

Тестовое задание на роль Java Developer

***В первую очередь мы ожидаем от кандидата любовь к играм в любых их проявлениях. Если так, то это тестовое задание доставит вам удовольствие.***

Общая информация

Представьте, что компания, в которой вы работаете, начинает разработку новой игры. Вашей задачей является проектирование и разработка Rest-сервиса (включая выбор технологии и продумывание структуры хранилища данных), который позволял бы выполнять синхронизацию пользовательских данных, а также принимать от пользователя и накапливать игровую статистику по каждому игроку.

Синхронизация

Передаваемая от пользователя информация для синхронизации - это строковый JSON о структуре которого ничего неизвестно за исключением следующего: в корне документа должно присутствовать поле money", в котором будет содержаться целое числовое значение и поле "country" с буквенным идентификатором страны пользователя (предполагается, что страна для конкретного пользователя не изменяется). Также известно, что передаваемый от пользователя JSON не должен превышать 10 кб. Прогнозируемое количество пользователей приложения - 100 000 000 человек, из которых активными ежедневно будут около 1 000 000 пользователей. Распределение активности пользователей по времени дня будет равномерным. Предполагается, что каждый день в среднем активный пользователь будет 100 раз отправлять на сервер информацию о синхронизации и 1 раз запрашивать информацию о своем текущем состоянии с сервера.

Игровая статистика

Пользовательская активность в игре за период измеряется на клиенте с помощью некоторого целочисленного показателя "activity". Каждый активный пользователь в день будет присылать 10000 раз значение этого показателя на сервер, эти данные необходимо принимать и накапливать.

Требования от отдела аналитики

При разработке решения нужно учитывать требования, которые были поставлены перед вами аналитическим отделом, им требуется (Х,Y - параметры запросов):

● быстро выбирать Х пользователей с самым большим текущим значением "money" по каждой стране "country".

● иметь возможность быстро подсчитать количество новых пользователей по каждой стране за период Х.

● для каждого конкретного пользователя X быстро получить отсортированный по дате список значений показателя "activity" и даты их получения за период Y.

Формальная постановка задачи

реализовать рест сервис с тремя эндпоинтами:

* прием данных синхронизации от пользователя

вход: строка - данные пользователя в формате Json, строка - UUID пользователя

выход: подтверждение получения | 100 раз в день на пользователя

* отправка данных пользователю

вход: строка - UUID пользователя

выход: строка сохраненные данные пользователя в формате Json | 1 раз в день на пользователя

* прием игровой статистики от пользователя

вход: числовое значение - activity пользователя, строка - UUID пользователя

выход: подтверждение получения | 10000 раз в день на пользователя

выбрать технологии хранения данных (их может быть несколько), спроектировать и реализовать структуру хранения, соответствующую решаемым задачам

реализовать три запроса к системам хранения данных (на соответствующем им языке), которые позволят выбирать информацию в соответствии с требованиями аналитического отдела

написать unit тесты, соответствующие вашему понимаю "хороших тестов", позволяющие автоматически верифицировать логику хотя бы одного рест эндпоинта.

Требования к используемым технологиям

● необходимо использовать J8, Spring (Boot, Data, MVC, etc), Maven

● технологии хранения данных определяются вами в соответствии с задачей, но предпочтение следует отдавать популярным вариантам http://db-engines.com/en/ranking учитывая указанный сценарий использования

● при выборе технических решений следует исходить из того, что выделение ресурсов для горизонтального масштабирования инфраструктуры не является проблемой для компании.

Ожидаемый результат

1. архив, содержащий мавен проект со всеми зависимостями

2. краткий файл с описанием шагов для развертывания и проверки вашего решения.

3. будет плюсом: использование Docker сompose для автоматизированного развертывания окружения со всеми внешними системам.

Свои вопросы и готовое решение присылайте на почту hr@carx-tech.com

Удачи!

### 1. Эндпоинт для приема данных синхронизации от пользователя

**URL:** /api/sync

**Метод:** POST

**Тело запроса:**

json

{

  "uuid": "1e6",

  "money": 1000,

  "country": "RU",

  "createdAt": "2023-12-04T21:24:00"

}

**Пример ответа:**

json

{

"status": "success",

"message": "Data received"

}

### 2. Эндпоинт для отправки данных пользователю

**URL:** /api/sync/{uuid}

**Метод:** GET

**Пример ответа:**

json

{

    "id": 1,

    "uuid": "1e6",

    "money": 1000,

    "country": "RU",

    "createdAt": "2023-12-04T21:24:00",

    "userActivityDataList": [

        {

            "id": 1,

            "activity": 777,

            "timestamp": "2023-10-10T15:30:00"

        }

    ]

}

### 3. Эндпоинт для приема игровой статистики от пользователя

**URL:** /api/stats

**Метод:** POST

**Тело запроса:**

json

{

  "activity": 777,

  "timestamp": "2023-10-10T15:30:00",

  "userSyncData": {

    "uuid": "1e6"

  }

}

**Пример ответа:**

json

{

"message": "Activity recorded"

}

Замечания:

1. *Запросы для аналитики должны быть в программе, а не текстом в инструкции*
2. *В инструкции пишет, что все эндпоинты протестированы нагрузочными тестами, этих тестов нет в репозитории*
3. *Не использовал библиотеки для устранения бойлерплейта (Lombook) дефакто уже стандарт для Java, при этом не переопределил важные equals и hashCode методы*
4. *Хотелось бы посмотреть использование второй нереляционной базы данных*
5. *Не использовал скрипты миграции*
6. *Не использовал профили Spring*
7. *Вынести данные пропертис в окружение. Использовать @RequiredArgsConstructor*
8. *Нет Junit тестов сервисов.*
9. *Интеграционные тесты мокированы, хотелось бы посмотреть тесты всего контекста Spring приложения*
10. *Излишний код в Docker файле при сборке проекта, можно сделать проще*
11. *Не использует интерфейсы при написании сервисов*
12. *Не умеет использовать Optional*
13. *Дублируется код*
14. *Игнорирует замечания IDE (могут быть и отключены)*
15. *Не использует JavaDoc комментарии*
16. *DTO*