// JavaJub

Собес в МТС Банк

AQA Java

AQA Junior

Вопросы, задачи и подготовка
к live-coding и техническому интервью

Стек

Java · Rest Assured · JUnit 5 · PostgreSQL · Kafka · WireMock · Allure

// содержание

Содержание

01 Про МТС Банк и формат собеса

МТС Банк — цифровой банк экосистемы МТС. Более 7 млн клиентов, 300+ IT-специалистов. Фокус: мобильный банк, кредитование, платежи, интеграции с экосистемой МТС (связь, маркетплейс).

AQA Java (Junior) — инженер автоматизации тестирования. Главный фокус: автотесты для REST API микросервисов, тестирование интеграций через Kafka/RabbitMQ, мок внешних систем через WireMock, SQL-проверки данных. UI-автоматизация (Selenium) НЕ указана в вакансии.

▸ Структура собеседования (~1.5–2 часа)

Блок	Время	Что проверяют
Знакомство + опыт	~5–10 мин	Пет-проекты, мотивация
Теория тестирования	~15 мин	Виды, уровни, классы эквивалентности
Java Core (Junior)	~15–20 мин	ООП, коллекции, исключения, equals
SQL (Postgres + Oracle)	~15 мин	JOIN + GROUP BY + HAVING
HTTP / REST / Postman	~15 мин	Методы, статусы, JWT, цепочки
Rest Assured + WireMock	~15–20 мин	given/when/then, стаб, verify
Kafka / RabbitMQ (базово)	~5–10 мин	Зачем нужны, как тестировать
Security (специфика банка)	~5–10 мин	OWASP Top 10, SQL-инъекции, IDOR

ФИШКА · Специфика этой вакансии

Rest Assured + CitrusFramework + WireMock — это API + интеграционное тестирование микросервисов с моками внешних систем + тестирование Kafka/RabbitMQ потоков. Selenium/Selenide НЕ упомянуты — это НЕ UI-автоматизация.

ВНИМАНИЕ · Для Junior

В МТС Банке обязательна секция Security Testing — OWASP Top 10, SQL-инъекции, XSS, CSRF, IDOR. Также любят давать практические задачи в Postman с Pre-request Scripts — генерация JWT, цепочка запросов, подпись HMAC.

02 Стек по вакансии

Основной стек

- Java + ООП — основной язык автотестов
- JUnit 5 / TestNG — тест-фреймворки
- Rest Assured — HTTP API автотесты (given/when/then)
- CitrusFramework — интеграционное тестирование Kafka/RabbitMQ
- WireMock — моки внешних сервисов
- Maven — сборка проекта
- Jackson — сериализация/десериализация JSON
- PostgreSQL + Oracle — SQL для проверки данных
- Kafka + RabbitMQ — брокеры сообщений
- Postman + Swagger/OpenAPI — ручное API-тестирование
- Docker — контейнеризация тестовой среды
- Git — контроль версий
- Linux + bash — работа с логами, серверами
- ELK — поиск по логам (Elasticsearch + Kibana)
- Allure — отчёты по тестам

03 Теория тестирования

Для Junior AQA — это ОБЯЗАТЕЛЬНЫЙ блок. Без теории тестирования не возьмут даже при идеальном Java.

- Верификация vs Валидация.

  → *Верификация: «Строим ли мы продукт правильно?» — соответствие требованиям (ревью, инспекции, статический анализ). Валидация: «Строим ли мы правильный продукт?» — соответствие ожиданиям пользователя (тестирование на реальных данных). Классика — путают местами.*

- QA vs QC.

  → *QA (Quality Assurance): процесс обеспечения качества — стандарты, процессы, предотвращение дефектов. QC (Quality Control): контроль качества — тестирование, нахождение дефектов. QA — шире, QC — часть QA.*

- Уровни тестирования.

  → *Unit → Integration → System → Acceptance. Unit: отдельный метод/класс (JUnit, Mockito). Integration: взаимодействие компонентов (Testcontainers, WireMock). System: вся система целиком. Acceptance: приёмка пользователем (UAT).*

- Smoke / Sanity / Regression / Confirmation — отличия.

  → *Smoke: «дымовой тест» — базовая работоспособность после деплоя (5–10 кейсов). Sanity: проверка конкретного фикса/фичи (узко). Regression: проверка, что старое не сломалось (широко, автоматизируем). Confirmation: повторная проверка исправленного бага.*

- Black Box / White Box / Grey Box.

  → *Black Box: не знаем внутреннюю реализацию, тестируем по спецификации. White Box: знаем код, тестируем ветвления, покрытие. Grey Box: знаем архитектуру, но тестируем как пользователь (AQA — это Grey Box: пишем тесты зная API-контракт и БД-схему).*

- Принципы тестирования.

  → *Полное тестирование невозможно. Кластеризация дефектов (80% багов в 20% модулей). Парадокс пестицида (одни тесты перестают находить баги — обновляй). Раннее тестирование дешевле. Отсутствие ошибок — заблуждение. Тестирование зависит от контекста.*

- Позитивное vs негативное тестирование.

  → *Позитивное: проверяем штатное поведение (валидные данные → ожидаемый результат). Негативное: проверяем реакцию на невалидные данные (пустое поле, отрицательная сумма, SQL-инъекция → ожидаем ошибку, а не падение). AQA покрывает оба типа: сначала позитивные (happy path), потом негативные + граничные.*

- End-to-End тест — что это?

  → *Тест полного пользовательского сценария от начала до конца. Пример: регистрация → логин → создание заявки → одобрение → выдача кредита → погашение. Проходит через все слои: UI/API → бэкенд → БД → Kafka → внешние сервисы. Дороже в поддержке, запускается реже.*

- Жизненный цикл бага.

  → *New → Open → Assigned → In Progress → Fixed → Resolved → Verified → Closed (или Reopen). Severity: Blocker (блокирует работу), Critical (потеря денег), Major (основной сценарий с workaround), Minor (косметика), Trivial (опечатка). Priority: бизнес-важность исправления.*

- Что делать, если разработчик говорит «не баг, а фича»?

  → *1) Проверить требования — ссылка на спецификацию. 2) Если требований нет — эскалировать на PO/аналитика. 3) Оформить баг с чётким ОР/ФР и приложить доказательства. 4) Обсудить на daily/grooming. 5) Если действительно фича — закрыть баг как Won't Fix, но зафиксировать решение.*

- Scrum церемонии: Planning, Daily, Review, Retro.

  → *Planning: что берём в спринт, декомпозиция. Daily: 15 мин, три вопроса (что сделал, что буду, что блокирует). Review: демо результатов стейкхолдерам. Retrospective: что было хорошо, что улучшить, action items. QA участвует во всех — на Planning оценивает тестируемость, на Review демонстрирует автотесты.*

04 Тест-дизайн

- Классы эквивалентности — как применять?

  → *Разбиваем входные данные на группы, в которых поведение системы одинаковое. Пример: поле «возраст» 18–60 → три класса: <18 (отказ), 18–60 (принимаем), >60 (отказ). Из каждого класса берём по одному значению — это и есть минимальный набор тестов.*

- Граничные значения — почему важны?

  → *Баги чаще всего на границах: 0, 1, max, max+1. Пример: пароль 8–20 символов → тестируем: 7 (отказ), 8 (ок), 20 (ок), 21 (отказ). Для Junior: уметь определять границы для любого поля.*

- Тест-кейс vs чек-лист.

  → *Тест-кейс: подробный (шаги, ожидаемый результат, предусловия) — для критичных сценариев, передачи другим. Чек-лист: краткий список проверок без деталей — для опытных, быстрой проверки. В автоматизации: тест-кейс → метод с @Test.*

- State Transition — как использовать?

  → *Для систем с состояниями: заявка на кредит (Новая → На рассмотрении → Одобрена/Отклонена → Выдана). Рисуем диаграмму состояний и переходов. Тестируем: все валидные переходы + невалидные (из «Выдана» нельзя в «Новая»). Пример для банкомата: Idle → Card Inserted → PIN Entered → Menu → ...*

- Severity vs Priority.

  → *Severity: техническая серьёзность (Critical/Major/Minor/Trivial). Priority: бизнес-важность исправления (High/Medium/Low). Пример: Critical + Low = баг критичный, но в третьестепенной фиче (которой пользуются 0.1% клиентов). Minor + High = косметический баг на главной странице.*

- Decision Table — когда применять?

  → *Когда поведение зависит от комбинации условий. Пример: кредит одобряется если (доход > 50к) И (кредитная история хорошая) И (возраст 21–65). Таблица: каждая комбинация условий → действие (одобрить / отклонить / запросить доп. документы). Покрывает все логические ветки.*

- Pairwise testing — зачем?

  → *Когда параметров много (5 полей по 4 значения = 1024 комбинации). Pairwise: покрыть все ПАРЫ параметров (не все комбинации). Инструменты: PICT (Microsoft), AllPairs. Сокращает количество тестов с 1024 до ~20, покрывая 80% багов.*

- Где брать тестовые данные в банке?

  → *Продовые данные нельзя — персональные данные (152-ФЗ). Варианты: 1) Синтетические генераторы (JavaFaker, DataFactory). 2) Маскированный прод (замена ФИО, телефонов, номеров карт). 3) Фикстуры в тестах (@BeforeEach). Для AQA: фикстуры + синтетика. В МТС Банке обязательно маскирование PII.*

- Структура тест-кейса.

  → *ID, заголовок (что тестируем), предусловие (авторизован, баланс > 0), шаги (1. POST /payment..., 2. GET /status...), ожидаемый результат (статус 200, баланс уменьшился), постусловие (откатить данные). В автоматизации: @Test метод = тест-кейс, @BeforeEach = предусловие, assertThat = ожидаемый результат.*

05 Java Core (Junior)

- Четыре принципа ООП своими словами.

  → *Инкапсуляция: банкомат — не знаешь как хранит деньги, есть методы «снять», «положить». Наследование: электросамокат наследует «транспорт». Полиморфизм: кнопка «play» — разное поведение для видео, музыки, подкаста. Абстракция: руль в машине — не думаешь о рейке и шестерёнках.*

- == vs equals для строк. Почему String immutable?

  → *== сравнивает ссылки (один объект в памяти?). equals — содержимое. String immutable: безопасность (ключи HashMap, передача в файлы/БД), потокобезопасность, кэширование hashCode, String Pool.*

- final, finally, finalize — три разных слова.

  → *final: класс (нельзя наследовать), метод (нельзя переопределить), поле (нельзя переприсвоить). finally: блок try-catch-finally, выполняется ВСЕГДА. finalize(): метод Object, вызывается GC перед удалением. Deprecated с Java 9. Для ресурсов — try-with-resources.*

- Коллекции: ArrayList vs LinkedList vs HashMap.

  → *ArrayList: динамический массив, O(1) доступ, O(n) вставка в середину. LinkedList: двусвязный список, O(1) вставка/удаление, O(n) доступ. HashMap: бакеты + связный список → дерево при ≥8 коллизиях. Для Junior: знать когда что использовать.*

- Stream API: что такое, промежуточные vs терминальные.

  → *Stream — конвейер обработки данных. Промежуточные (ленивые): filter, map, flatMap, sorted. Терминальные (запускают): collect, forEach, count, reduce. Стрим одноразовый.*

- Checked vs Unchecked исключения.

  → *Checked: от Exception (не RuntimeException) — компилятор требует обработки. IOException, SQLException. Unchecked: от RuntimeException — не обязательно ловить. NullPointerException, IllegalArgumentException. try-with-resources: ресурс реализует AutoCloseable.*

- Overload vs Override — в чём разница?

  → *Overload (перегрузка): несколько методов с одним именем, но разными параметрами. Компилятор выбирает (статическое связывание). Override (переопределение): подкласс меняет реализацию метода родителя. JVM выбирает в рантайме (динамическое связывание, полиморфизм). Override: нельзя сузить видимость, нельзя расширить throws.*

- Абстрактный класс vs интерфейс.

  → *Абстрактный класс: состояние (поля), конструкторы, один наследник. Интерфейс: контракт, множественная реализация, Java 8 — default/static методы. Когда что: общее поведение + состояние → абстрактный класс. Контракт без реализации → интерфейс. Можно реализовать несколько интерфейсов, но наследовать один класс.*

- Generics — зачем нужны?

  → *Типобезопасность на этапе компиляции. Без generics: List list = new ArrayList(); list.add(123); String s = (String)list.get(0) → ClassCastException в рантайме. С generics: List<String> — компилятор не даст положить Integer. Type erasure: в рантайме параметр стирается (List<String> → List).*

- Integer cache: почему valueOf(100)==valueOf(100) true, а valueOf(200) false?

  → *IntegerCache кэширует значения от -128 до 127. valueOf(100) возвращает один и тот же объект из кэша → == сравнивает ссылки → true. valueOf(200) создаёт новый объект каждый раз → разные ссылки → false. Мораль: для объектов всегда equals(), никогда ==.*

- Records в Java — что это?

  → *Java 14+: компактный immutable data-класс. record User(String name, int age) {} — автоматически: конструктор, getters (name(), age()), equals, hashCode, toString. Нельзя наследовать (implicitly final). Можно реализовывать интерфейсы. Идеально для DTO в тестах.*

- Optional — основные методы и антипаттерны.

  → *isPresent/isEmpty — проверка. ifPresent(consumer) — действие если есть. orElse(default) — значение по умолчанию (eager). orElseGet(supplier) — ленивый. map/flatMap — трансформация. Антипаттерн: optional.isPresent() + optional.get() → заменить на map().orElseGet(). НЕ использовать как поле класса или параметр метода.*

- Модификаторы доступа — порядок.

  → *private → (default/package-private) → protected → public. private: только в классе. default: в пакете. protected: в пакете + наследники (даже в другом пакете). public: везде. Для AQA: тестируемые методы обычно public или package-private (тесты в том же пакете).*

06 HTTP, REST API и Postman

REST API — главный блок для AQA. HTTP-методы, статусы, идемпотентность, JWT — спрашивают на каждом собесе.

- HTTP-методы и идемпотентность.

  → *GET (получить, идемпотентен), POST (создать, НЕ идемпотентен), PUT (заменить полностью, идемпотентен), PATCH (частично обновить, обычно нет), DELETE (удалить, идемпотентен). Идемпотентность: повторный запрос = тот же результат. POST не идемпотентен → может создать дубль.*

- 401 vs 403 — в чём разница?

  → *401 Unauthorized: не аутентифицирован (нет токена или токен невалидный). 403 Forbidden: аутентифицирован, но нет прав (пытаешься удалить чужой заказ). Ловушка: 401 — про identity, 403 — про permissions.*

- JWT — структура и как работает?

  → *Три части через точку: header.payload.signature. Header: алгоритм (HS256). Payload: данные (userId, exp, roles). Signature: HMAC(header+payload, secret). Хранится на клиенте (localStorage/cookie). Проверяется сервером без БД (stateless). exp — время жизни, после которого нужен refresh token.*

- REST vs SOAP.

  → *REST: HTTP, JSON, легковесный, stateless, гибкий. SOAP: XML, WSDL, строгий контракт, WS-Security. В банках: REST для новых API, SOAP для legacy (АБС, процессинг). Junior AQA: REST — основной, но упомянуть знание SOAP — плюс.*

- Цепочка запросов в Postman.

  → *1) POST /auth/login → получить accessToken. 2) В Tests: pm.environment.set("token", jsonData.accessToken). 3) GET /users/me с заголовком Authorization: Bearer {{token}}. 4) Проверка: pm.expect(pm.response.code).to.eql(200). Pre-request Script: генерация timestamp, подпись HMAC-SHA256.*

- Где передавать данные: body / path / query / headers?

  → *Path: идентификатор ресурса (/users/123). Query: фильтрация, пагинация (?page=2&sort=name). Body: данные для создания/обновления (POST/PUT/PATCH). Headers: метаданные (Authorization, Content-Type, Accept). НЕ в URL: пароли, токены, персональные данные.*

- CORS — что это и когда возникает?

  → *Cross-Origin Resource Sharing: браузер блокирует запросы к другому домену. Сервер разрешает через заголовки: Access-Control-Allow-Origin, Access-Control-Allow-Methods. Preflight запрос (OPTIONS) перед POST/PUT. Для AQA: CORS не мешает Rest Assured (нет браузера), но может мешать Postman в браузерной версии.*

- PUT vs PATCH — когда что?

  → *PUT: замена ресурса целиком. Отправляешь ВСЕ поля, даже неизменённые. Идемпотентен. PATCH: частичное обновление. Отправляешь ТОЛЬКО изменённые поля. Обычно НЕ идемпотентен. Для AQA: тестировать оба — PUT с неполным body → ожидать ошибку или обнуление полей.*

- OpenAPI / Swagger — для чего?

  → *Спецификация REST API в формате YAML/JSON. Описывает endpoints, методы, параметры, тела запросов/ответов, коды ответов. Swagger UI — интерактивная документация (можно отправлять запросы). API-First: сначала пишем спецификацию, потом код. Для AQA: из OpenAPI можно автогенерировать тесты и модели.*

- Pagination: offset/limit vs cursor-based.

  → *offset/limit: GET /users?offset=20&limit=10. Простой, но медленный на больших данных (БД пропускает offset строк). cursor-based: GET /users?after=abc123&limit=10. Быстрый (индекс по*

// JavaJub

Это только начало гайда.

МТС Банк · AQA · AQA Junior

Полная версия — бесплатно в Telegram-канале:

- все вопросы с ответами по грейду
- задачи: live-coding, SQL, System Design — с разбором
- чек-лист «за 2 часа до собеса»

@java_jub

t.me/java_jub · подписка бесплатная