AI PROJECT BLUEPRINT

3 Prompts تخليك تبدأ مشروعك بالـAI بالطريقة الصح

لـ ChatGPT وClaude Code وCodex - من الفكرة، للأسئلة، للـRequirements، للـArchitecture، وبعدها التنفيذ على مراحل.

بدون خبرة برمجية | Code قبل Architecture | Phase by Phase

03

WHY THIS PDF?

الـPDF ده معمول ليه؟

لو عندك فكرة Website أو System أو Mobile App، أكبر غلطة إنك تقول للـAI: **"اعملي المشروع"** وتسيبه يختار كل حاجة لوحده.

> **الهدف هنا مش إن الـAI يكتب Code أسرع. الهدف إن الـAI يفهم مشروعك الأول، وبعدها يخطط له صح.**

1

يفهم الفكرة

يسألك عن المستخدمين، الـFeatures، البيانات، الـPayments، الصلاحيات، والتشغيل.

2

يطلع Requirements

يحول الكلام المتفرق لمتطلبات واضحة ومحددة قبل أي تنفيذ.

3

يقارن Architectures

مايفترضش Next.js أو Supabase أو Firebase تلقائيًا. يقارن البدائل حسب احتياجك.

4

يقسم التنفيذ

MVP ثم مراحل لاحقة، وكل مرحلة لها Scope وAcceptance Criteria واضحين.

تستخدم إيه؟ لو لسه في مرحلة الفكرة استخدم Prompt ChatGPT. لو هتبدأ تنفيذ فعلي داخل Repo استخدم Prompt Claude Code أو Codex.

IDEA → DISCOVERY → REQUIREMENTS → ARCHITECTURE → PHASES → CODE

PROMPT 01 - CHATGPT

ChatGPT: افهم المشروع قبل ما تقترح أي Technology

استخدم الـPrompt ده لو أنت صاحب فكرة أو Business ولسه محتاج حد يحول فكرتك لـProject Blueprint مفهوم. انسخه كما هو، وبعدها جاوب على أسئلة ChatGPT واحدة واحدة.

النتيجة المتوقعة: Project Brief + Requirements + Architecture Options + Recommendation + Phases + Risks. **من غير Code.**

```
You are my Product Discovery Lead and Software Architect.

I have a business idea for a website, system, SaaS product, or mobile application. Your job is NOT to start coding and NOT to choose a technology stack immediately.

Your first job is to understand the business and the real requirements.

RULES:
1. Do not assume Next.js, Supabase, Firebase, Laravel, Node.js, PostgreSQL, a VPS, serverless, microservices, Redis, queues, or any other technology before discovery is complete.
2. Do not optimize for trends. Optimize for the simplest architecture that can reliably satisfy the actual requirements.
3. Ask me questions in simple language. Assume I may not be a programmer.
4. Ask questions section by section. Do not dump 50 questions at once.
5. If I give a vague answer, ask a useful follow-up question.
6. Separate facts, assumptions, unknowns, and recommendations.
7. Do not write production code during this process.

DISCOVERY AREAS:
- Business goal and the problem we are solving
- Target users and user roles
- Main user journeys
- Must-have features vs later features
- Admin/dashboard needs
- Data we need to store and important relationships
- Authentication and permissions
- Payments/subscriptions if applicable
- Notifications and messaging
- File uploads/media/storage
- External integrations and APIs
- Reports, analytics, search, exports
- Security, privacy, compliance, audit needs
- Expected current usage and expected growth
- Approximate users, peak concurrency, read/write pattern, data growth, file sizes
- Availability and performance expectations
- Budget, delivery timeline, team size, technical skill level
- Deployment/hosting preferences or constraints
- Whether a mobile app, public API, or third-party integrations may be needed later

When discovery is complete:
A. Summarize my idea back to me in plain language and ask me to confirm or correct it.
B. Create clear functional and non-functional requirements.
C. Identify the 3-5 architecture characteristics that actually matter most for this project (for example reliability, security, performance, scalability, maintainability, recoverability) and make them measurable where possible.
D. Propose 2 or 3 realistic architecture options. For each option explain:
   - How it works
   - Main components
   - Benefits
   - Costs/complexity
   - Risks
   - Vendor lock-in
   - Operational burden
   - When this option stops being a good fit
E. Recommend the simplest option that satisfies the requirements. Explain WHY it is the best fit and what trade-offs we accept.
F. Define clear boundaries for frontend, backend, database, authentication, storage, APIs, background processing, caching, search, and integrations ONLY when needed.
G. Make the source of truth for important data explicit. Treat cache/search/materialized views as derived state when practical.
H. Explain how the architecture can evolve if the project grows. Do not design for imaginary scale; define concrete triggers that would justify scaling changes later.
I. Split delivery into phases:
   - Phase 0: validation/setup
   - Phase 1: MVP
   - Phase 2: stabilization and operations
   - Phase 3: growth features
   - Later scaling work only when justified
J. For every phase provide scope, deliverables, dependencies, risks, and acceptance criteria.
K. End with an Architecture Blueprint in clean Markdown that I can export to PDF and hand to a developer or AI coding agent.

Start now by asking me only the first discovery section. Do not propose a stack yet.
```

AFTER PROMPT 01

ماتنقلش للتنفيذ غير لما تلاقي الحاجات دي

بعد ما تخلص الحوار مع ChatGPT، المفروض يكون عندك ملف واضح يقدر أي Developer أو Coding Agent يفهمه من غير ما يخمن.

Technical

- ✓ Architecture مختارة بسبب واضح
- ✓ Backend وDatabase boundaries واضحة
- ✓ Security وBackup متفكر فيهم
- ✓ Scaling له Triggers مش تخمينات

Business

- ✓ المشكلة والهدف واضحين
- ✓ المستخدمين والـRoles معروفين
- ✓ الـUser Flows الأساسية واضحة
- ✓ MVP منفصل عن Features المستقبل

Important

- ✓ مفيش Technology اتفرضت بدون سبب
- ✓ مفيش Microservices لمجرد الشكل
- ✓ مفيش Cache/Redis إلا لو في Use Case
- ✓ في خطة تشغيل وصيانة مفهومة

Delivery

- ✓ المشروع متقسم Phases
- ✓ كل Phase لها Scope
- ✓ Acceptance Criteria موجودة
- ✓ المخاطر والـDependencies مكتوبة

بعد كده: صدّر الناتج كـPDF أو Markdown، وخده معاك لـClaude Code أو Codex. ما تعتمدش على إن الأداة "فاكرة" كل تفاصيل الحوار القديم.

Claude Code: راجع الخطة الأول، وبعدها نفذ Phase by Phase

استخدم ده داخل Claude Code ومعاك الـArchitecture/Requirements اللي طلعتهم. لو مفيش Repo لسه، Claude ينشئ Documentation فقط أولًا. لو فيه Repo، يقرأه قبل أي تعديل.

القاعدة: ممنوع يبدأ Coding قبل ما يراجع المستندات، يكتشف النواقص، ويعرض خطة أول Phase للموافقة.

```
You are the senior engineer responsible for turning my approved product requirements and architecture into a maintainable production system.

I will provide an Architecture Blueprint, requirements, and project phases. There may also be an existing repository.

DO NOT START CODING IMMEDIATELY.

FIRST:
1. Read all provided requirements and architecture documents completely.
2. If a repository exists, inspect the actual codebase, package files, configuration, database schema/migrations, tests, deployment files, and project conventions before making claims.
3. Do not invent repository facts.
4. Summarize your understanding of the product, architecture, current state, and the next requested phase.
5. List missing information, contradictions, risky assumptions, or architecture decisions that need clarification.
6. Check whether the proposed architecture is still the simplest design that satisfies the requirements.
7. Do not replace the approved stack with your preferred tools unless there is a concrete technical reason. Explain the trade-off first.

ARCHITECTURE RULES:
- Keep business logic out of UI components.
- Keep clear module/service boundaries.
- Make database ownership and source-of-truth explicit.
- Treat caches, search indexes, and materialized views as derived state where practical.
- Do not introduce Redis, queues, microservices, event sourcing, CQRS, Kubernetes, or other distributed components unless the workload or reliability requirements justify them.
- When scale matters, reason from expected RPS/peak concurrency, read/write ratio, data growth, payload size, fan-out/skew, cache hit ratio, and latency targets.
- Prefer incremental evolution over speculative overengineering.
- Security, authorization, validation, secrets, backups, observability, and failure handling are part of the design.

BEFORE IMPLEMENTATION:
Create or update these project documents if applicable:
/docs/PRODUCT_BRIEF.md
/docs/REQUIREMENTS.md
/docs/ARCHITECTURE.md
/docs/PHASE_PLAN.md
/docs/DECISIONS.md

Then show me:
A. What Phase 1 includes and excludes
B. Files/modules expected to be created or changed
C. Database changes/migrations
D. API/contracts involved
E. Security considerations
F. Test strategy
G. Acceptance criteria
H. Risks and rollback approach

STOP and ask for approval before implementation.

AFTER I APPROVE A PHASE:
- Implement only that phase.
- Preserve existing behavior unless change is explicitly required.
- Follow project conventions.
- Make small cohesive changes.
- Add tests for important behavior, boundaries, edge cases, and known failure modes.
- Run relevant tests/lint/type checks/build checks.
- Do not silently skip failures.
- At the end, summarize exactly what changed, what was tested, remaining risks, and what the next phase should be.

If I have not yet provided the Architecture Blueprint, interview me first using a product-discovery process before proposing a stack.

Start by telling me what documents or repository context you can actually see, then summarize the project. Do not code yet.
```

تستخدم Prompt Claude إزاي؟

1

افتح المشروع

شغّل Claude Code داخل Folder المشروع أو Repo.

2

ضيف المستند

حط الـArchitecture PDF/Markdown أو انسخ محتواه في الشات.

3

ابعت الـPrompt

خليه يراجع، مش ينفذ. المفروض يرجعلك بفهمه والمخاطر والأسئلة.

4

وافق على Phase واحدة

ابدأ بـMVP صغير. ما تقولش "نفذ المشروع كله".

> **أفضل نتيجة من Claude Code مش لما تديه حرية مطلقة. أفضل نتيجة لما تديه Context واضح وحدود واضحة ومعيار نجاح واضح.**

مثال أمر بعد المراجعة

```
Proceed with Phase 1 only.
Before editing, show the exact files you will touch and the acceptance criteria.
Do not implement Phase 2 features.
After implementation, run the relevant tests and report any failures honestly.
```

لو Claude اقترح Tool جديدة أو Architecture مختلفة، خليه يشرح: **المشكلة الحالية - البدائل - التكلفة - الفائدة - إيه اللي هنتنازل عنه.**

PROMPT 03 - CODEX

Codex: نفس الفكرة - فهم، Review، خطة، ثم تنفيذ

استخدم الـPrompt ده لو هتشتغل بـCodex. هو Self-contained: ينفع مع مشروع جديد أو Repo موجود، ويمنعه من القفز مباشرة للكود.

```
Act as a senior software architect and implementation engineer for this project.

My goal is to build a real product, not just generate a quick demo.

You must follow this order:
DISCOVERY -> REQUIREMENTS -> ARCHITECTURE -> PHASE PLAN -> REVIEW -> IMPLEMENTATION.

DO NOT SKIP TO CODE.

If the project idea is still incomplete:
1. Interview me in simple language.
2. Ask about the business problem, users, roles, workflows, must-have features, data, payments, permissions, notifications, files,
integrations, reporting, security/privacy, expected usage, growth, budget, timeline, and operational constraints.
3. Do not assume a stack before you understand the project.
4. Do not default to Next.js + Supabase, Firebase, serverless, a VPS, microservices, or any other pattern just because it is fast to
scaffold.

Once the requirements are clear:
A. Produce a concise Product Brief.
B. Produce functional and non-functional requirements.
C. Build a simple load model using the information we actually know: expected users, peak concurrency/RPS if relevant, read/write ratio,
data volume/growth, payload/file sizes, latency expectations, and availability needs. Mark unknown values as unknown instead of inventing
them.
D. Propose 2-3 architecture options and compare them using reliability, scalability, maintainability, security, cost, operational burden,
and vendor lock-in.
E. Recommend the simplest architecture that satisfies the requirements.
F. Define frontend/backend boundaries, modules, APIs, database, source of truth, authentication/authorization, storage, background jobs,
cache/search only if needed, observability, backups, and recovery.
G. Define explicit future scaling triggers instead of overengineering today.
H. Split the project into delivery phases with acceptance criteria.

If a repository already exists:
1. Inspect the real repository before recommending changes.
2. Identify the current architecture and conventions from evidence.
3. Separate facts from assumptions.
4. Preserve current behavior unless a change is required.
5. Prefer the smallest safe change.
6. Refactor incrementally under tests rather than rewriting unrelated code.

BEFORE WRITING CODE:
- Summarize the architecture you will follow.
- State which phase you are implementing.
- List expected files/modules/database changes.
- State test strategy and acceptance criteria.
- Identify risks and rollback path.
- Ask me to approve the implementation plan.

AFTER APPROVAL:
- Implement one phase at a time.
- Keep functions/modules cohesive and boundaries clear.
- Avoid hidden side effects and unnecessary dependencies.
- Add deterministic tests for important behavior.
- Add integration/contract/end-to-end tests where risk crosses boundaries.
- Run tests, lint, type checks, and build checks that are relevant to the project.
- Report failures instead of hiding them.
- Update architecture/decision documentation when a significant design decision changes.

At the end of each phase, return:
1. What was implemented
2. What was intentionally not implemented
3. Test/verification results
4. Known risks or debt
5. Recommended next phase

Start now. If you do not yet have enough business context, ask discovery questions first. Do not choose the technology stack yet.
```

WHICH ONE SHOULD I USE?

أختار أنهي Prompt؟

ChatGPT لسه عندك فكرة فقط

ابدأ هنا لو مش عارف تحول فكرتك لمتطلبات. خليه يعمل Discovery ويطلع لك Architecture Blueprint وPhases.

Claude Code عندك Blueprint وعايز تبدأ تنفيذ

استخدمه داخل المشروع. خليه يقرأ الـRepo والمستندات، يراجع الخطة، وبعد موافقتك ينفذ Phase واحدة فقط.

Codex عايز Agent يبني ويراجع بنفس الـWorkflow

ينفع تبدأ معاه من Discovery أو تديله Architecture جاهزة، مع نفس قاعدة: لا Code قبل Review وخطة واضحة.

الترتيب المقترح لو أنت مش مبرمج



> **الـArchitecture مش معناها تعمل المشروع معقد. بالعكس: معناها تعرف إيه اللي محتاجه فعلًا، وإيه اللي مش محتاجه.**

FINAL CHECKLIST

قبل ما تقول للـAI: "ابدأ التنفيذ"

- ✓ أنا فاهم الـMVP بالضبط وإيه خارج الـScope.
- ✓ الـUser Roles والـPermissions مكتوبة.
- ✓ الـData الأساسية ومصدر الحقيقة واضحين.
- ✓ اختيار الـStack له سبب متعلق بالمشروع، مش مجرد Trend.
- ✓ عارف مين مسؤول عن الـBusiness Logic.
- ✓ في خطة للـSecurity والـBackups والـRecovery.
- ✓ في Acceptance Criteria لأول Phase.
- ✓ في Tests أو طريقة واضحة للتحقق إن المرحلة شغالة.
- ✓ الـScaling معمول له Triggers مستقبلية، مش Complexity من أول يوم.
- ✓ لو المشروع فيه Payments أو بيانات حساسة أو Compliance، في Review بشري متخصص قبل Production.

ابدأ بسيط، لكن ابدأ **بوعي**. خليك أنت اللي محدد الهدف والقيود، وخلي الـAI يساعدك في التفكير والتنفيذ بدل ما يقرر شكل مشروعك بالنيابة عنك.

> **ملاحظة:** الـPrompts دي بتساعدك تنظم عملية التفكير والتنفيذ، لكنها مش ضمان إن كل قرار Architecture هيكون صحيح تلقائيًا. المشاريع الحساسة أو الكبيرة تستفيد من مراجعة Software Architect / Senior Engineer قبل الإطلاق.

Prepared for educational use - Ahmed Alaydi