دانشکده مهندسی کامپیوتر

برنامه نویسی موبایل

تمرین دوم

مدرس:

مهندس پویا یارندی

۱۴ خرداد ۱۴۰۴

فهرست مطالب

۱ ۱. ارزیابی کلی و نقاط قوت: ۱

۲ ۲. محدودیت‌ها و کاستی‌ها: ۲

۳ ۳. پیشنهادها برای بازطراحی و بهبود API: ۳

API بازی Mastermind که از طریق مستندات Swagger ارائه شده است، یک رویکرد حداقلی و در عین حال کاربردی برای پیاده‌سازی بازی ارائه می‌دهد. این تحلیل به بررسی عمیق‌تر این API و ارائه پیشنهادهایی برای بهبود آن می‌پردازد.

۱ ۱ . ارزیابی کلی و نقاط قوت:

API فعلی دارای چندین نقطه قوت است که آن را برای شروع مناسب می‌سازد:

- **سادگی و کاربردی بودن (Minimalistic and Functional):** API بر روی عملکردهای اصلی بازی (ایجاد بازی، حدس زدن، حذف بازی) تمرکز کرده و از پیچیدگی‌های غیرضروری پرهیز می‌کند. این ویژگی باعث می‌شود که توسعه‌دهندگان کلاینت بتوانند به سرعت با API آشنا شده و یک کلاینت پایه را پیاده‌سازی کنند.

- **استفاده از متدهای استاندارد HTTP:** API از متدهای استاندارد HTTP مانند POST برای ایجاد منابع (بازی جدید) یا انجام عملیات (ارسال حدس) و DELETE برای حذف منابع (حذف بازی) استفاده می‌کند. این امر با اصول RESTful و رویه‌های رایج وب هماهنگ است و API را برای توسعه‌دهندگان قابل پیش‌بینی می‌کند.

- **تبادل اطلاعات با فرمت JSON:** استفاده از JSON به عنوان فرمت تبادل اطلاعات، یک انتخاب هوشمندانه است. JSON سبک، خوانا برای انسان و به طور گسترده توسط زبان‌ها و پلتفرم‌های مختلف پشتیبانی می‌شود که فرآیند توسعه کلاینت را تسهیل می‌بخشد.

- **اسکیمای واضح برای درخواست‌ها و پاسخ‌ها (Clear Schemas):** مستندات Swagger به روشنی ساختار مورد انتظار برای درخواست‌ها و پاسخ‌ها را تعریف می‌کند. به عنوان مثال، main.CreateGameResponse به وضوح مشخص می‌کند که یک game_id بازگردانده خواهد شد، main.GuessRequest نیازمندی به game_id و guess (رشته حدس) را تصریح می‌کند و main.GuessResponse جزئیات بازخورد مهره‌های

سیاه (black) و سفید (white) را ارائه می‌دهد. این شفافیت، ابهام را کاهش داده و به ساخت پیام‌های با فرمت صحیح کمک شایانی می‌کند.

- **مدیریت خطای پایه (Basic Error Handling):** وجود یک اسکیمای تعریف‌شده برای خطا (main.ErrorResponse که شامل یک رشته "error" است) نشان‌دهنده تلاش برای گزارش‌دهی استاندارد خطاها است، حتی اگر این مکانیزم پایه باشد. این به کلاینت یک روش سازگار برای دریافت پیام‌های خطا ارائه می‌دهد.

۲ ۲. محدودیت‌ها و کاستی‌ها:

علی‌رغم نقاط قوت، API فعلی دارای محدودیت‌های قابل توجهی است:

- **عدم وجود اندپوینتی برای بازیابی وضعیت فعلی یا تاریخچه بازی:** این یک کاستی مهم است. اگر برنامه کلاینت دچار مشکل شود، اتصال اینترنت کاربر قطع گردد یا کاربر بخواهد دستگاه خود را تغییر دهد، هیچ راهی برای ادامه بازی فعلی وجود ندارد. game_id در چنین سناریوهایی عملاً بی‌فایده می‌شود. این محدودیت همچنین مانع از پیاده‌سازی ویژگی‌هایی مانند حالت تماشاچی (spectator mode) یا داشبورد مدیریتی برای نظارت بر بازی‌ها می‌شود.

- **عدم ارائه پارامترهای کلیدی بازی توسط API:** پارامترهای حیاتی بازی مانند طول کد مخفی (مثلاً ۴ رقم)، محدوده ارقام مجاز (مثلاً اعداد ۱ تا ۶) و حداکثر تعداد تلاش‌های مجاز، توسط API در اختیار کلاینت قرار نمی‌گیرند (این موارد تنها در توضیحات تمرین ذکر شده‌اند، نه در مشخصات API). کلاینت مجبور است این قوانین را به صورت hardcode شده در خود داشته باشد یا از منبعی خارج از API دریافت کند. اگر سرور بازی تصمیم بگیرد این قوانین را تغییر دهد (مثلاً کد ۵ رقمی یا ارقام ۱ تا ۸)، تمام برنامه‌های کلاینت موجود از کار خواهند افتاد مگر اینکه به صورت دستی به‌روزرسانی شوند. این موضوع سیستم را شکننده و تکامل آن را دشوار می‌سازد. API باید تنها منبع حقیقت (single source of truth) برای قوانین بازی باشد.

- **پیغام‌های خطای عمومی و ناکافی:** هرچند main.ErrorResponse یک ساختار برای خطا ارائه می‌دهد، اما یک رشته "error" تکی اغلب برای مدیریت خطای قوی در سمت کلاینت کافی نیست. به عنوان مثال، اگر یک حدس نامعتبر باشد، آیا به این دلیل است که game_id وجود ندارد (خطای 404)، فرمت حدس اشتباه است (مثلاً تعداد ارقام زیاد یا کم، کاراکترهای نامعتبر) یا بازی قبلاً تمام شده است؟ کدهای خطای مشخص یا پیام‌های خطای ساختاریافته‌تر به کلاینت اجازه می‌دهند تا بازخورد دقیق‌تری به کاربر نمایش دهد یا منطق تلاش مجدد (retry logic) خاصی را پیاده‌سازی کند. برای مثال، پاسخ

{"error_code": "INVALID_GUESS_FORMAT", "message": "The guess must be a 4-digit number between 1 and 6."}

بسیار مفیدتر از یک پاسخ عمومی مانند {"error": "Bad Request"} است.

- **ابهام در مورد فرمت دقیق ورودی حدس:** API فیلد guess را به عنوان یک رشته دریافت می‌کند. در حالی که توضیحات تمرین به «۴ رقم، هر رقم بین ۱ تا ۶» اشاره دارد، خود API در اسکیمای خود جزئیاتی فراتر از "string" ارائه نمی‌دهد. اگر کاربر "123"، "abcde" یا "12345" ارسال کند، رفتار API چگونه خواهد بود؟ API باید به وضوح فرمت‌های معتبر حدس را تعریف کرده و برای انحراف از این فرمت‌ها، خطاهای مشخصی بازگرداند. پاسخ 400 Bad Request فعلی برای اندپوینت guess/ بسیار کلی است.

۳ ۳. پیشنهادها برای بازطراحی و بهبود :API

اگر قرار بر بازطراحی این API باشد، چندین بهبود کلیدی را می‌توان در نظر گرفت:

- **اضافه کردن اندپوینت GET /game/{gameID}:** این اندپوینت به کلاینت‌ها اجازه می‌دهد تا وضعیت کامل یک بازی خاص را بازیابی کنند. پاسخ این اندپوینت می‌تواند یک شیء JSON شامل موارد زیر باشد:
  - تعداد تلاش‌های انجام شده تاکنون.

- تاریخچه حدس‌ها (مثلاً آرایه‌ای از اشیاء که هر کدام شامل حدس و بازخورد مهره‌های سیاه و سفید مربوطه هستند).
- وضعیت فعلی بازی (مثلاً "in_progress"، "won"، "lost").
- احتمالاً پیکربندی بازی (طول کد، محدوده ارقام برای آن بازی خاص) اگر در جای دیگری ارائه نشده باشد.

- **ارائه جزئیات پیکربندی بازی (Game Configuration):**

- **گزینه ۱ : اندپوینت GET /config:** یک اندپوینت اختصاصی مانند GET /config می‌تواند تنظیمات کلی بازی مانند طول کد پیش‌فرض، محدوده‌های ارقام موجود و حداکثر تلاش‌های پیش‌فرض را بازگرداند.
- **گزینه ۲: در پاسخ POST /game:** هنگامی که یک بازی جدید از طریق POST /game ایجاد می‌شود، پاسخ می‌تواند شامل پیکربندی خاص *آن* نمونه از بازی باشد. این روش زمانی مفید است که بازی‌ها بتوانند تنظیمات متفاوتی داشته باشند. به عنوان مثال: {"game_id": "xyz", "code_length": 4, "digit_range": [1, 6], "max_attempts": 10}.

این کار به کلاینت‌ها اجازه می‌دهد پویاتر باشند و کمتر به مقادیر hardcode شده وابسته باشند.

- **ارائه پاسخ‌های خطای دقیق‌تر و ساختاریافته (Granular Error Responses):** به جای پاسخ ساده {"error": "some message"}، از یک شیء خطای ساختاریافته‌تر استفاده شود. این شیء می‌تواند شامل موارد زیر باشد:

- یک کد خطای منحصر به فرد (عددی یا رشته‌ای، مثلاً 1001 یا ERR_GAME_NOT_FOUND).
- یک پیام مناسب برای توسعه‌دهنده (developer-friendly).
- به صورت اختیاری، یک پیام محلی‌سازی شده مناسب برای کاربر (user-friendly) (اگرچه محلی‌سازی اغلب دغدغه کلاینت است).

- به صورت اختیاری، یک فیلد details برای ارائه جزئیات بیشتر و زمینه spécifiques خطا.

مثال: برای یک درخواست نامناسب (کد وضعیت 400)، بدنه پاسخ می‌تواند به این شکل باشد:

.{"error_code": "INVALID_GUESS_LENGTH", "message": "The guess must be exactly 4 digits."}

- **استفاده از نسخه‌بندی صریح API در URL (API Versioning):** با تکامل API گاهی اوقات ایجاد تغییراتی که با نسخه‌های قبلی سازگار نیستند (breaking changes) اجتناب‌ناپذیر است. نسخه‌بندی (مثلاً در URL مانند /api/v1/game یا از طریق هدرهای HTTP مانند Accept: application/vnd.myapi.v1+json) به کلاینت‌های موجود اجازه می‌دهد تا به استفاده از نسخه قدیمی و پایدار API ادامه دهند، در حالی که کلاینت‌های جدید می‌توانند از نسخه جدیدتر استفاده کنند. این کار از اختلال در عملکرد سرویس‌های موجود هنگام انتشار به‌روزرسانی‌ها جلوگیری می‌کند. نسخه‌بندی مبتنی بر URL اغلب سرراست‌ترین روش است.

- **بهبود اندپوینت ایجاد بازی:**

  - **کد پاسخ 201 Created:** هرچند استفاده از کد پاسخ 200 OK برای POST /game قابل قبول است، اما در اصول RESTful، ایجاد منابع اغلب با کد 201 Created همراه با یک هدر Location که به منبع تازه ایجاد شده اشاره می‌کند (مثلاً Location: /game/{new_game_id}) پاسخ داده می‌شود. این یک قرارداد رایج و استاندارد است.
  - **ویژگی Idempotency (اختیاری):** بررسی شود که آیا نیاز به ویژگی Idempotency برای ایجاد بازی وجود دارد یا خیر. اگر کلاینت به دلیل مشکل شبکه، درخواست POST /game را دو بار ارسال کند، آیا دو بازی جداگانه ایجاد می‌شود؟ برای برخی عملیات، Idempotency حیاتی است. برای ایجاد بازی، شاید ایجاد دو بازی قابل قبول باشد، اما این باید یک تصمیم آگاهانه در طراحی باشد.

- **ملاحظات عملیاتی (فراتر از طراحی صرف API):**

- **امنیت و محدودیت نرخ درخواست‌ها (Rate Limiting):** یک API در محیط پروداکشن باید ملاحظات امنیتی مانند محدودیت نرخ درخواست‌ها را در نظر بگیرد تا از سوءاستفاده (مثلاً تلاش یک کلاینت برای حدس زدن کد با سرعت بسیار بالا از طریق brute-force) جلوگیری شود. همچنین اگر بازی‌ها به کاربران خاصی مرتبط بودند، مکانیزم‌های احراز هویت نیز ضروری می‌بودند.
- **استفاده از HATEOAS (اختیاری و پیشرفته):** برای یک طراحی RESTful پیشرفته‌تر، پاسخ‌ها می‌توانند شامل لینک‌هایی به عملیات بعدی ممکن باشند. به عنوان مثال، پس از POST /game، پاسخ می‌تواند لینکی برای ارسال حدس برای آن بازی را شامل شود. این امر API را قابل کشف‌تر کرده و به کلاینت‌ها کمک می‌کند تا جریان برنامه را به صورت پویاتری هدایت کنند (این ممکن است برای این بازی ساده بیش از حد نیاز باشد، اما به عنوان یک اصل خوب در طراحی API مطرح می‌شود).

با اعمال این تغییرات و بهبودها، API بازی Mastermind می‌تواند به یک سرویس قوی‌تر، انعطاف‌پذیرتر و با تجربه کاربری بهتر برای توسعه‌دهندگان تبدیل شود.