обертки сторонних библиотек.

Их основная задача:

- Инкапсуляция взаимодействия с библиотекой;
- Подгон логики библиотеки под конкретные задачи и специфику автотестов и продукта.

```
class KafkaConsumer:
    __slots__ = ("_consumer", "_topic_messages", "_convert_methods", "_poll_loop", "run_event")

    def __init__(self, host: str, port: int, **kwargs) -> None:
        address = f"{host}:{str(port)}"
        consumer_config = {
            "bootstrap.servers": address,
            "client.id": "python-consumer",
            "auto.offset.reset": "earliest",
            **kwargs
        }
        self._consumer = Consumer(consumer_config)
        self._topic_messages: t.MutableMapping[str, t.List[t.Tuple[str | None, t.Any]]] = {}
        self._convert_methods: t.MutableMapping[T, t.Callable[[bytes], T]] = {}
        self._poll_loop = threading.Thread(target=self._worker)
        self.run_event = threading.Event()

1 usage
    def _worker(self) -> None:
        while not self.run_event.is_set():
            message = self._consumer.poll(timeout=0.1)
            if type(message) is Message:
                topic, message, key = message.topic(), message.value(), message.key()
                if key is not None:
                    key = key.decode('utf-8')
                self._topic_messages[topic].append((key, message))
```

Kafka Consumer

```
def add_convert_method_model(self, *topics: str, method: t.Callable[[T], BaseModel]) -> None:
    for topic in topics:
        self._convert_methods[topic] = method

def clear_convert_method_model(self) -> None:
    self._convert_methods = {}

def wait_received_messages(self, topic: str, count: int = 1, timeout: int = 5,
                           condition: t.Callable[[T], bool] = None, key: str | None = None) -> t.List[T] | T:
    """
    This method works similarly to self.pop_received_messages(), but has the ability to catch messages by condition
    :param count: The number of messages to return. If count = -1, all received messages are returned.
    """
    sleeping_second = 0.1
    wait_message_iteration = timeout / sleeping_second
    count_iter = itertools.count()
    messages = []
    if count == -1:
        count = len(self._topic_messages.get(topic))
    while next(count_iter) <= wait_message_iteration:
        try:
            topic_messages = self.pop_received_messages(topic=topic, count=1, key=key)
            if condition is not None:
                try:
                    if condition(topic_messages[0]):
                        messages.append(topic_messages[0])
                except Exception:
                    continue
            else:
                messages.append(topic_messages[0])
            if len(messages) == count:
                if count == 1:
                    return messages[0]
                else:
                    return messages
        except MessageNotFound:
            time.sleep(sleeping_second)
    raise MessageNotFound
```

XI. Клиенты

```
def subscribe(self, topic_list: t.List[str]) -> None:
    self._consumer.subscribe(topic_list)
    for topic in topic_list:
        if self._topic_messages.get(topic) is None:
            self._topic_messages[topic] = []

2 usages (2 dynamic)

def unsubscribe(self) -> None:
    self._consumer.unsubscribe()

1 usage (1 dynamic)

def run(self) -> None:
    self._poll_loop.start()

3 usages (3 dynamic)

def stop(self) -> None:
    self.run_event.set()
    self._poll_loop.join()
    self._consumer.unsubscribe()
    self._consumer.close()
```

```
@allure.title("Инициализация Kafka консьюмера")
@pytest.fixture(scope='session')
def kafka_consumer(gmsb_host: str, gmsb_port: int) -> KafkaConsumer:
    consumer = KafkaConsumer(gmsb_host, gmsb_port, **{"group.id": "autotests"})
    consumer.run()

    yield consumer

    consumer.stop()
```

```
@pytest.fixture(scope='session', autouse=True)
def subscribe_ptc_deals(kafka_consumer: KafkaConsumer) -> None:
    kafka_consumer.subscribe([PTC_DEALS_TOPIC_NAME])
    kafka_consumer.add_convert_method_model(
        PTC_DEALS_TOPIC_NAME,
        method=lambda x: BatchFailedDeals.model_validate(xmltodict.parse(x)["FIXML"]["Batch"]["TrdCaptRpt"])
    )
```

```
message = kafka_consumer.wait_received_messages(
    topic=PTC_DEALS_TOPIC_NAME,
    condition=lambda x: x.Instrmt.Sym == rep_reject_report.security
)
```

Kafka Producer

XI. Клиенты

Возможности Consumer'а:

- Подписка на топик с автоматической десериализацией;
- Ожидание сообщения по условию.

Возможности Producer'а:

- Продюсирование с автоматической сериализацией;
- Динамическая конвертация.

```
class KafkaProducer:
    __slots__ = ("_producer", "_convert_methods")

    def __init__(self, host: str, port: int, **kwargs) -> None:
        config = {
            "bootstrap.servers": f"{host}:{str(port)}",
            "client.id": "python-producer",
            **kwargs
        }
        self._convert_methods: t.MutableMapping[T, t.Callable[[T], bytes]] = {}
        self._producer = Producer(config)

    def add_convert_method_model(self, *model_types: T, method: t.Callable[[T], bytes]) -> None:
        for model_type in model_types:
            self._convert_methods[model_type] = method

    def clear_convert_method_model(self) -> None:
        self._convert_methods = {}

    def produce(self, topic: str, message: T, key: str | None = None) -> None:
        convert_message = self._convert_methods.get(type(message))
        try:
            if convert_message is not None:
                message = convert_message(message)
            else:
                for key_, value_ in self._convert_methods.items():
                    if isinstance(message, key_):
                        message = value_(message)
                        break
                else:
                    message = str(message).encode()
        except Exception as e:
            raise ConvertMethodError(e)

        try:
            p_kwargs = {}
            if key is not None:
                p_kwargs = {"key": key.encode('utf-8')}
            self._producer.produce(topic, message, **p_kwargs)
            self._producer.flush()
        except Exception as e:
            raise ProducerError(e)
```

Подытожим: Не всё нужно обрачивать в сервис. Простая логика
— достаточно клиента-обёртки