ShahnamehMap — سند ۴,۴: کیفیت کدبیس و مهندسی

نسخه: ۱,۰

تاریخ: ۱۰/۰۸/۱۴۰۳

تهیه‌کننده: مدیر فنی (CTO)

وضعیت: فعال

---

۱. **ساختار پروژه و معماری کدبیس**

ما از **آرایش مونورپو (Monorepo)** با استفاده از **Turborepo** استفاده می‌کنیم. این تصمیم به مدیریت مستقل اما هماهنگ سرویس‌های فرانت‌اند و بک‌اند کمک می‌کند.

```
shahnamehmap/
├── apps/
```

```
|   ├── web/                  # فرانت‌اند اصلی (Next.js 14 - App Router)
|   |   ├── app/              # صفحات و layoutها
|   |   ├── components/       # کامپوننت‌های قابل استفاده مجدد
|   |   ├── lib/              # توابع کمکی، کلاینت API
|   |   ├── styles/           # فایل‌های Tailwind/SCSS گلوبال
|   |   └── public/           # asset های استاتیک
|   ├── api-gateway/          # API Gateway سفارشی (Node.js + Express)
|   └── game-engine/          # سرویس موتور بازی (Node.js + WebSocket)
├── packages/                 # پکیج‌های اشتراکی داخلی
|   ├── shared-types/         # انواع TypeScript مشترک (فرانت/بک)
|   ├── shared-config/        # تنظیمات ESLint، Prettier، TSConfig
|   ├── database/             # Prisma Client و مدل‌های دیتابیس
|   └── event-schema/         # Schema رویدادها (با Zod)
```

```
├── infra/                    # Infrastructure as Code (Terraform)
توسعه لوکال #      ├── docker-compose.yml
Turborepo تنظیمات #      ├── turbo.json
├── package.json          # Workspace root
راهنمای جامع شروع به کار #      └── README.md
```

بین **(Types) دلیل انتخاب مونورپو:** تسهیل **اشتراک کد و انواع** فرانت و بک، اجرای **یکپارچه تست و لینت** در کل پروژه، و از یک ریپو Docker **ساده‌سازی استقرار** با ساخت تصاویر.

۲.** استانداردهای کد و ابزارها**

۲,۱.** زبان‌ها و فریمورک‌ها**:

* **فرانت‌اند:** Next.js 14 (App Router)، React 18، TypeScript 5.x.
* **بک‌اند:** Node.js 20 LTS، Express، TypeScript 5.x، Socket.io.
* **دیتابیس اورم (ORM):** **Prisma** برای PostgreSQL + **Raw Queries** رویکرد دست‌نوشت برای کوئری‌های پیچیده گراف Neo4j**.
* **استایل‌ینگ:** **CSS Modules** + **Tailwind CSS** برای کامپوننت‌های پیچیده.

۲,۲. کانونشن‌ها (Conventions):

* **نام‌گذاری:**
* کامپوننت‌های React: `PascalCase` (مانند `CharacterSheet.tsx`).
* متغیرها و توابع: `camelCase`.
* فایل‌های غیرکامپوننت: `kebab-case` (مانند `api-handler.ts`).
* ثابت‌ها (Constants): `UPPER_SNAKE_CASE`.

:ساختار فایل *

* شامل `CharacterBuilder/` :هر کامپوننت اصلی در پوشه خودش
`index.tsx`، `styles.module.css`، `utils.ts`، `types.ts`.

* برای صادرات (`index.ts`) **Barrel Exports** استفاده از
.تمیز از پکیج‌های مشترک

۲,۳. ابزارهای کیفیت کد:

* **لینتر:** **ESLint** (حتی `"error"` با پیکربندی strict)
.(قواعد مهم

* **فرمت‌کننده:** **Prettier** همه برای یکسان قوانین.

* **کامیت هوک‌ها:** با **lint-staged** + **Husky**، قبل از
.شده اجرا می‌شود stage هر کامیت، لینت و تست روی فایل‌های

* **چک تایپ:** `tsc --noEmit` در خط لوله CI اجرا
.می‌شود

۲,۴. مستندسازی کد:

* **JSDoc** برای توابع پیچیده عمومی و APIها.**

* **README** در هر سرویس/پکیج که هدف و نحوه اجرای لوکال را توضیح دهد.

* استفاده از **Confluence** برای مستندات معماری سطح داخلی بالا (مثل این سند) و تصمیمات فنی (ADRها).

---

۳. استراتژی تست (Testing Strategy)

ما از **هرم تست (Test Pyramid)** پیروی می‌کنیم: تست‌های زیاد و ارزان در پایه، تست‌های کمتر و گران در راس.

نوع تست	ابزار	پوشش هدف	اجرا در CI
Unit Tests	**Vitest** (Jest سریع‌تر از)	توابع منطق کسب‌وکار، utilities هوک‌های React ، اجباری	✓

| **Component Tests** | **Vitest** + **React Testing Library** | کامپوننت‌های UI ایزوله (بدون وابستگی به API). | ✅ |

| **Integration Tests** | **Vitest** + **Supertest** | تعامل بین چند ماژول (مثلاً API endpoint و سرویس دیتابیس). | ✅ (برای مسیرهای حیاتی) | اجباری |

| **E2E Tests** | **Playwright** | جریان‌های کاربری حیاتی (ثبت‌نام، ساخت کاراکتر، شروع بازی). | ✅ (Nightly) در شبانه staging |

| **Manual / Exploratory** | - | تست UX، تست بازی‌سازی. | تیم محصول/QA توسط |

* **معیار پوشش کد (Code Coverage):** هدف حداقل ۸۰٪ برای توابع منطق کسب‌وکار (Business Logic) و حداقل ۶۰٪ برای کل پروژه. این معیار توسط CI چک می‌شود و در گزارش Vitest پوشش کلی اندازه‌گیری می‌شود.

* **محیط تست:** برای راه‌اندازی سرویس‌های وابسته (PostgreSQL، Redis) از Docker Compose استفاده می‌شود.

۴.** خط لوله CI/CD و مدیریت نسخه**

۴,۱.** خط لوله CI/CD (با GitLab CI):**

```yaml
# مراحل اصلی
stages:
  - lint
  - test
  - build
  - deploy-staging
  - deploy-production

# قوانین:
# 1. همه feature branch ها باید از lint و test عبور کنند.
# 2. merge به `develop` باعث deploy خودکار به محیط staging
می‌شود.
```

.می‌شود production به deploy باعث `main` روی tag ایجاد .3 #

```

**:مراحل کلیدی**

1. **Lint & Type Check:** اجرای ESLint، Prettier و `tsc`.
2. **Unit & Integration Tests:** اجرای Vitest با پوشش کد.
3. **Build:** ساخت تصاویر Docker برای سرویس‌های مربوطه.
4. **Deploy to Staging:** استقرار خودکار روی محیط staging (با استفاده از Kubernetes Helm Charts).
5. **Manual Approval & Deploy to Prod:** نیاز به تأیید دستی توسط **CTO** یا **تیم لید** قبل از استقرار در production.

### **۴,۲. مدیریت نسخه (Versioning):**

* **برنچینگ (Branching):** **GitFlow** سبک‌شده (Simplified GitFlow)**.
  * `main`: همیشه قابل استقرار، متناظر با production.
  * `develop`: شاخه یکپارچه‌سازی، متناظر با staging.
```

.برای توسعه ویژگی‌های جدید :`feature/*` *

.production برای رفع باگ‌های فوری روی :`hotfix/*` *

(Versioning):** **Semantic Versioning** ورژنینگ** *
انتشار نسخه‌ها با ایجاد .(`MAJOR.MINOR.PATCH`) **(SemVer)**
.انجام می‌شود `main` روی **Git Tag**

۴,۳. وابستگی‌ها (Dependencies):

* از **Dependabot** (در GitHub) یا **Renovate** برای
.استفاده می‌شود PR به‌روزرسانی خودکار وابستگی‌ها با

* همه به‌روزرسانی‌ها **باید از تست‌ها عبور کنند**. به‌روزرسانی‌های ماژور
.نیاز به بررسی دستی دارند (`major`)

---

۵. مدیریت Technical Debt

ما Technical Debt را به عنوان **یک تصمیم آگاهانه** (و نه یک اتفاق
.ناخواسته) مدیریت می‌کنیم

۵,۱. **ردیابی و اولویت‌بندی:**

* **ابزار:** استفاده از **Jira** در **Issues/ Tasks** با برچسب `tech-debt`.
* **شناسایی (Code Review):** در جلسات **بازبینی کد** و **ریترو اسپرینت (Sprint Retrospective)** شناسایی و ثبت می‌شود.
* **اولویت‌بندی (Impact/Cost Matrix):** بر اساس ماتریس **تأثیر/هزینه** بدهی‌هایی که مانع توسعه جدید یا ریسک امنیتی دارند، اولویت بالا می‌گیرند.

۵,۲. **تخصیص زمان:**

* **قانون ۲۰٪:** در هر اسپرینت توسعه، **حداکثر ۲۰٪** از زمان های با اولویت Technical Debt برنامه‌ریزی‌شده تیم توسعه به پرداختن به بالا اختصاص می‌یابد.
* **فصل‌های پاکسازی (Cleanup Sprints):** هر ۶-۴ اسپرینت، یک **اسپرینت کامل پاکسازی** (بدون فیچر جدید) اختصاص داده می‌شود تا روی بدهی‌های انباشته شده تمرکز کنیم.

۵٫۳. نمونه‌های Technical Debt فعلی و برنامه:**

مورد	**تأثیر**	**برنامه پرداخت**
API های قدیمی endpoint ها در برخی DTO عدم استفاده از	افزایش احتمال خطا در فرانت، مستندسازی ضعیف.	Refactoring در اسپرینت آینده با استفاده از Zod برای validation.
کامپوننت `MapExplorer` با منطق بیش از حد پیچیده (God Component)	شکستن به کامپوننت‌های کوچکتر (`MapView`, `LocationPanel`, `SearchBar`) کاهش قابلیت تست و نگهداری.	در اسپرینت پاکسازی بعدی.
ریسک باگ در جریان پرداخت	نداشتن تست E2E برای Playwright	پیاده‌سازی ۲ تست E2E کلیدی با درآمدزایی. در اسپرینت جاری.

---

۶. مدیریت انتقال دانش (Knowledge Transfer) و کاهش ریسک فرد کلیدی**

۶,۱. مستندسازی (Documentation):**

* **Onboarding Guide:** یک سند گام‌به‌گام برای توسعه‌دهنده جدید شامل: نصب محیط، راه‌اندازی سرویس‌ها، اجرای تست‌ها، معرفی معماری.

* **ADR (Architectural Decision Record):** مستندات کوتاه مثلاً چرا) که **دلیل** انتخاب‌های فنی مهم در `/docs/adr` را توضیح می‌دهد (؟Prisma ؟ چراTurborepo.

* **Runbooks:** مستندات عملیاتی برای سناریوهای رایج عیب‌یابی و بازیابی (مثلاً "چگونه داده کاربر را از پشتیبان بازیابی کنیم؟").

۶,۲. بازبینی کد اجباری (Mandatory Code Review):**

* **مرج `main` یا `develop` هیچ کدی** مستقیماً به شاخه‌های نمی‌شود.

* هر PR از یک توسعه‌دهنده **(Approval) نیاز به حداقل **یک تأیید دیگر (ترجیحاً کسی که در آن بخش کد تخصص ندارد) دارد.

* این فرآیند، **توزیع دانش** را در تیم افزایش می‌دهد.

و جلسات (Pair Programming) جفت‌برنامه‌نویسی .۶,۳** ###
**:اشتراک دانش

* **روی کارهای ❊❊حیاتی و پیچیده (مثل طراحی معماری Campaign Builder) انجام می‌شود جفت‌برنامه‌نویسی.

* **جلسات هفتگی Tech Talk:** یک عضو تیم یک موضوع فنی (مثلاً "بهینه‌سازی کوئری در Neo4j") را برای بقیه توضیح می‌دهد.

۶,۴. کاهش تکیه بر فرد:

* **مالکیت جمعی (Collective Code Ownership):** با وجود حق ❊❊(Tech Owners) مالک‌های فنی برای ماژول‌ها، اما همه اعضای تیم تغییر کد در هر بخش را دارند (پس از درک کافی) و مسئولیت❊❊.

* **چرخش وظایف:❊❊ توسعه‌دهندگان را تشویق می‌کنیم تا روی بخش‌های مختلف سیستم کار کنند تا دانش متمرکز نشود.

۶,۵. برنامه خروج (Exit Plan) برای اعضای کلیدی:

مثلاً CTO):) در صورت اعلام خروج یک عضو کلیدی

1. **دوره انتقال ۱ ماهه (Transition Period) اجباری.**

2. **تدوین لیست دانش حیاتی (Critical Knowledge List):**
توسط فرد خروج‌کننده.

3. **جلسات انتقال دانش (Knowledge Transfer Sessions):**
ضبط ویدئویی جلسات + مستندسازی.

4. **واگذاری مالکیت دسترسی‌ها (Access Ownership):** تغییر
رمزهای عبور، انتقال مالکیت حساب‌های سرویس.

جمع‌بندی: قابلیت تحویل پایدار و قابل نگهداری

این رویکرد تضمین می‌کند که کدبیس ShahnamehMap نه تنها
امروز کار می‌کند، بلکه **فردا** نیز:

* **قوی، CI/CD پایدار است:** با تست‌های خودکار گسترده و
شکست‌ها سریع شناسایی می‌شوند.

* **قابل نگهداری است:** با استانداردهای کد، ساختار تمیز و مدیریت
Technical Debt فعال است و افزودن ویژگی‌های جدید آسان و کم‌ریسک ..

قابل انتقال است:** با مستندسازی، بازبینی کد و فرهنگ اشتراک ** *
دانش، از انحصار دانش جلوگیری می‌شود و تیم در برابر خروج اعضا مقاوم
.می‌شود

هدف ما "کد تمیز در حد کتاب" نیست، بلکه یک "سیستم مهندسی قابل **
اعتماد" است که به کسب‌وکار اجازه می‌دهد با سرعت و اطمینان رشد
**.کند