CAPTCHA-сервис

Необходимо разработать HTTP-сервис, который может быть интегрирован с сайтом или любым другим web-приложением, обеспечивающий простой механизм выявления ботов.

В сервисе должна быть реализована поддержка 5 запросов:

1. POST-запрос, регистрирующий клиента CAPTCHA-сервиса. В ответ на запрос приходит JSON-объект с двумя UUID-строками: **secret** – приватный ключ клиента, **public** – публичный ключ, используемый на сайте.
   ```
   POST /client/register HTTP/1.1
   ```
   Пример JSON-ответа:
   ```
   { "secret": "7810ebec-b343-42ef-84da-a8990c6a9125",
     "public": "93bbf5c8-f223-4d70-83a9-a23a7683c2e9" }
   ```
2. GET-запрос, инициирующий процесс проверки пользователя. Параметры строки запроса: **public** – публичный ключ клиента, выданный при регистрации. В теле ответа приходит JSON-объект с двумя строками: **request** – идентификатор теста CAPTCHA, **answer** – слово-разгадка к решению теста.
   ```
   GET /captcha/new HTTP/1.1
   ```
   Пример JSON-ответа:
   ```
   { "request": "aaaaaaaa", "answer": "42" }
   ```
3. GET-запрос, предоставляющий CAPTCHA-картинку. Параметры строки запроса: **public** – публичный ключ клиента, **request** – идентификатор теста CAPTCHA. В теле ответа передаётся CAPTCHA-картинка в формате PNG.
   ```
   GET /captcha/image HTTP/1.1
   ```
4. POST-запрос с ответом на CAPTCHA. Параметры строки запроса: **public** – публичный ключ клиента, **request** – идентификатор теста CAPTCHA и **answer** – ответ на него. В случае успешного прохождения теста в теле ответа возвращается строка, содержащая токен **response**, используемый для последующей верификации, в противном случае должен быть использован соответствующий HTTP-код.
   ```
   POST /captcha/solve HTTP/1.1
   ```
   Пример JSON-ответа:
   ```
   { "response": "bbbbbbbb" }
   ```
5. GET-запрос проверки предоставляемого пользователем токена. Параметры строки запроса: **secret** – приватный ключ клиента, **response** – токен, полученный пользователем после прохождения теста CAPTCHA. В теле ответа приходит JSON-объект со следующими полями: **success** – результат верификации токена (булевый тип), **errorCode** – строка с информацией об ошибке, если таковая имеется.
   ```
   GET /captcha/verify HTTP/1.1
   ```
   Пример JSON-ответа:
   ```
   { "success": true, "errorCode": null }
   ```

Требования к CAPTCHA:

1. На один тест CAPTCHA верно можно ответить только 1 раз (последующие попытки отправить запрос с верным ответом возвращают ошибку).
2. Ответить на тест CAPTCHA можно только в течение ограниченного промежутка времени (запрос с верным ответом по истечении времени вернёт ошибку). Время жизни теста должно передаваться в виртуальную машину параметром **ttl**, измеряющемся в секундах.
3. Предоставляемый пользователю токен **response** одноразовый.
4. В CAPTCHA-картинке шифруется слово, состоящее из символов латинского алфавита и цифр.
5. Алгоритм генерации CAPTCHA-картинок предлагается разработать самостоятельно. Разрешено использование алгоритмов преобразования изображений, если это будет необходимо.
6. Шифруемое слово должно генерироваться случайным образом.

Требования к сервису:

1. Сервис должен запускаться в двух режимах: тестовом (слово-разгадка передаётся в теле ответа на запрос #2) и боевом (слово-разгадка не передаётся). Выбор режима осуществляется передачей соответствующего параметра **production** в виртуальную машину (см. использование *System.getProperty("production")*).
2. При создании сервиса можно воспользоваться любым из указанных фреймворков: Spring Framework, Vert.x. При желании можно написать всё на чистой Java.
3. Запускаться сервис должен как standalone-приложение (как-то так: *java –jar service.jar*). В случае выбора Spring Framework, можно воспользоваться Spring Boot.
4. Необходимо написать тесты на сервис.
5. Приложение должно собираться с использованием **Apache Maven**.
6. Необходимо использовать подходящие HTTP-коды в зависимости от смысла ответа.

Прочее:

1. Язык программирования: Java 8.
2. Оформление кода должно соответствовать общепринятым нормам (например, https://google.github.io/styleguide/javaguide.html).
3. Исходники необходимо упаковать в ZIP-архив вместе с кратким описанием решения и инструкцией по сборке и запуску.
4. Проверка решения будет осуществляться в автоматизированном режиме (функциональные и нагрузочные тесты), поэтому важно соблюдение указанных в задании требований.
5. Всё, что явно не указано в условиях, остаётся на усмотрение автора решения.