[هي أكبر ميزة تنافسية تمتلكها. [1 **(Software Engineering)** دراستك تخصص **هندسة البرمجيات**

يركز على "كيف أكتب كود يظهر التصميم والميزات على الشاشة بـ **Frontend Developer** الفرق الجوهري هو أن الـ يركز على **"بنية النظام البرمجي، الأداء، الأمان، وإمكانية التوسع والتكامل كجزء من Frontend Engineer** بينما الـ ،"React **منظومة هندسية متكاملة"**. [1, 2]

محترف، عليك نقل تركيزك من مجرد "كتابة الواجهات" **Frontend Engineer** لكي تستغل خلفيتك الهندسية وتتحول سريعاً إلى إلى هندستها عبر المسارات التالية:

1. الانتقال من JavaScript العادية إلى TypeScript (إلزامي للهندسة)

-
- **لماذا؟** هندسة البرمجيات تقوم على مبادئ الأنظمة الصارمة والآمنة لتجنب أخطاء وقت التشغيل (Runtime Errors).
- لتعريف Next.jsو React مع **TypeScript** عادية. استخدم JavaScript **ماذا تفعل؟** توقف عن بناء أي مشروع جديد بـ [وهذا يضمن جودة الكود وسهولة صيانته عند كبر حجم المشروع. [2 ،(Types & Interfaces) أنواع البيانات
-

2. هندسة وإدارة البيانات (State Management & Architecture)

-
- أو تمرير البيانات بين المكونات بشكل عشوائي useState **الوضع الحالي:** كمطور مبتدئ قد تعتمد على.
- **Redux Toolkit** أو **Zustand** **العقلية الهندسية:** كمهندس، يجب أن تصمم "معمارية بيانات" للتطبيق. تعلم أدوات مثل لإدارة البيانات العامة، وتعلم كيف تنظم مجلدات مشروعك بناءً على معايير هندسية واضحة وقابلة للتوسع (مثل Feature-Driven Architecture). [2]
-

3. التركيز على قياس وتحسين الأداء البرمجي (Performance Optimization)

-
- المهندس لا يكتفي بأن الكود "يعمل"، بل يدرس كفاءته:
  - المختلفة Rendering Strategies المتقدمة مثل الـ Next.js تعلم ميزات: **SSR** (Server-Side Rendering) و**SSG** (Static Site Generation).
  - تعلم تقنيات تقليل حجم الملفات المكتوبة مثل **Lazy Loading** والـ **Code Splitting**.
  - لتحسين سرعة استجابة الموقع ومؤشرات الويب **Google Lighthouse** افهم كيف تقرأ أدوات القياس مثل الحيوية (Core Web Vitals). [2, 3]
-

4. تطبيق منهجية الاختبار الآلي (Automated Testing)

-
- أكبر دليل على أنك مهندس برمجيات هو كتابة كود يختبر الكود الآخر للتأكد من عدم حدوث أخطاء مستقبلية (Scalability). [2]
- ابدأ بتعلم وكتابة اختبارات أساسية لمشاريعك باستخدام:
  - **Unit Testing** (مثل Jest): لاختبار الدوال والمكونات الصغيرة بشكل منفصل.
  - **E2E Testing** (مثل Playwright أو Cypress): [2] لمحاكاة تجربة مستخدم حقيقية واختبار التطبيق كاملاً.

-

5. فهم بيئة تشغيل الويب وهندسة الشبكات (Under the Hood)

-
- لا تتعامل مع المتصفح والإنترنت كصندوق أسود. كمهندس، تعمق في:
  - كيف يعمل المتصفح داخلياً (DOM, Critical Rendering Path).
  - بروتوكولات الشبكة (HTTP/HTTPS, HTTP/2, Caching Headers).
  - أمن الويب الأساسي وحماية الواجهات من الهجمات الشائعة مثل (XSS و CSRF). [2]
-

🚀 مهارات إضافية مهمة جداً للمهندس المستقبلي

بما أنك تبحث عن التميز المهني، أضف هذه المهارات الثلاث إلى رادارك:

- **CI/CD (Continuous Integration / Continuous Deployment):** تعلم كيف تستخدم **GitHub Actions**. المهندس لا يرفع الموقع يدوياً، بل يضبط إعدادات برمجية تقوم تلقائياً باختبار الكود (Testing) وبناء المشروع (Build) ورفعه إلى نيتليفاي بمجرد عمل `Push` على جيتهاب.
- **Design Systems & Component Storybook:** الشركات الكبرى لا تصمم الأزرار والقوائم في كل مشروع من الصفر. بل تبني "نظام تصميم موحد" (Design System). أداة **Storybook** تمكنك من بناء واختبار مكونات الـ UI بشكل منفصل تماماً عن منطق التطبيق لتكون قابلة لإعادة الاستخدام في أي مشروع مستقبلي للشركة.
- **Web Accessibility (a11y):** هندسة الويب تعني أن يكون موقعك قابلاً للاستخدام من الجميع، بما في ذلك ذوي الاحتياجات الخاصة (كالمكفوفين الذين يستخدمون قارئات الشاشة). فهمك لكيفية كتابة HTML دلالي (Semantic) مهيأ هندسياً لسهولة الوصول يرفع من قيمتك جداً أمام الشركات العالمية والمحلية المحترفة.