

Вызовы Java экосистем в эру AI

Михаил Поливаха, Axelix Labs

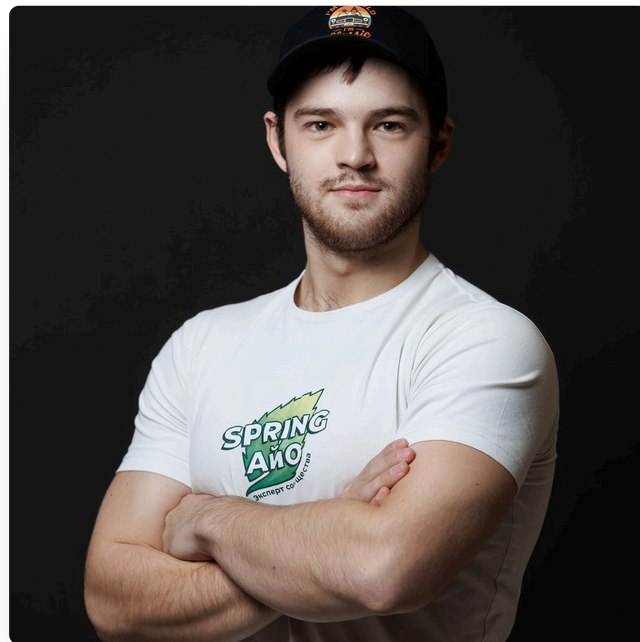
Михаил Поливаха

Open Source разработчик

- Создатель OSS-проекта **Axelix**
- Контрибьютор Spring Data
- Спикер Devoxx, Spring I/O и других конференций
- Технический автор на Baeldung и других ресурсах

Контакты:

- Telegram: <https://t.me/mipo256>



ЧАСТЬ 0

AI-агенты уже здесь

Давайте честно: этот факт невозможно отрицать.

Мы все пишем код с AI

84% разработчиков используют или планируют использовать AI-инструменты

Мы все пишем код с AI

84% разработчиков используют или планируют использовать AI-инструменты

90% — уровень внедрения AI в разработке по данным DORA, +14% за год

Мы все пишем код с AI

84% разработчиков используют или планируют использовать AI-инструменты

90% — уровень внедрения AI в разработке по данным DORA, +14% за год

75% нового кода в Google генерируется AI

Stack Overflow Developer Survey 2025 · Google DORA 2025 · Sundar Pichai, апрель 2026

Наша Java это особая экосистема

Десятилетиями — бэкенд по умолчанию для enterprise. Там, где работает реальный бизнес.

Стек, который у нас общий

Разные компании. Один и тот же короткий список технологий.



**Spring Framework ·
Boot**

основа всего



**Hibernate · Spring
Data**

работа с данными

Стек, который у нас общий

Разные компании. Один и тот же короткий список технологий.



**Spring Framework ·
Boot**

основа всего



**Hibernate · Spring
Data**

работа с данными



Jackson

JSON-маппинг

Стек, который у нас общий

Разные компании. Один и тот же короткий список технологий.



**Spring Framework ·
Boot**

основа всего



**Hibernate · Spring
Data**

работа с данными



Jackson

JSON-маппинг



Spring Cloud · Feign

HTTP-взаимодействие

Стек, который у нас общий

Разные компании. Один и тот же короткий список технологий.



**Spring Framework ·
Boot**

основа всего



**Hibernate · Spring
Data**

работа с данными



Jackson

JSON-маппинг



Spring Cloud · Feign

HTTP-взаимодействие

Десятилетиями оттачиваем один и тот же набор инструментов.

Один стек → одни и те же проблемы

Типичные проблемы с производительностью, безопасностью и конфигурацией **одинаковы** во всей Java-экосистеме.

Одни и те же грабли — в каждом проекте



Производительность — EAGER-загрузка, N+1, пагинация в памяти

Одни и те же грабли — в каждом проекте



Производительность — EAGER-загрузка, N+1, пагинация в памяти



Безопасность — открытые actuator-эндпоинты, стектрейсы в ответах

Одни и те же грабли — в каждом проекте



Производительность — EAGER-загрузка, N+1, пагинация в памяти



Безопасность — открытые actuator-эндпоинты, стектрейсы в ответах



Конфигурация — Open Session in View, **ddl-auto: update** на проде

Одни и те же грабли — в каждом проекте



Производительность — EAGER-загрузка, N+1, пагинация в памяти



Безопасность — открытые actuator-эндпоинты, стектрейсы в ответах



Конфигурация — Open Session in View, **ddl-auto: update** на проде



Зависимости — заброшенные и снятые с поддержки библиотеки

Три вызова, о которых мы поговорим



ВЫЗОВ 1

Качество

что на самом деле
уходит в прод

Три вызова, о которых мы поговорим



ВЫЗОВ 1

Качество

что на самом деле
уходит в прод



ВЫЗОВ 2

Инциденты

их больше —
дебажить надо
быстрее

Три вызова, о которых мы поговорим



ВЫЗОВ 1

Качество

что на самом деле
уходит в прод



ВЫЗОВ 2

Инциденты

их больше —
дебажить надо
быстрее



ВЫЗОВ 3

Зависимости

здоровье того, что у
нас в classpath

Знакомьтесь — команда

Один банк. Один платёжный сервис. Три истории.



Аня

Java-разработчик — пишет фичи вместе с AI-агентом

Знакомьтесь — команда

Один банк. Один платёжный сервис. Три истории.



Аня

Java-разработчик — пишет фичи вместе с AI-агентом



Денис

senior Java-разработчик — на этой неделе дежурит по прод

Знакомьтесь — команда

Один банк. Один платёжный сервис. Три истории.



Аня

Java-разработчик — пишет фичи вместе с AI-агентом



Денис

senior Java-разработчик — на этой неделе дежурит по прод



Макс

тимлид — пишет код, ревьюит, принимает технические решения

ВЫЗОВ 1

Что на самом деле уходит в прод?

История Ани.

AI-код – не бесплатный

1.7× больше проблем в PR, написанных AI, чем в PR, написанных людьми

AI-код – не бесплатный

1.7x больше проблем в PR, написанных AI, чем в PR, написанных людьми

1.4x больше **критических** проблем

CodeRabbit, State of AI vs Human Code Generation, дек. 2025

Понедельник — новая задача

МАКС · ТИМЛИД

Аня, антифроду нужно помечать платежи флагами риска, а ночной батч должен забирать их вместе с флагами. К пятнице успеешь? 🙏

Понедельник — новая задача

МАКС · ТИМЛИД

Аня, антифроду нужно помечать платежи флагами риска, а ночной батч должен забирать их вместе с флагами. К пятнице успеешь? 🙏

АНЯ

С агентом — к обеду 😎

Агент сгенерировал сущность

```
@Entity
public class Payment {

    @Id @GeneratedValue private Long id;

    private PaymentStatus status;

    @ManyToMany
    private List<RiskFlag> riskFlags;
}
```

Компилируется. Тесты зелёные. Выглядит как сотни примеров из интернета.

...и репозиторий для батча

```
interface PaymentRepository extends JpaRepository<Payment, Long> {  
  
    @Query("""  
        select p from Payment p  
        join fetch p.riskFlags  
        where p.status = :status  
        """)  
    List<Payment> findByStatus(PaymentStatus status, Pageable page);  
}
```

Батч забирает платежи порциями по 2 000 — сразу с флагами.

...и репозиторий для батча

```
interface PaymentRepository extends JpaRepository<Payment, Long> {  
  
    @Query("""  
        select p from Payment p  
        join fetch p.riskFlags  
        where p.status = :status  
        """)  
    List<Payment> findByStatus(PaymentStatus status, Pageable page);  
}
```

Батч забирает платежи порциями по 2 000 — сразу с флагами.



В среду код уже **в проде**.



Axelix

Через неделю – сообщение от DBA

ОЛЕГ · DBA

Аня, у вас на **payment_risk_flags** десятки тысяч
DELETE и INSERT в минуту. Что вы там делаете? 😐

Через неделю – сообщение от DBA

ОЛЕГ · DBA

Аня, у вас на **payment_risk_flags** десятки тысяч
DELETE и INSERT в минуту. Что вы там делаете? 😐

Антифрод добавляет флаги **по одному**. Откуда столько запросов?

Добавили один флаг

```
delete from payment_risk_flags where payment_id = 42;  
insert into payment_risk_flags (payment_id, risk_flag_id) values (42, 1);  
insert into payment_risk_flags (payment_id, risk_flag_id) values (42, 2);  
insert into payment_risk_flags (payment_id, risk_flag_id) values (42, 3);  
insert into payment_risk_flags (payment_id, risk_flag_id) values (42, 4);
```

Hibernate удалил **все** флаги платежа из join-таблицы — и вставил их заново, вместе с новым.

Виноват один List

```
@Entity
public class Payment {

    @Id @GeneratedValue private Long id;

    private PaymentStatus status;

    @ManyToMany
    private List<RiskFlag> riskFlags;
}
```

List в **@ManyToMany** Hibernate не умеет менять построчно — пересоздаёт всю коллекцию. С **Set** был бы **один** INSERT.

А по ночам – сообщение от Ops

СТЕПАН · OPS

Аня, **payments-service** каждую ночь в 02:00 съедает всю память и уходит в Full GC. Это ваш батч?

А по ночам — сообщение от Ops

СТЕПАН · OPS

Аня, **payments-service** каждую ночь в 02:00 съедает всю память и уходит в Full GC. Это ваш батч?

Метрики подтверждают: heap — под потолок, паузы GC по несколько секунд, батч идёт **втрое дольше**.

Батч просит 2 000 платежей...

```
interface PaymentRepository extends JpaRepository<Payment, Long> {  
  
    @Query("""  
        select p from Payment p  
        join fetch p.riskFlags  
        where p.status = :status  
        """)  
    List<Payment> findByStatus(PaymentStatus status, Pageable page);  
}
```

join fetch коллекции + **Pageable**: LIMIT в SQL не применить. Hibernate загружает **все** PENDING-платежи в память и режет страницу уже в Java.

Единственная подсказка

```
02:00:04.118  WARN  o.h.orm.query : HHH90003004: firstResult/maxResults  
specified with collection fetch;  
applying in memory!
```

Одна строчка WARN — среди миллионов строк логов. Её никто не читает.

Аня не плохой разработчик

Агент учился на среднем коде из интернета — и часто воспроизводит **те же самые** типичные ошибки. Только быстрее и в большем объёме.

ЖЕЛАНИЕ



«Было бы здорово определить такие проблемы заранее — до того, как их найдут DBA и метрики на проде.»

— Аня, после недели разборов

Вызов 1 — выводы



Агенты едят **много кода** — мультипликатор экспертизы.

Вызов 1 — выводы



Агенты пишут **много кода** — мультипликатор экспертизы.



Столько кода уже не отревьюить: ревью — **узкое место**.

Вызов 1 — выводы



Агенты пишут **много кода** — мультипликатор экспертизы.



Столько кода уже не отревьюить: ревью — **узкое место**.



Отсюда — **падение качества** того, что уходит в прод.

Вызов 1 — выводы



Агенты пишут **много кода** — мультипликатор экспертизы.



Столько кода уже не отревьюить: ревью — **узкое место**.



Отсюда — **падение качества** того, что уходит в прод.



Нужен инструмент, который **в масштабе** ловит типичные проблемы Java-экосистемы.

ВЫЗОВ 2

Инцидентов становится больше

История Дениса.

Больше кода – больше инцидентов

-7.2% стабильности поставки при росте внедрения AI на 25%

Больше кода — больше инцидентов

-7.2% стабильности поставки при росте внедрения AI на 25%

67% разработчиков тратят **больше** времени на отладку AI-кода

Больше кода — больше инцидентов

-7.2% стабильности поставки при росте внедрения AI на 25%

67% разработчиков тратят **больше** времени на отладку AI-кода

Инциденты будут. Вопрос — **как быстро** мы их разбираем.

Google DORA 2024 · Harness, State of Software Delivery 2025

09:12 – сообщение от коллеги

ЖЕНЯ · КОМАНДА ПЛАТЕЖЕЙ

Привет! Наш платёжный батч почему-то не обработал вчерашние платежи. Не знаешь почему? 😬

09:12 — сообщение от коллеги

ЖЕНЯ · КОМАНДА ПЛАТЕЖЕЙ

Привет! Наш платёжный батч почему-то не обработал вчерашние платежи. Не знаешь почему? 😬

Денис на этой неделе дежурный — разбираться ему. Он знает этот код: платежи обрабатывает задача **@Scheduled**, которая запускается по cron-выражению.

Код, который обрабатывает платежи

```
@Component
public class PaymentBatchJob {

    @Scheduled(cron = "${payments.batch.cron}")
    public void processPendingPayments() {
        var pending = payments.findByStatus(PENDING, Pageable.ofSize(2_000));
        payGrid.processAll(pending);
    }
}
```

Код, который обрабатывает платежи

```
@Component
public class PaymentBatchJob {

    @Scheduled(cron = "${payments.batch.cron}")
    public void processPendingPayments() {
        var pending = payments.findByStatus(PENDING, Pageable.ofSize(2_000));
        payGrid.processAll(pending);
    }
}
```

Задача **должна была** запускаться прошлой ночью в 02:00. Что произошло на самом деле?

Первые три вопроса



Задача вообще **запускалась**? Или сервис был недоступен в 02:00?

Первые три вопроса



Задача вообще **запускалась**? Или сервис был недоступен в 02:00?



Если запускалась — она **завершилась успешно**? Или упала?

Первые три вопроса



Задача вообще **запускалась**? Или сервис был недоступен в 02:00?



Если запускалась — она **завершилась успешно**? Или упала?



Если упала — **по какой причине**?

Первые три вопроса



Задача вообще **запускалась**? Или сервис был недоступен в 02:00?



Если запускалась — она **завершилась успешно**? Или упала?



Если упала — **по какой причине**?

Три простых вопроса. И **ни одного места** в системе, которое прямо на них отвечает.

Но у Дениса есть мониторинг — OpenSearch!

Представления по scheduled-задачам нет. Значит — дашборд OpenSearch.



Фильтр по имени логгера... сузить временной диапазон... скролл... скролл...

Но у Дениса есть мониторинг — OpenSearch!

Представления по scheduled-задачам нет. Значит — дашборд OpenSearch.



Фильтр по имени логгера... сузить временной диапазон... скролл... скролл...



Сопоставить время с деплоями и рестартами подов...

Проблема в логах

```
02:00:04.118  WARN    o.h.orm.query      : HHH90003004: firstResult/maxResults
                  specified with collection fetch;
                  applying in memory!

02:00:10.561  INFO    o.h.SQL_SLOW       : SlowQuery: 6459 ms. SQL: 'select ...'

02:00:10.584  INFO    PaymentBatchJob    : picked up 1,842 pending payments

02:01:34.518  ERROR    PayGridClient      : request to PayGrid failed:
                  I/O error on POST – read timed out
```

Проблема в логах

```
02:00:04.118  WARN    o.h.orm.query      : HHH90003004: firstResult/maxResults
                  specified with collection fetch;
                  applying in memory!

02:00:10.561  INFO     o.h.SQL_SLOW       : SlowQuery: 6459 ms. SQL: 'select ...'

02:00:10.584  INFO     PaymentBatchJob    : picked up 1,842 pending payments

02:01:34.518  ERROR    PayGridClient      : request to PayGrid failed:
                  I/O error on POST – read timed out
```



Пагинация в памяти и шестисекундный запрос... Знакомо? Это **репозиторий Ани**. Но сейчас — не до этого.

Проблема в логах

```
02:00:04.118  WARN    o.h.orm.query      : HHH90003004: firstResult/maxResults
                  specified with collection fetch;
                  applying in memory!

02:00:10.561  INFO    o.h.SQL_SLOW       : SlowQuery: 6459 ms. SQL: 'select ...'

02:00:10.584  INFO    PaymentBatchJob    : picked up 1,842 pending payments

02:01:34.518  ERROR   PayGridClient      : request to PayGrid failed:
                  I/O error on POST – read timed out
```

Настоящая проблема: внешний процессинг **PayGrid** не ответил. Наш код его вызывает, PayGrid проводит сами платежи.

Код, который вызывает PayGrid

```
@Bean
RestClient payGridRestClient(
    ApiTokenInterceptor tokenInterceptor,
    @Value("${paygrid.host}") String host) {
    return RestClient.builder()
        .baseUrl(host)
        .requestInterceptor(tokenInterceptor)
        .build();
}
```

11:05 — к созвону подключается поддержка PayGrid

ТИМУР · ПОДДЕРЖКА PAYGRID

Мы проверили у себя. Похоже, ваш запрос **до нас вообще не дошёл**. На какой хост вы стучались?

11:05 — к созвону подключается поддержка PayGrid

ТИМУР · ПОДДЕРЖКА PAYGRID

Мы проверили у себя. Похоже, ваш запрос **до нас вообще не дошёл**. На какой хост вы стучались?

Хороший вопрос. Денис знает, что хост берётся из **переменной окружения** — своей для каждого дата-центра, ради минимальной задержки.

На какой хост мы на самом деле стучались?

```
@Bean
RestClient payGridRestClient(
    ApiTokenInterceptor tokenInterceptor,
    @Value("${paygrid.host}") String host) {
    return RestClient.builder()
        .baseUrl(host)
        .requestInterceptor(tokenInterceptor)
        .build();
}
```

«Хост должен быть... вот этот. Наверное...?»

ЖЕЛАНИЕ №1



**«Просто покажите мне точный хост, на
который ушёл запрос, — в рантайме, на
этом самом инстансе.»**

— Денис, мечтательно

13:40 — поддержка PayGrid присылает детали

Денис всё ещё выясняет, на какой хост мы стучались... Но Тимур его опережает:

13:40 — поддержка PayGrid присылает детали

Денис всё ещё выясняет, на какой хост мы стучались... Но Тимур его опережает:

ТИМУР · ПОДДЕРЖКА PAYGRID

Нашли ваши запросы на нашем edge. Они были отклонены: к ним **не был приложен API-токен.**

13:40 — поддержка PayGrid присылает детали

Денис всё ещё выясняет, на какой хост мы стучались... Но Тимур его опережает:

ТИМУР · ПОДДЕРЖКА PAYGRID

Нашли ваши запросы на нашем edge. Они были отклонены: к ним **не был приложен API-токен**.

Нет токена? Но токен добавляется в **каждый** запрос к PayGrid интерсептором — чтобы PayGrid мог опознать вызывающего.

Кто добавляет токен?

```
@Bean
RestClient payGridRestClient(
    ApiTokenInterceptor tokenInterceptor,
    @Value("${paygrid.host}") String host) {
    return RestClient.builder()
        .baseUrl(host)
        .requestInterceptor(tokenInterceptor)
        .build();
}
```

Токен добавляет **ApiTokenInterceptor**. Денис толком не знает, как он устроен — этот код писал не он.

Интерсептор, который добавляет токен

```
class VaultApiTokenInterceptor implements ApiTokenInterceptor {  
  
    @Override  
    public ClientHttpResponse intercept(  
        HttpRequest request,  
        byte[] body,  
        ClientHttpRequestExecution execution) {  
  
        request.getHeaders().setBearerAuth(vault.currentApiToken());  
        return execution.execute(request, body);  
    }  
}
```

Добавляет API-токен и пропускает вызов дальше. Всё просто.

Вот только... это интерфейс

Денис открывает код и видит, что у **ApiTokenInterceptor** минимум две реализации.

```
interface ApiTokenInterceptor extends ClientHttpRequestInterceptor {  
    // просто интерфейс  
}
```

```
class VaultApiTokenInterceptor implements ApiTokenInterceptor {  
    // настоящий токен из Vault  
}
```

```
class NoOpApiTokenInterceptor implements ApiTokenInterceptor {  
    // не делает... ничего. Для тестов.  
}
```

Вот только... это интерфейс

Денис открывает код и видит, что у **ApiTokenInterceptor** минимум две реализации.

```
interface ApiTokenInterceptor extends ClientHttpRequestInterceptor {  
    // просто интерфейс  
}
```

```
class VaultApiTokenInterceptor implements ApiTokenInterceptor {  
    // настоящий токен из Vault  
}
```

```
class NoOpApiTokenInterceptor implements ApiTokenInterceptor {  
    // не делает... ничего. Для тестов.  
}
```

Похоже, на тестовых окружениях должна работать одна, а на проде — другая.

Ещё два вопроса



Какая из них **реально работает** прямо сейчас на проде?

Ещё два вопроса



Какая из них **реально работает** прямо сейчас на проде?



И главный вопрос: **почему?** Почему эта, а не другая?

Почему? Ответ — в аннотациях

Тестовые окружения

```
@Component
@Fallback
class NoOpApiTokenInterceptor
    implements ApiTokenInterceptor {

    // sandbox-окружениям
    // токен не нужен
}
```

Прод

```
@Component
@ConditionalOnProperty(
    name = "deploy.env",
    havingValue = "production")
class VaultApiTokenInterceptor
    implements ApiTokenInterceptor {

    // настоящий токен из Vault
}
```

Почему? Ответ — в аннотациях

Тестовые окружения

```
@Component
@Fallback
class NoOpApiTokenInterceptor
    implements ApiTokenInterceptor {

    // sandbox-окружениям
    // токен не нужен
}
```

Прод

```
@Component
@ConditionalOnProperty(
    name = "deploy.env",
    havingValue = "production")
class VaultApiTokenInterceptor
    implements ApiTokenInterceptor {

    // настоящий токен из Vault
}
```

Победил fallback → условие **не** выполнилось → **deploy.env** был **не "production"**. Тогда... а что там было?

ЖЕЛАНИЕ №2



«Покажите, какая реализация на самом деле победила и чем окончился каждый *@Conditional*. И почему.»

— Эх. Мечты

16:40 — Денис пишет Степану из Ops

ДЕНИС

Степан, откуда на проде берётся **deploy.env**? В нашем конфиге его нет.

16:40 — Денис пишет Степану из Ops

ДЕНИС

Степан, откуда на проде берётся **deploy.env**? В нашем конфиге его нет.

СТЕПАН · OPS

Это **DEPLOY_ENV** — наш Helm-чарт кладёт его в поды для тегов в логах. Недавно сменили значение на **prod**.

16:40 — Денис пишет Степану из Ops

ДЕНИС

Степан, откуда на проде берётся **deploy.env**? В нашем конфиге его нет.

СТЕПАН · OPS

Это **DEPLOY_ENV** — наш Helm-чарт кладёт его в поды для тегов в логах. Недавно сменили значение на **prod**.

ДЕНИС

От него зависит наш интерсептор. **Прод сломался.**

16:40 – Денис пишет Степану из Ops

СТЕПАН · OPS

Это наша внутренняя переменная для observability.

Зачем вы вообще на нее завязывались? 🤔

16:40 – Денис пишет Степану из Ops

СТЕПАН · OPS

Это наша внутренняя переменная для observability.

Зачем вы вообще на нее завязывались? 🤔

ДЕНИС

Не я этот код не писал... Автор давно **ушёл из компании.**

16:40 – Денис пишет Степану из Ops

СТЕПАН · OPS

Это наша внутренняя переменная для observability.
Зачем вы вообще на нее завязывались? 🤔

ДЕНИС

Не я этот код не писал... Автор давно **ушёл из компании.**

Переменная платформы молча стала частью контракта. И **никто об этом не знал.**

Фикс — одна строчка. *Наконец-то...*

```
@Component
@ConditionalOnProperty(
    name = "deploy.env",
    havingValue = "prod")    // было: "production"
class VaultApiTokenInterceptor implements ApiTokenInterceptor {
```

Причина **найдена**. План простой:

1 Проверить на sandbox → 2 задеплоить на прод → 3
перепровести платежи

17:30 – «просто проверь на sandbox»

Cron срабатывает в 02:00. Денис *не* собирается сидеть до 02:00.



Поменять **payments.batch.cron** на «через две минуты» → **перепланируй**

17:30 – «просто проверь на sandbox»

Cron срабатывает в 02:00. Денис *не* собирается сидеть до 02:00.



Поменять **payments.batch.cron** на «через две минуты» → **пердеплой**



Смотреть, как бегут логи... запустилась... токен на месте... вроде нормально?

17:30 – «просто проверь на sandbox»

Cron срабатывает в 02:00. Денис *не* собирается сидеть до 02:00.



Поменять **payments.batch.cron** на «через две минуты» → **перепланируй**



Смотреть, как бегут логи... запустилась... токен на месте... вроде нормально?



Вернуть cron **обратно** → **ещё один перепланируй**

17:30 – «просто проверь на sandbox»

Cron срабатывает в 02:00. Денис *не* собирается сидеть до 02:00.



Поменять **payments.batch.cron** на «через две минуты» → **переплоти**



Смотреть, как бегут логи... запустилась... токен на месте... вроде нормально?



Вернуть cron **обратно** → **ещё один переплоти**

Два переплота. Чтобы запустить **один метод, один раз.**



ЖЕЛАНИЕ №3



**«Одна кнопка:  запустить
processPendingPayments сейчас. И
показать — успешно ли и сколько заняло.»**

— Денис, нажимая воображаемую кнопку

19:05 — тот же танец, теперь на проде



Фикс на проде, cron-танец повторён — платежи **перепроведены**.

Цена: **целый рабочий день** — ради фикса в одну строчку.

Вызов 2 — выводы



AI **умножает** инциденты на проде. Это уже происходит — это факт.

Вызов 2 — выводы



AI **умножает** инциденты на проде. Это уже происходит — это факт.



Spring Boot — сложная машина: приоритеты источников свойств, **@Conditional**, сборка контекста на старте.

Вызов 2 — выводы



AI **умножает** инциденты на проде. Это уже происходит — это факт.



Spring Boot — сложная машина: приоритеты источников свойств, **@Conditional**, сборка контекста на старте.



Нужен инструмент отладки, который **понимает Spring Boot** изнутри.

ВЫЗОВ 3

Здоровье зависимостей

История Макса.

Число CVE стремительно растёт

~28 900 CVE опубликовано в 2023

Число CVE стремительно растёт

~28 900 CVE опубликовано в 2023

~40 000 в 2024 — **+39%**

Число CVE стремительно растёт

~28 900 CVE опубликовано в 2023

~40 000 в 2024 — **+39%**

~48 200 в 2025 — ещё **+21%**

Число CVE стремительно растёт

~28 900 CVE опубликовано в 2023

~40 000 в 2024 — **+39%**

~48 200 в 2025 — ещё **+21%**

AI находит уязвимости быстрее, чем когда-либо. NVD в 2026 перестал успевать обогащать все CVE.

Jerry Gamblin, CVE Data Review 2024/2025 · NIST, апрель 2026

Год назад

МАКС → АГЕНТ

Сделай workflow согласования крупных переводов:
создан → на проверке → одобрен / отклонён.

Год назад

МАКС → АГЕНТ

Сделай workflow согласования крупных переводов:
создан → на проверке → одобрен / отклонён.

АГЕНТ

Отличная задача для конечного автомата! Добавляю
spring-statemachine 🙌

Год назад

МАКС → АГЕНТ

Сделай workflow согласования крупных переводов:
создан → на проверке → одобрен / отклонён.

АГЕНТ

Отличная задача для конечного автомата! Добавляю
spring-statemachine 🙌

По Spring Statemachine — годы туториалов. Агент учился на них.

Агент сгенерировал конфигурацию

```
@Configuration
@EnableStateMachine
class TransferApprovalConfig
    extends StateMachineConfigurerAdapter<Status, Event> {

    @Override
    public void configure(
        StateMachineTransitionConfigurer<Status, Event> t)
        throws Exception {
        t.withExternal().source(CREATED).target(IN_REVIEW).event(SUBMIT);
    }
}
```

Работает. Тесты зелёные. Фича в проде больше года.

Чего не знали ни агент, ни Макс



Апрель 2025: Spring объявил — Statemachine больше **не поддерживается** как open source

Чего не знали ни агент, ни Макс



Апрель 2025: Spring объявил — Statemachine больше **не поддерживается** как open source



Все будущие релизы и патчи — только для **платных** клиентов Tanzu Spring

Чего не знали ни агент, ни Макс



Апрель 2025: Spring объявил — Statemachine больше **не поддерживается** как open source



Все будущие релизы и патчи — только для **платных** клиентов Tanzu Spring

Чего не знали ни агент, ни Макс



Апрель 2025: Spring объявил — Statemachine больше **не поддерживается** как open source



Все будущие релизы и патчи — только для **платных** клиентов Tanzu Spring

Найдут CVE завтра — в open source его **никто не исправит**.

spring.io/blog, 21.04.2025 · github.com/spring-projects/spring-statemachine

Июнь 2026 – CVE-2026-41862

СЛУЖБА БЕЗОПАСНОСТИ

Макс, в **spring-statemachine** критическая уязвимость:
CVE-2026-41862, CVSS 8.8. Срок на устранение — 7
дней.

Июнь 2026 – CVE-2026-41862

СЛУЖБА БЕЗОПАСНОСТИ

Макс, в **spring-statemachine** критическая уязвимость:
CVE-2026-41862, CVSS 8.8. Срок на устранение — 7
дней.

Это не выдумка: реальная уязвимость, опубликованная Spring в июне 2026.

Что сейчас может сделать Макс



Форкнуть и пропатчить самому — и поддерживать форк вечно

Что сейчас может сделать Макс



Форкнуть и пропатчить самому — и поддерживать форк вечно



Выпилить библиотеку из платёжного workflow — в аварийном режиме

Что сейчас может сделать Макс



Форкнуть и пропатчить самому — и поддерживать форк вечно



Выпилить библиотеку из платёжного workflow — в аварийном режиме

О том, что библиотека умерла, Макс узнал **из CVE**. Слишком поздно.

Это не единичный случай

18.6%

Java- и JS-проектов, поддерживавшихся в 2022, через год уже **не поддерживались**

Это не единичный случай

18.6%

Java- и JS-проектов, поддерживавшихся в 2022, через год уже **не поддерживались**

80%

зависимостей приложений не обновляются **больше года**

Sonatype, State of the Software Supply Chain 2023 / 2024

ЖЕЛАНИЕ



**«Скажите мне, что библиотека умирает, —
за год до CVE, а не через день после.»**

— Макс, в аварийном режиме

Вызов 3 — выводы



CVE становится больше — и находят их всё быстрее.

Вызов 3 — выводы



CVE становится больше — и находят их всё быстрее.



Агенты охотно тянут библиотеки из своего **training set**. Часть из этих библиотек уже не поддерживается

Вызов 3 — выводы



CVE становится больше — и находят их всё быстрее.



Агенты охотно тянут библиотеки из своего **training set**. Часть из этих библиотек уже не поддерживается



О проблеме с зависимостью узнаём, только когда уже **горит**.

Три истории – три желания



Аня

«Покажите
известные
ловушки — до
прода»

Три истории – три желания



Аня

«Покажите
известные
ловушки — до
прода»



Денис

«Покажите, что
происходит
внутри — прямо
сейчас»

Три истории — три желания



Аня

«Покажите
известные
ловушки — до
прода»



Денис

«Покажите, что
происходит
внутри — прямо
сейчас»



Макс

«Предупредите,
что библиотека
умирает — до
CVE»

РЕШЕНИЕ

Что с этим делать?

Три истории. Один инструмент.



**Инструмент для Spring Boot в эру AI — с
открытым ядром.**

Исходный код:

github.com/axelixlabs/axelix

Всё, что вы сегодня увидите, доступно в **Open Source**



**Открытое ядро: исходный код — на GitHub,
github.com/axelixlabs/axelix**

Всё, что вы сегодня увидите, доступно в **Open Source**



Открытое ядро: исходный код — на GitHub,
github.com/axelixlabs/axelix



Скачать и поставить можно **уже сегодня** бесплатно.

Всё, что вы сегодня увидите, доступно в **Open Source**



Открытое ядро: исходный код — на GitHub,
github.com/axelixlabs/axelix



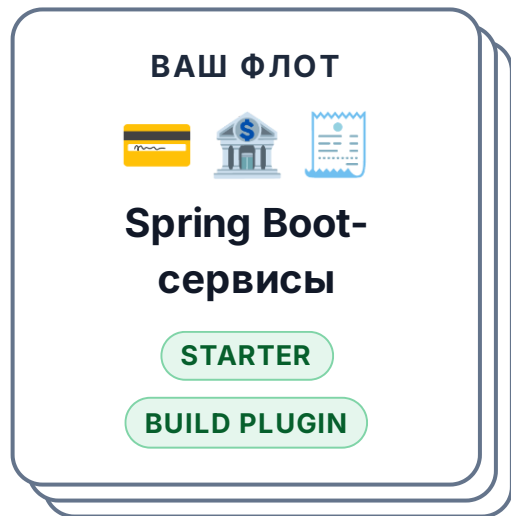
Скачать и поставить можно **уже сегодня** бесплатно.



90%+ того, что будет в демо, уже в релизе. Остальное выйдет — тоже в open source.

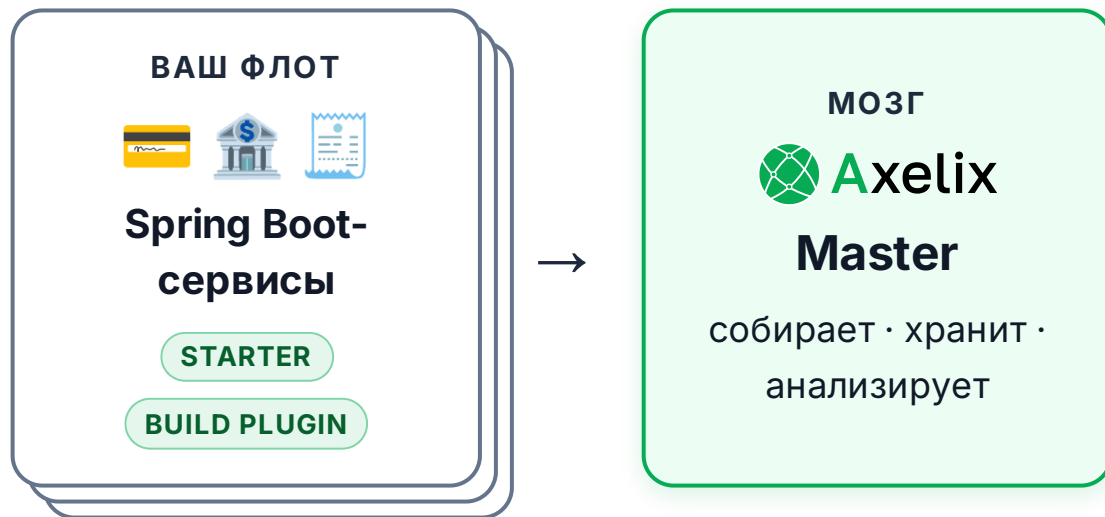
Как работает Axelix

Один центральный «мозг» и маленькие клиенты рядом с вашими приложениями.



Как работает Axelix

Один центральный «мозг» и маленькие клиенты рядом с вашими приложениями.



Как работает Axelix

Один центральный «мозг» и маленькие клиенты рядом с вашими приложениями.



Как работает Axelix

Один центральный «мозг» и маленькие клиенты рядом с вашими приложениями.



Нужны **оба**: **starter** в приложении + **build plugin** в сборке.

ЖЕЛАНИЕ АНИ

Axelix знает типичные ловушки

Потому что Spring Boot-проекты очень похожи друг на друга:



Карта сущностей — EAGER, List в @ManyToMany, опасные каскады

ЖЕЛАНИЕ АНИ

Axelix знает типичные ловушки

Потому что Spring Boot-проекты очень похожи друг на друга:



Карта сущностей — EAGER, List в @ManyToMany, опасные каскады



Транзакции — N+1, пагинация в памяти, сеть внутри @Transactional

ЖЕЛАНИЕ АНИ

Axelix знает типичные ловушки

Потому что Spring Boot-проекты очень похожи друг на друга:



Карта сущностей — EAGER, List в @ManyToMany, опасные каскады



Транзакции — N+1, пагинация в памяти, сеть внутри @Transactional



Опасные свойства — open-in-view, ddl-auto, exposure.include=*

ЖЕЛАНИЕ АНИ

Axelix знает типичные ловушки

Потому что Spring Boot-проекты очень похожи друг на друга:



Карта сущностей — EAGER, List в @ManyToMany, опасные каскады



Транзакции — N+1, пагинация в памяти, сеть внутри @Transactional



Опасные свойства — open-in-view, ddl-auto, exposure.include=*

Все ловушки из истории Ани — **на одном экране.**

ЖЕЛАНИЯ ДЕНИСА

Axelix: заглянуть внутрь Spring Boot

Рентген для работающего приложения:



Свойства — откуда пришло значение, что активно, что устарело

ЖЕЛАНИЯ ДЕНИСА

Axelix: заглянуть внутрь Spring Boot

Рентген для работающего приложения:



Свойства — откуда пришло значение, что активно, что устарело



Бины — что создано, откуда взялось и кто куда внедрён

ЖЕЛАНИЯ ДЕНИСА

Axelix: заглянуть внутрь Spring Boot

Рентген для работающего приложения:



Свойства — откуда пришло значение, что активно, что устарело



Бины — что создано, откуда взялось и кто куда внедрён



@Conditional — что вычислялось, что сработало и почему

ЖЕЛАНИЯ ДЕНИСА

Axelix: заглянуть внутрь Spring Boot

Рентген для работающего приложения:



Свойства — откуда пришло значение, что активно, что устарело



Бины — что создано, откуда взялось и кто куда внедрён



@Conditional — что вычислялось, что сработало и почему



Scheduled-задачи — расписание, история запусков, кнопка  Run

ЖЕЛАНИЯ ДЕНИСА

Axelix: заглянуть внутрь Spring Boot

Рентген для работающего приложения:



Свойства — откуда пришло значение, что активно, что устарело



Бины — что создано, откуда взялось и кто куда внедрён



@Conditional — что вычислялось, что сработало и почему



Scheduled-задачи — расписание, история запусков, кнопка  Run

Каждый вопрос из дня Дениса — **с ответом.**

ЖЕЛАНИЕ МАКСА

Статус каждой зависимости



Active — проект живой и развивается

ЖЕЛАНИЕ МАКСА

Статус каждой зависимости



Active — проект живой и развивается



Maintenance — только исправления: JUnit 4, Hystrix

ЖЕЛАНИЕ МАКСА

Статус каждой зависимости



Active — проект живой и развивается



Maintenance — только исправления: JUnit 4, Hystrix



Dormant — официально не закрыт, но релизов нет годами

ЖЕЛАНИЕ МАКСА

Статус каждой зависимости



Active — проект живой и развивается



Maintenance — только исправления: JUnit 4, Hystrix



Dormant — официально не закрыт, но релизов нет годами



Discontinued — закрыт: Spring Statemachine, Log4j 1

Axelix — это не CVE-сканер



Поиск CVE — задача Dependabot, Snyk, OWASP Dependency-Check. Axelix их **не заменяет**.

Axelix — это не CVE-сканер



Поиск CVE — задача Dependabot, Snyk, OWASP Dependency-Check. Axelix их **не заменяет**.



Axelix отвечает на другой вопрос: **какие зависимости доставят проблемы в будущем?**

Людям — UI. Агентам — MCR.

Раз мы работаем с агентами — Axelix отдаёт им те же знания.

ВСЁ, ЧТО ЗНАЕТ AXELIX

 сущности и транзакции

 бины и  условия

 scheduled-задачи

 свойства ·  RBAC

Людям — UI. Агентам — МСР.

Раз мы работаем с агентами — Axelix отдаёт им те же знания.

ВСЁ, ЧТО ЗНАЕТ AXELIX

- 🧩 сущности и транзакции
- 🔴 бины и ? условия
- 🕒 scheduled-задачи
- 🔑 свойства · 🔒 RBAC



Люди

UI — то, что вы сейчас увидите

Людям — UI. Агентам — MCP.

Раз мы работаем с агентами — Axelix отдаёт им те же знания.

ВСЁ, ЧТО ЗНАЕТ AXELIX

- 🧩 сущности и транзакции
- 🔴 бины и ? условия
- 🕒 scheduled-задачи
- 🔑 свойства · 🔒 RBAC



Люди

UI — то, что вы сейчас увидите



AI-агенты

MCP-сервер — те же данные в виде tools

Людям — UI. Агентам — MCP.

Раз мы работаем с агентами — Axelix отдаёт им те же знания.

ВСЁ, ЧТО ЗНАЕТ AXELIX

🧩 сущности и транзакции

🔴 бины и ? условия

🕒 scheduled-задачи

🔑 свойства · 🔒 RBAC



Люди

UI — то, что вы сейчас увидите



AI-агенты

MCP-сервер — те же данные в виде tools

Обе двери — под ролевой моделью доступа.

Демо

Повторим все три истории — на этот раз с **Axelix**.

сервис Ани → день Дениса → classpath Макса

Что унести с собой



Ваши проблемы **не уникальны** — они есть почти у всех.

Что унести с собой



Ваши проблемы **не уникальны** — они есть почти у всех.



AI-агенты их **умножают**: больше кода, больше инцидентов, больше риска в зависимостях.

Что унести с собой



Ваши проблемы **не уникальны** — они есть почти у всех.



AI-агенты их **умножают**: больше кода, больше инцидентов, больше риска в зависимостях.



Появилось решение от экспертов в экосистеме Spring для всего пути от кода до прода.

Что унести с собой



Ваши проблемы **не уникальны** — они есть почти у всех.



AI-агенты их **умножают**: больше кода, больше инцидентов, больше риска в зависимостях.



Появилось решение от экспертов в экосистеме Spring для всего пути от кода до прода.

Всё, что вы видели сегодня находится в **open source**



Спасибо! Вопросы? 🙌



Исходный код

github.com/axelixlabs/axelix



Telegram

t.me/axelix_official