Pawga777 1 апр 2024 в 14:40

Электронная подпись, шифрование данных с помощью RSA, AES. Реализация на Kotlin, Micronaut, bouncycastle

Средний 10 мин 3K

Криптография*, Программирование*, Kotlin*

Из песочницы

Введение

Разбирался со сложной темой: шифрование данных криптографическими способами, электронная подпись, гостовские алгоритмы шифрования и так далее. Решил поделиться опытом своих изысканий на примере создания сервиса работы с подписями и шифрованием данных. Покажу реализацию криптографическими средствами:

1. зашифровать файл / массив байт
2. расшифровать файл / массив байт
3. поставить «электронную подпись» на файл (обьясню, почему в кавычках) / массив байт
4. проверить «электронную подпись» файла / массив байт

Массив байт присутсвует в требованиях, потому что часто приходится информацию хранить не в виде файлов, а виде массива байтов в каком-нибудь хранилище, которое не файловая система(например специальное поле записи базы данных).

Шифрование

Сам термин шифрование описывать подробно не буду, так как он понятен. Само шифрование должно быть стойким. Это означает: расшифровка информации занимает огромное время, чтобы после расшифровки пропала ценность этой информации. Т.е теоретически расшифровать можно всё, вопрос стоит только в потраченном на этот процесс времени. В самой JAVA есть спецификации по этой теме: Java Cryptography Architecture (JCA) и Java Cryptography Extension (JCE) и удобные библиотеки, реализующие эти спецификации, например bouncycastle.

Существуют два основных алгоритма шифрования:

- симметричный алгоритм шифрования (AES)
- ассиметричный алгоритм шифрования (RSA)

В AES используется один ключ, в RSA два ключа(закрытый и публичный) и области применения этих алгоритмов разные.

Хэш-функция, сигнатура, электронная подпись

Тема электронных подписей очень обширная, так как здесь и связь с владельцем подписи. Здесь же появляется и тема сертификатов и много чего другого. Но глобально суть электронных подписей проста. Начнем с понятия хэш-функции.

Свойства хэш-функции:

- фиксированный результат длины хэша, независимо от размера источника данных;
- уникальный хэш, позволяющий определять изменение содержимого источника (при изменение даже одного бита в байте огромного документа хэш становится другим. На этом свойстве хэшей основываются все проверки неизменяемости документов);

Самые известные алгоритмы реализации хэш-функций: MD5, SHA, KeydHashAlgorithm и др. MD5 в настоящее время для серьезных задач практически не используется, так как уязвим и имеет самую высокую вероятность среди подобных алгоритмов по получению одинаковых хеш-сумм для разных объектов.

Реализация

Сервис будем реализовать на Kotlin и Micronaut. Далее в статье будут описания и мои личные субьективные предпочтения, которые совсем не претендуют на абсолютную правоту.

Почему Kotlin

Недавно пришлось исследовать реализацию ГОСТ-подписей. Все примеры были на JAVA. Код, который я потом переписал на Kotlin, был раза в полтора-два меньше по обьему кода на Java и, на мой взгляд, более «чистый». Еще раз напомню, что это моё личное предпочтение и прошу не начинать в комментариях «религиозных» дискусий на тему какой язык программирования лучше. Моё субьективное мнение — самый лучший язык програмирования — это C++ 🙄, но так как на нём очень немногие программисты умеют писать безопасные программы, были придуманы языки со сборщиками мусора за программистами 🙄 Прошу извинить за небольшое отклонение от темы.

Возвращемся к Kotlin. У Kotlin масса незаметных удобств, к тому же он null-безопасен как язык программирования из «коробки». Надеюсь этот раздел подвинет Java-разработчиков

«попробовать» Kotlin. В IntelliJ IDEA есть мастер, который великолепно умеет конвертировать Java в Kotlin.

Почему Micronaut

Столкнулся с огромной проблемой на Spring Boot при подготовке дистрибутивов, где сборка нативная (не байт-код, а уже целевой исполняемый код). Наверное, многие слышали что у нас, любителей разных JVM теперь есть такая возможность. Например, GraalVM нам в помощь.
В нативный вид приложения Spring Boot хоть и собираются, но частенько собираются чисто «символически», т. е. крашатся в разных местах во время своей работы. Попробуйте разные варианты сборки в нативный вид Spring Boot, работающие с БД на JPA. Спасибо глабольному использованию чудо техник рефлексии в SpringBoot. Кто еще не интересовался, полюбопытсвуйте как интерфейсы JPA без реализующих их классов волшебным образом имплементируются в классы и объекты в runtime SpringBoot.

Обнаружил, что у Micronaut нет этих проблем, потому что здесь архитектурно другой принцип. При необходимости можем получить и сборку в нативном коде, которая работает прямо из «коробки». Самые сложные конструкции работы с разными БД (PostgreSql, MySQL) тоже работают. При этом Micronaut почти близнец Spring Boot по конструкциям, всё интуитивно понятно. А такой структурированной документации с примерами и описаниями API я ранее не встречал нигде.

- Ссылки по теме

Основная часть

Для Micronaut есть соответсвующий инициализатор в трех видах: 1) командная строка 2) https://micronaut.io/launch 3) Плагины в средах разработки (я использую IntelliJ IDEA)

Ссылка на рабочий проект в конце статьи.

Пример учебный, местами реализация может быть спорна. Например, сервис дает возможность генерировать ассиметричные ключи и потом по соответсвующей «ручке» их скачивать.
В промышленных системах с приватными и секретными ключами так не работают.

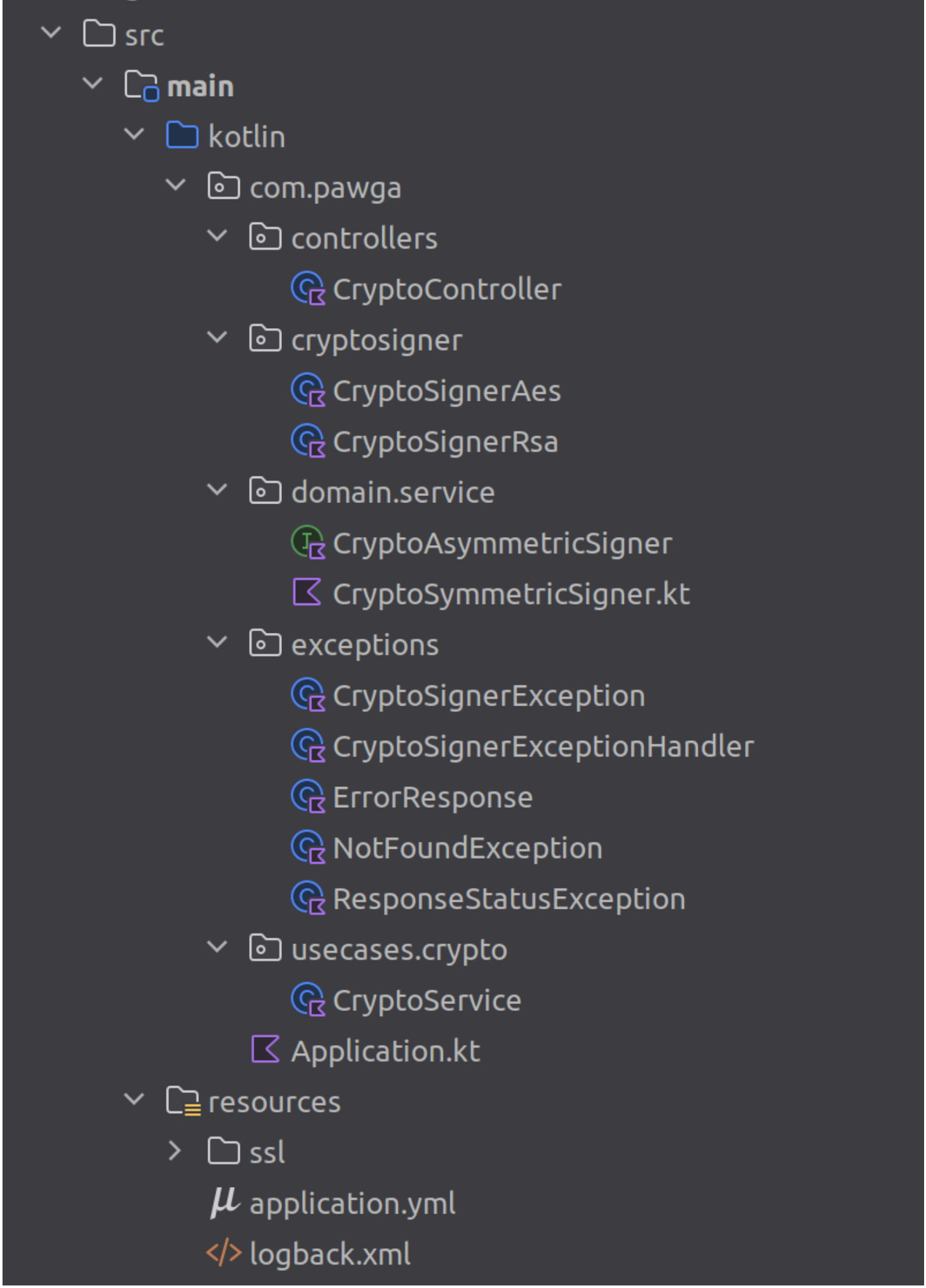
Основные зависимости проекта создаются автоматически инициализатором:

```
dependencies {
    ksp("io.micronaut.data:micronaut-data-processor")
    ksp("io.micronaut:micronaut-http-validation")
    ksp("io.micronaut.openapi:micronaut-openapi")
    ksp("io.micronaut.serde:micronaut-serde-processor")
    ksp("io.micronaut.validation:micronaut-validation-processor")
```

```
    implementation("io.micronaut.reactor:micronaut-reactor-http-client")
    implementation("io.micronaut.kotlin:micronaut-kotlin-runtime")
    implementation("io.micronaut.serde:micronaut-serde-jackson")
    implementation("io.micronaut.validation:micronaut-validation")
    implementation("jakarta.validation:jakarta.validation-api")
    implementation("org.jetbrains.kotlin:kotlin-reflect:${kotlinVersion}")
    implementation("org.jetbrains.kotlin:kotlin-stdlib-jdk8:${kotlinVersion}")
    implementation("org.bouncycastle:bcprov-jdk18on:1.77")
    implementation("org.bouncycastle:bcpkix-jdk18on:1.77")
    compileOnly("io.micronaut:micronaut-http-client")
    compileOnly("io.micronaut.openapi:micronaut-openapi-annotations")
    runtimeOnly("ch.qos.logback:logback-classic")
    runtimeOnly("com.fasterxml.jackson.module:jackson-module-kotlin")
    runtimeOnly("org.yaml:snakeyaml")
    testImplementation("io.micronaut:micronaut-http-client")
    testImplementation("org.mockito:mockito-core")
    testImplementation("org.assertj:assertj-core:3.25.3")
}
```

Обратите внимание, что для работы с криптографией используется библиотеки bouncycastle. Bouncycastle имеет реализацию и наших ГОСТ-овских криптоалгоритмов.

Проект выглядит так:



Скриншот проекта

application.yml:

```yaml
micronaut:
  ssl:
    enabled: true
  server:
    ssl:
      enabled: true
      key-store:
        path: classpath:ssl/keystore.p12
        password: XXXXXXXXXXXXX
        type: PKCS12
    max-request-size: '100MB'
    multipart:
      max-file-size: '100MB'
  application:
    name: crypto
  router:
    static-resources:
      swagger:
        paths: classpath:META-INF/swagger
        mapping: /swagger/**
      swagger-ui:
        paths: classpath:META-INF/swagger/views/swagger-ui
        mapping: /swagger-ui/**
```

Для ассимитричных алгоритмов интерфейс:

```
interface CryptoAsymmetricSigner {
    fun generateKeyPair()
    fun getPublicKey(): PublicKey
    fun getPrivateKey(): PrivateKey
    fun exportKeyPair(fileKeyPem: String, filePublicPem: String)
    fun exportKeyPair(fosKeyPem: FileOutputStream, fosPublicPem: FileOutputStream)
    fun exportPublicKey(fosPublicPem: FileOutputStream)
    fun exportPrivateKey(fosKeyPem: FileOutputStream)
    fun importKeyPair(fileKeyPem: File?, filePublicPem: File?)
    fun importKeyPair(private: ByteArray?, public: ByteArray?)
    fun importPrivateKey(fileKeyPem: File)
    fun importPrivateKey(private: ByteArray)
    fun importPublicKey(filePublicPem: File)
    fun importPublicKey(public: ByteArray)
    fun encrypt(fis: FileInputStream, fos: FileOutputStream)
```

```
    fun encrypt(data: ByteArray): ByteArray
    fun decrypt(fis: FileInputStream, fos: FileOutputStream)
    fun decrypt(data: ByteArray): ByteArray
    fun sign(fis: FileInputStream, fos: FileOutputStream)
    fun sign(data: ByteArray): ByteArray
    fun verify(fis: FileInputStream, sig: FileInputStream): Boolean
    fun verify(data: ByteArray, sig: ByteArray): Boolean
}
```

Специфицировано:

1. генерация ключевой пары
2. экспорт разных комбинаций ключевой пары и элементов
3. импорт разных комбинаций ключевой пары и элементов
4. шифрование файла, массива в памяти
5. дешифпования файла, массива в памяти
6. получение сигнатуры(подписи) на файл, массив в памяти
7. проверка сигнатуры(подписи) на файл, массив в памяти

> Обратите внимание на возможность «комбинаций». Нет необходимости иметь полную пару ключей. Для некоторый операций используется только публичный ключ (шифрование), только приватный ключ(расшифровка).

Для симетричных алгоритмов интерфейс:

```
interface CryptoSymmetricSigner {
    fun generateKey(size: Int, mode: ModeIv = ModeIv.SIMPLE): SecretKey
    fun getKey(): SecretKey?
    fun exportKey(fos: FileOutputStream)
    fun exportKeyToByteArray(): ByteArray
    fun importKey(fis: FileInputStream)
    fun importIvParameterSpec(bytes: ByteArray)
    fun import(keyFis: FileInputStream, ivFis: FileInputStream)
    fun import(keyBytes: ByteArray, ivBytes: ByteArray)
    fun exportIvParameterSpec(fos: FileOutputStream)
    fun exportIvParameterSpecToByteArray(): ByteArray
    fun importIvParameterSpec(fis: FileInputStream)
    fun importKey(bytes: ByteArray)
    fun encrypt(fis: FileInputStream, fos: FileOutputStream)
    fun encrypt(data: ByteArray): ByteArray
    fun decrypt(fis: FileInputStream, fos: FileOutputStream)
    fun decrypt(data: ByteArray): ByteArray
```

```
}
```

Специфицировано:

1. генерация секретного ключа
2. экспорт секретного ключа и IV
3. импорт секретного ключа и IV
4. шифрование файла, массива в памяти
5. дешифпования файла, массива в памяти

Функция шифрования:

```
override fun encrypt(data: ByteArray): ByteArray {
        if (publicKey == null) {
            throw CryptoSignerException("The publicKey is uninitialized!")
        }
        val encryptCipher = Cipher.getInstance(DEFAULT_ALGORITHM)
        encryptCipher.init(Cipher.ENCRYPT_MODE, publicKey)
        return encryptCipher.doFinal(data)
    }
```

Функция дешифрования:

```
override fun decrypt(fis: FileInputStream, fos: FileOutputStream) {
        if (privateKey == null) {
            throw CryptoSignerException("The privateKey is uninitialized!")
        }
        val data = fis.readAllBytes()
        val decData = decrypt(data)
        fos.write(decData)
    }
```

Функция расчета подписи/сигнатуры:

```
override fun sign(data: ByteArray): ByteArray {
        if (privateKey == null) {
            throw CryptoSignerException("The privateKey is uninitialized!")
        }
```

```
        val signature = Signature.getInstance(DEFAULT_SIGNATURE_ALGORITHM)
        signature.initSign(privateKey)
        signature.update(data)
        return signature.sign()
    }
```

Функция проверки подписи/сигнатуры:

```
override fun verify(data: ByteArray, sig: ByteArray): Boolean {
        if (publicKey == null) {
            throw CryptoSignerException("The publicKey is uninitialized!")
        }
        val signature = Signature.getInstance(DEFAULT_SIGNATURE_ALGORITHM)
        signature.initVerify(publicKey)
        signature.update(data)
        return signature.verify(sig)
    }
```

Реализация Controller-а

В реализации контроллера, на мой взгляд, тоже много чего интересного. Особенно с входными и выходными стримами(файлами). Например, прием конкретно двух файлов на обработку (не массива файлов, а именно конкретного количества). И чтобы при этом «swagger» эту ситуацию корректно визуализировал. Для SpringBoot и его «swagger» довольно проблемная ситуация.

```
@Post(
        value = "/import/key-pair",
        consumes = [MediaType.MULTIPART_FORM_DATA],
        produces = [MediaType.TEXT_PLAIN]
    )
    @Operation(
        summary = "Import Key Pair",
        description = "Attention! Dangerous operation! Shown for educational purposes!"
    )
    @Tag(name = "import")
    fun importKeyPair(
        @Part("public") public: CompletedFileUpload,
        @Part("private") private: CompletedFileUpload
    ): HttpResponse<Any> {
        cryptoService.importKeyPair(private.inputStream.readAllBytes(), public.inputStream.r
        return HttpResponse.ok()
    }
```

```
@Post(
        value = "/verify-sign",
        consumes = [MediaType.MULTIPART_FORM_DATA],
        produces = [MediaType.APPLICATION_JSON]
    )
    @Operation(
        summary = "Verify file signature",
    )
    @Tag(name = "sign")
    fun verify(
        @Part("data") data: CompletedFileUpload,
        @Part("sig") sig: CompletedFileUpload
    ): HttpResponse<Boolean> {
        val verify = cryptoService.verify(data.bytes, sig.bytes)
        log.debug("verify is $verify")
        return HttpResponse.ok(verify)
    }
```

Возврат результата в виде выходного стрима:

```
@Post(
        value = "/sign-file",
        consumes = [MediaType.MULTIPART_FORM_DATA],
        produces = [MediaType.APPLICATION_OCTET_STREAM]
    )
    @Operation(
        summary = "Calculate file signature",
    )
    @Tag(name = "sign")
    fun sign(data: CompletedFileUpload): StreamedFile {
        val inputStream = cryptoService.sign(data.bytes).inputStream()
        return StreamedFile(inputStream, MediaType.APPLICATION_OCTET_STREAM_TYPE)
    }
```

Работа сервиса

Запуск сервиса в обычном режиме:

```
  __  __ _                            _
 |  \/  (_) ___ _ __ ___  _ __   __ _| |_
 | |\/| | |/ __| '__/ _ \| '_ \ / _` | | | | __|
 | |  | | | (__| | | (_) | | | | (_| | |_| | |_
 |_|  |_|_|\___|_|  \___/|_| |_|\__,_|\__,_|\__|
00:09:50.816 [main] INFO  io.micronaut.runtime.Micronaut - Startup completed in 1390ms. Server Running: https://localhost:8443
```

Время старта приложения 1390 ms

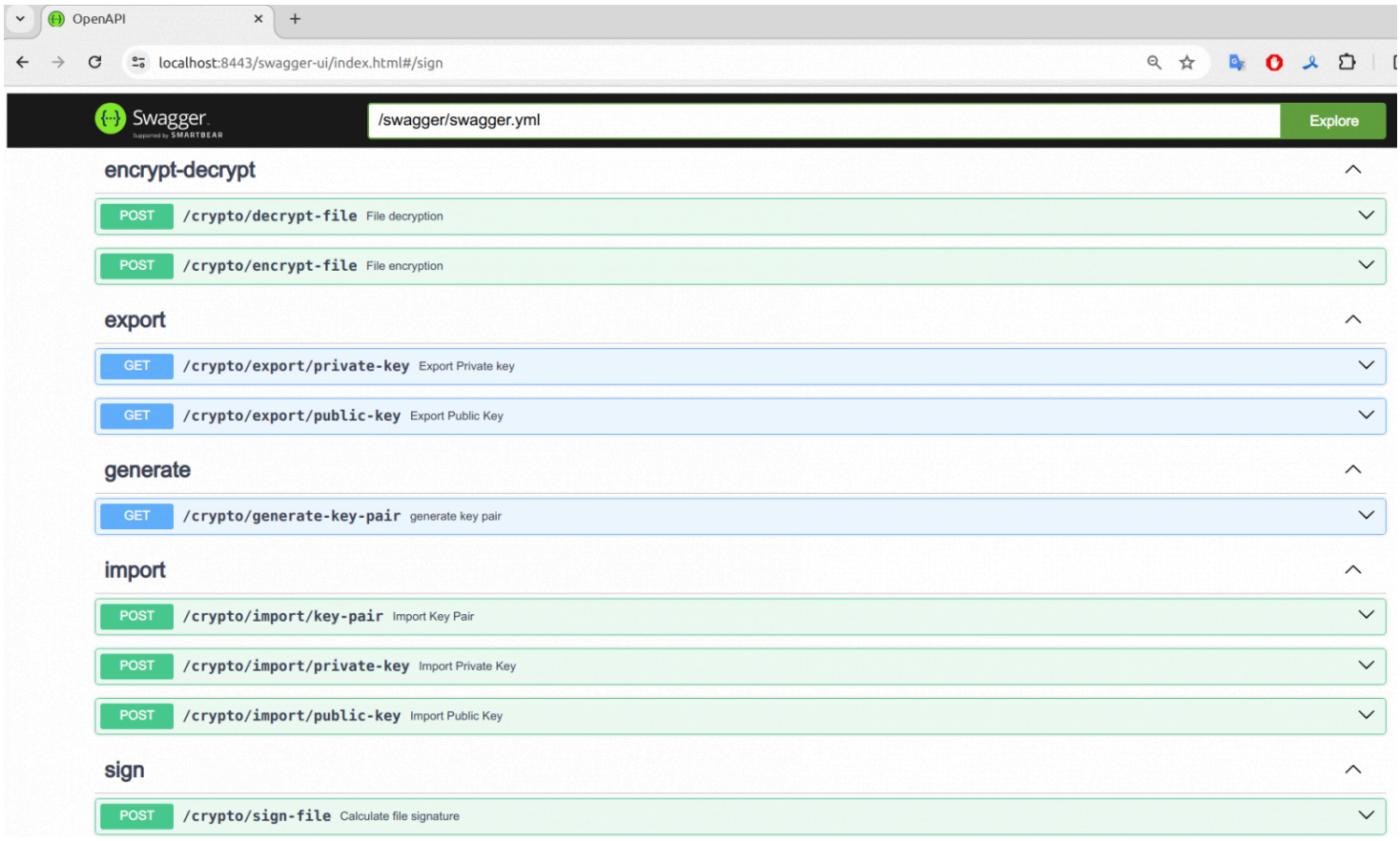
Я откомпилировал приложение в нативный вид и упаковал в докер-образ «pawga777/crypto:latest». Докер образ можете скачать из docker-hub.

Запустим приложение:

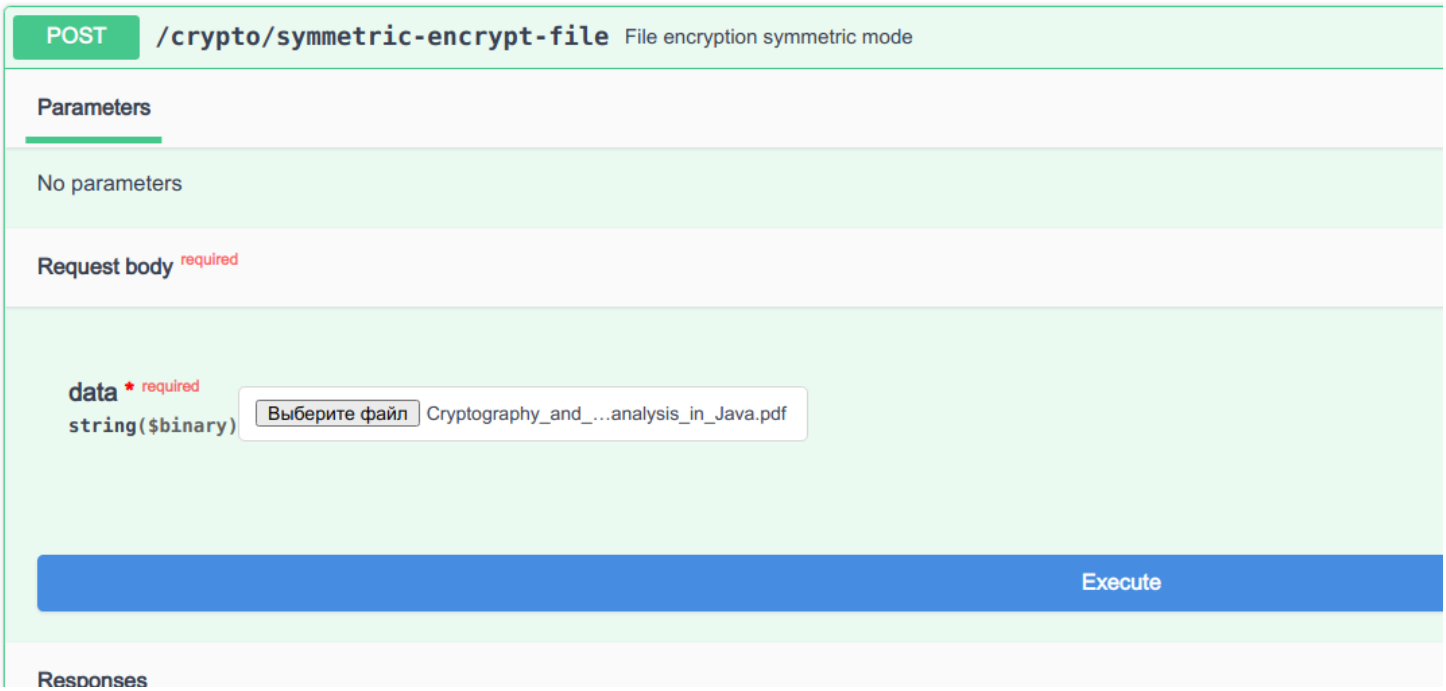
Время старта приложения 85 ms

Обратите внимание на разницу времени старта: 85 ms нативно откомплированной версии против 1390 ms JVM-версии Micronaut. Впечатляет? Аналогичная версия на JVM-версии SpringBoot стартует еще дольше (порядка 2000 ms), так как SpringBoot по своей «природе» медленнее Micronaut.

Давайте немного потестируем.



Зашифруем файл размеров 8 M



Шифруем файл чере web-версию swagger нашего приложения

Curl

```
curl -X 'POST' \
  'https://localhost:8443/crypto/symmetric-encrypt-file' \
  -H 'accept: application/octet-stream' \
  -H 'Content-Type: multipart/form-data' \
  -F 'data=@Cryptography_and_Cryptanalysis_in_Java.pdf;type=application/pdf'
```

Request URL

```
https://localhost:8443/crypto/symmetric-encrypt-file
```

Server response

Code	Details
200	Response body Download file Response headers

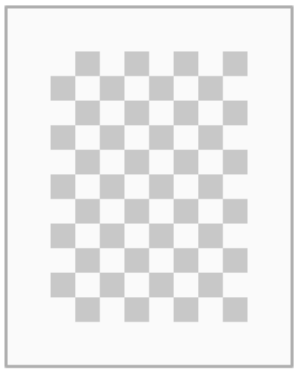
```
cache-control: private,max-age=60
content-type: application/octet-stream
date: Thu,28 Mar 2024 21:39:54 GMT
expires: Thu,28 Mar 2024 21:40:54 GMT
last-modified: Thu,28 Mar 2024 21:39:54 GMT
transfer-encoding: chunked
```

Responses

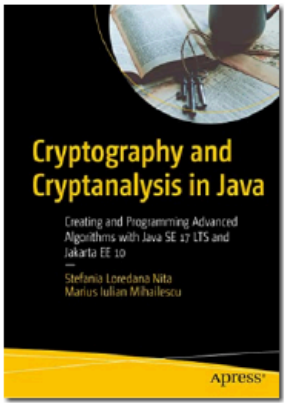
Code	Description
200	symmetricEncrypt 200 response

Аналогично, расшифруем:

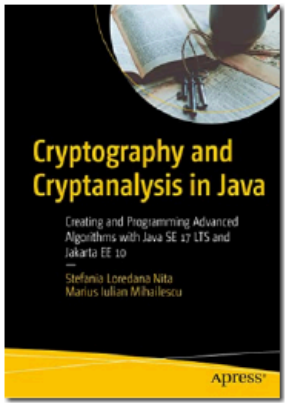
```
curl -X 'POST' \
  'https://localhost:8443/crypto/symmetric-decrypt-file' \
  -H 'accept: application/octet-stream' \
  -H 'Content-Type: multipart/form-data' \
  -F 'data=@Cryptography_and_Cryptanalysis_in_Java.enc'
```



Cryptography_and_
Cryptanalysis_in_Java.enc
8,7 MB
Пт 29 мар 2024 00:42:31



Cryptography_and_
Cryptanalysis_in_Java.pdf
8,7 MB
Сб 11 мар 2023 20:28:11



Cryptography_and_
Cryptanalysis_in_Java_enc.pdf
8,7 MB
Пт 29 мар 2024 00:43:17

Работа сервиса по шифрованию файла и его дешифровке

Cryptography_and_Cryptanalysis_in_Java.pdf — оригинал

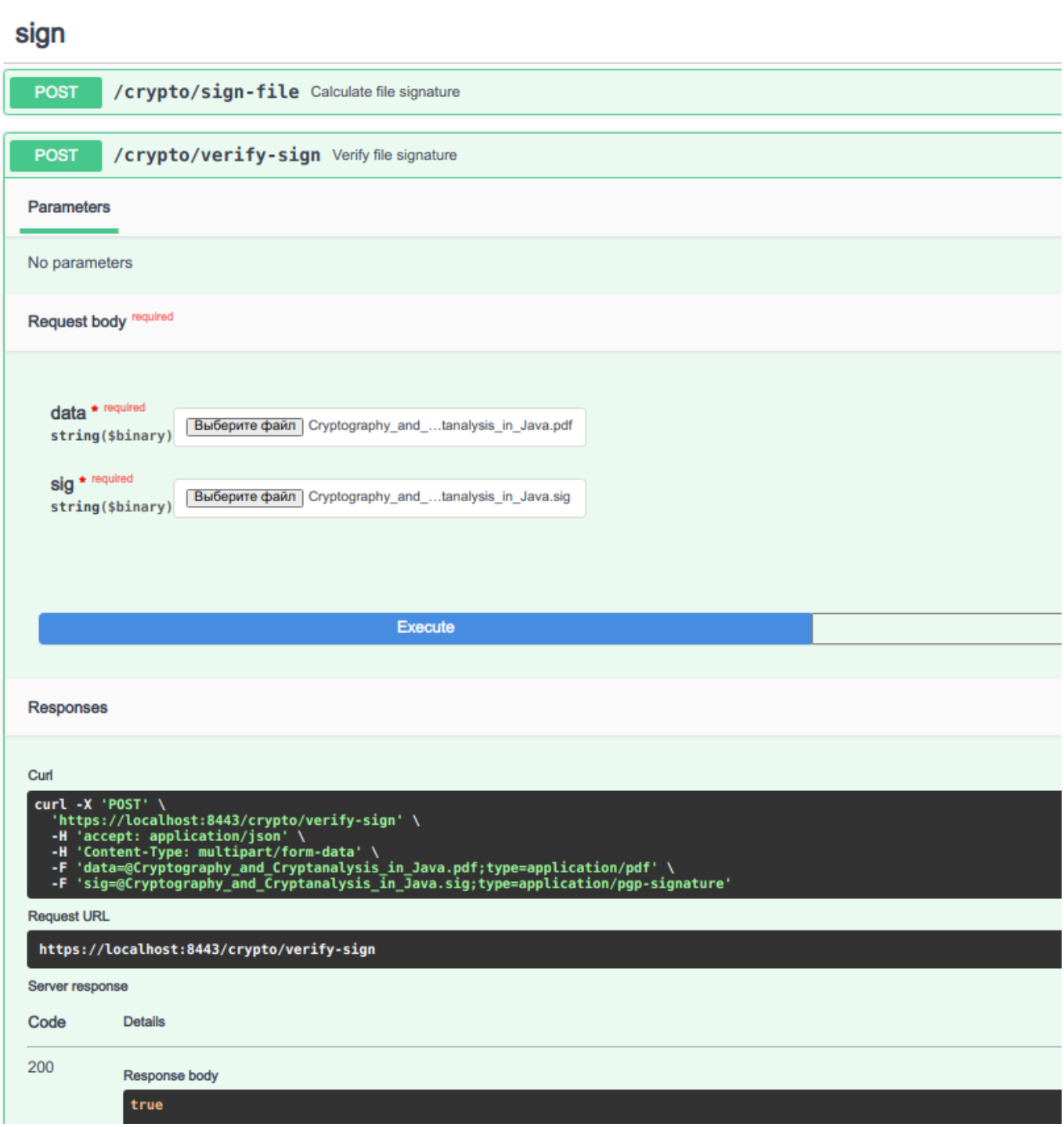
Cryptography_and_Cryptanalysis_in_Java.enc — зашифрованный файл

Cryptography_and_Cryptanalysis_in_Java_enc.pdf — дешифрованная версия

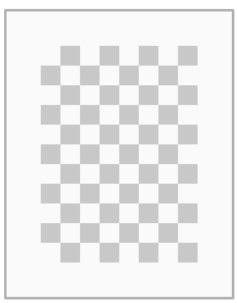
Теперь давайте получим сигнатуру(электронную подпись) файла.

```
curl -X 'POST' \
  'https://localhost:8443/crypto/sign-file' \
  -H 'accept: application/octet-stream' \
  -H 'Content-Type: multipart/form-data' \
  -F 'data=@Cryptography_and_Cryptanalysis_in_Java.pdf;type=application/pdf'
```

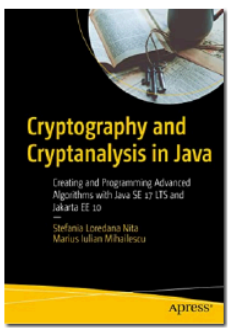
Проверим полученную электронную подпись:



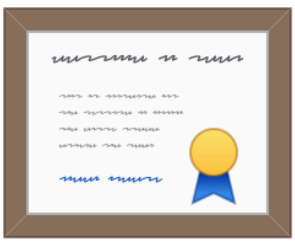
Проверка успешна. Размер подписи 256 байт



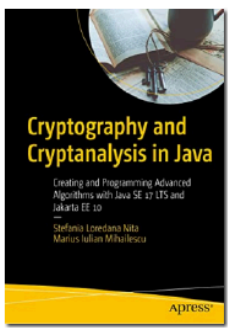
Cryptography_and_
Cryptanalysis_in_Java.enc
8,7 MB
Пт 29 мар 2024 00:42:31



Cryptography_and_
Cryptanalysis_in_Java.pdf
8,7 MB
Сб 11 мар 2023 20:28:11



Cryptography_and_
Cryptanalysis_in_Java.sig
256 байт
Пт 29 мар 2024 00:48:58



Cryptography_and_
Cryptanalysis_in_Java_enc.pdf
8,7 MB
Пт 29 мар 2024 00:43:17

Эпилог

Код приложения можно скачать отсюда: код приложения на github.

Кому любопытно, посмотрите тесты из bouncycastle для гостовских криптоалгоритмов, тесты перевел на Kotlin и добавил в раздел «тесты» приложения (к самому приложению эти тесты отношение пока не имеют).

Если у кого будет время, или есть уже такая информация, а именно как «отторгнуть» гостовские криптоалгоритмы от их контейнерной среды (КРИПТО ПРО...) — поделитесь. На эту тему нашел интересную реализацию: esia-crypto. Но повторить «подвиг» в виде «keystore.jks», в котором гостовские ключи хранятся в отдельном контейнере, мне не удалось.

Да, оставлю за скобками юридескую основу работы с гостовскими ключами, так как юридическая значимость работы с этими ключами появляется после удовлетворения ряда требований. У нас с вами просто исследовательский интерес по не очень тривиальной задаче.

По теме Micronaut планирую описать опыт работы с базами данных (PostgreSQL) в реактивном исполнении, в том числе и на корутинах. Все работает замечательно и в нативно откомпилированном виде, и в обычном. Но есть нюансы настроек такого «нативного» приложения для работы из среды докер-образов. Постараюсь рассказать.

Всем прочитавшим до конца мою статью — спасибо.

Happy Coding!

Теги: micronaut, rsa, aes

Хабы: Криптография, Программирование, Kotlin

2 Карма 0 Рейтинг

Сергей @Pawga777

Пользователь

Комментарии 5 +5

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ

Erwinmal 13 часов назад

Чапаев и Матрица: почему культура 90-х бунтовала против пластмассового мира? Часть 1

Простой 9 мин 3.7K

Ретроспектива

+37 23 16 +16

Lunathecat 9 часов назад

Электрогитара по доступной цене, не нуждающаяся в доработках

Простой 7 мин 4.3K

Обзор

+23 7 7 +7

DavidAsatryan 10 часов назад

Agile умер: из-за своего сострадания к product- и project-менеджерам (с) Фридрих Ницше

Простой · 8 мин · 7.4K

Мнение

+22 · 51 · 8 +8

erbanovanastasia 14 часов назад

Индия продолжает экспансию на рынок полупроводников: чего ожидать в ближайшем будущем

4 мин · 1.7K

+22 · 3 · 13 +13

tw0face 15 часов назад

Покажи свой стартап/пет-проект (Январь)

1 мин · 1.5K

+22 · 17 · 31 +31

ParfenovIgor 12 часов назад

Опыт написания компилятора вручную

Средний · 9 мин · 3.3K

+17 · 41 · 8 +8

Giox_Nostr 13 часов назад

Классика научной фантастики: хронология

Простой · 22 мин · 4.5K

Ретроспектива

+17 · 65 · 30 +30

Lexx_Nimofff 13 часов назад

Игра, вдохновлённая UFO и Jagged Alliance: интервью с главным разработчиков «Спарты 2035»

Простой 9 мин 1K

Интервью

+13 3 2 +2

guselnikov 4 часа назад

То о чем многие молчат, или может не знают...

3 мин 2.2K

+11 9 11 +11

bodyawm 8 часов назад

Я купил легендарный игровой смартфон из утиля и отремонтировал его — смотрим на Nokia N-Gage Classic

10 мин 3K

Ретроспектива

+10 3 10 +10

Сэкономили на логистике — выиграли пациенты. Как мы придумали онлайн-дозаказ анализов

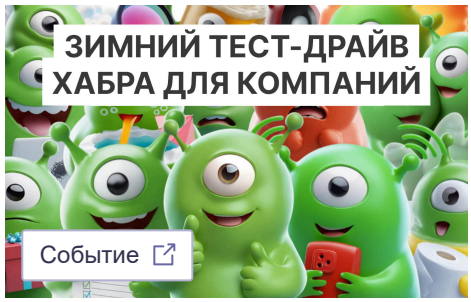
Турбо

Показать еще

МИНУТОЧКУ ВНИМАНИЯ

Как хабравчане следят за здоровьем?

Гиперконвергентная среда: OpenStack или VMware?

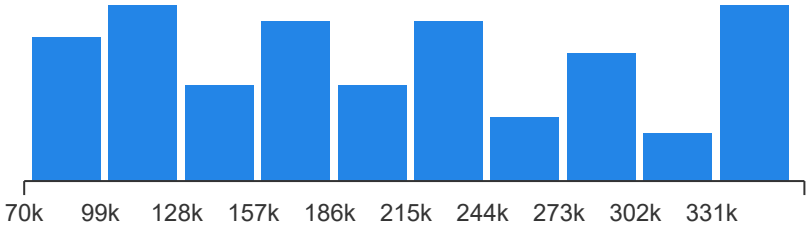


Зимний тест-драйв Хабра для компаний — 30 января в Москве

СРЕДНЯЯ ЗАРПЛАТА В IT

199 282 ₽/мес.

— средняя зарплата во всех IT-специализациях по данным из 12 396 анкет, за 1-ое пол. 2025 года. Проверьте «в рынке» ли ваша зарплата или нет!



Проверить свою зарплату

ЧИТАЮТ СЕЙЧАС

YouTube начал показывать пользователям с блокировщиками рекламы многочасовую рекламу, которую нельзя пропустить

27K 104 +104

Куда деваются отходы в самолетных туалетах

84K 127 +127

Преподавание английского — самый большой скам 21 века

45K 117 +117

Agile умер: из-за своего сострадания к product- и project-менеджерам (c) Фридрих Ницше

7.4K 8 +8

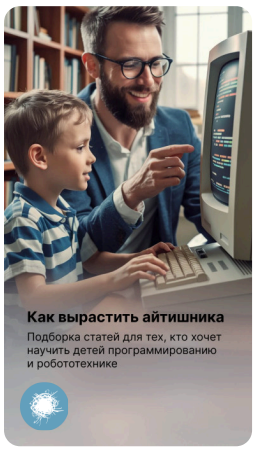
Веб-приложения будущего: что нужно знать о WebAssembly

9.3K 19 +19

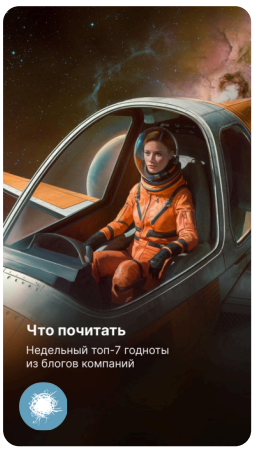
Сэкономили на логистике — выиграли пациенты. Как мы придумали онлайн-дозаказ анализов

Турбо

ИСТОРИИ



Как вырастить айтишника



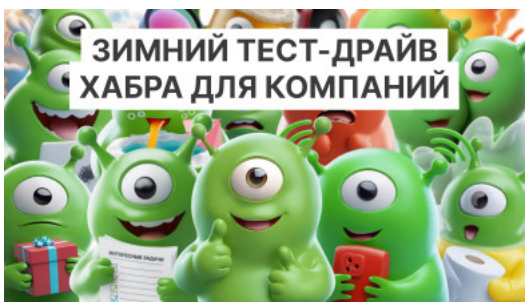
Годнота из блогов компаний

Нейрозима 2025

Статьи с новогодним вайбом

Кто выступит на конференции мечты

БЛИЖАЙШИЕ СОБЫТИЯ



30 января

Зимний тест-драйв Хабра для компаний

Москва

27 марта

Deckhouse Conf 2025

Москва

Разработка

25 – 26 апреля

IT-конференция M Tatarstan 2025

Казань

Маркетинг Другое Администрирование Разработка Маркетин

Менеджмент Другое

Больше событий в календаре