AI PROJECT BLUEPRINT

من فكرة **Business** إلى **Architecture** واضحة ثم تنفيذ منظم بـ **Claude Code**

دليل عملي لغير المبرمجين وأصحاب المشاريع: كيف تجعل ChatGPT يفهم الفكرة أولاً، يحولها إلى Requirements وArchitecture ومراحل تنفيذ، ثم تسلم كل مرحلة لـClaude Code بدون أن يبدأ المشروع من Stack عشوائي أو Code متسرع.

Claude Code | Phases | Architecture | Requirements | Discovery

إعداد: Ahmed Alaydi - نسخة عملية للاستخدام المباشر

الفكرة الأساسية

> لا تطلب من الـ**AI** أن «يفهم ويقرر ويصمم ويبرمج» في **Prompt** واحد.

أفضل Workflow للمشاريع الحقيقية هو أن تفصل القرارات إلى مراحل. في البداية نخلي ChatGPT يعمل كـBusiness Analyst وSolution Architect، وبعد أن تصبح الصورة واضحة نستخدم Claude Code كـImplementation Agent.

1. **Discovery**: فهم الفكرة، المستخدمين، العمليات، القيود والهدف التجاري.
2. **Requirements**: تحويل الحوار إلى متطلبات واضحة وقابلة للقياس.
3. **Architecture Options**: مقارنة أكثر من تصميم بدل القفز مباشرة إلى Stack واحد.
4. **Architecture Document**: توثيق القرار النهائي، الـModules، البيانات، الـAPIs، الأمان والـDeployment.
5. **Phases**: تقسيم التنفيذ إلى MVP ثم مراحل توسع واضحة.
6. **Claude Code**: تنفيذ Phase واحدة في كل مرة، مع Testing ومراجعة قبل الانتقال لما بعدها.

قاعدة الدليل: الـArchitecture هي التي تختار الأدوات المناسبة. لا تجعل أداة معينة - Supabase أو Firebase أو VPS أو أي Framework - تفرض شكل المشروع قبل أن نفهم احتياجاته.

قبل أن تبدأ: 5 قواعد تمنع الـ**AI** من تخريب الـ**Scope**

- لا تسمح بالحل قبل الفهم: في مرحلة Discovery ممنوع اقتراح Stack أو Code.
- افصل الـ**Functional** عن الـ**Non-functional Requirements**: ماذا يفعل النظام؟ وكيف يجب أن يعمل من ناحية الأداء والأمان والاعتمادية؟
- اطلب بدائل و**Trade-offs**: لا تقبل Recommendation واحدة بدون مقارنة.
- اكتب الافتراضات والأسئلة المفتوحة: أي شيء غير معروف يجب أن يظل Unknown، لا يتم اختراعه.
- نفّذ على مراحل: لا تطلب من Coding Agent تنفيذ المشروع كله مرة واحدة.

الصح

"Interview me first. Do not propose a solution until the requirements are sufficiently clear."

هذا يجعل الـAI يجمع Context قبل القرار.

غلط شائع

"Build me a complete SaaS with Next.js and Supabase."

هذا يختار الحل قبل أن يفهم المشكلة.

ملاحظة: لا يوجد Architecture واحدة صحيحة لكل المشاريع. الاختيارات تتغير حسب الـBusiness، الفريق، الميزانية، المتطلبات القانونية، مستوى التوفر المطلوب وحجم الاستخدام المتوقع.

Prompt: Discovery Interview 1

استخدمه أولاً في **ChatGPT**. اكتب تحته وصف بسيط لفكرتك - حتى لو سطرين فقط. هدفه أن يحول ChatGPT إلى Business Analyst يسألك حتى يفهم المشروع.

```
You are a senior Business Analyst and Product Discovery Consultant.

I have a business idea for a website, software system, SaaS product, or mobile application.
Your job is NOT to propose technologies, architecture, database choices, UI, or implementation
yet.

Your first job is to understand the business completely.

INTERVIEW RULES:
1. Ask me structured questions in small batches (maximum 5 questions at a time).
2. Start with the business problem, target users, value proposition, and current workflow.
3. Then cover user types and roles, key workflows, required features, admin operations, data,
permissions, payments, notifications, integrations, files/media, reporting, analytics, search,
localization, and support needs.
4. Ask about expected traffic, geographic scope, sensitive data, compliance constraints,
availability requirements, budget sensitivity, timeline, team size, and future growth.
5. Distinguish must-have MVP requirements from nice-to-have and future requirements.
6. Challenge vague answers politely. If I say "I need a dashboard" or "I need AI," ask what
business outcome it must produce.
7. Never invent missing information. Mark unknown items explicitly.
8. Do not recommend a stack until the discovery is complete.
9. At the end, summarize what you understood and show me the remaining open questions.
10. Only when I explicitly say "Discovery approved" should you move to requirements
documentation.

Start by asking me about the business problem and the people who will use the product.

My initial idea:
[PASTE YOUR IDEA HERE]
```

متى تنتقل للخطوة التالية؟ عندما تقرأ الـsummary وتقول: نعم، ده فعلًا المشروع اللي في دماغي، والأسئلة المفتوحة أصبحت قليلة وواضحة.

Prompt: Requirements / PRD 2

بعد ما تخلص الحوار، استخدم هذا الـPrompt في نفس المحادثة. هدفه تحويل الكلام إلى وثيقة واضحة قبل الدخول في الـArchitecture.

```
Discovery approved.

Now act as a senior Product Manager and Requirements Engineer.
Using ONLY the information confirmed in our discovery conversation, create a structured Product Requirements Document (PRD).

The PRD must contain:
1. Executive summary
2. Business problem
3. Product goals
4. Explicit non-goals
5. Target users/personas
6. User roles and permissions matrix
7. Core user journeys
8. Functional requirements grouped by module
9. Non-functional requirements:
   - performance
   - availability
   - scalability
   - security
   - privacy
   - accessibility
   - maintainability
   - observability
   - backup/recovery
10. Data entities and important relationships (conceptual only)
11. External integrations
12. Notifications and communication channels
13. Reporting and analytics requirements
14. MVP scope
15. Post-MVP scope
16. Acceptance criteria for each major feature
17. Risks, assumptions, dependencies, and open questions
18. Success metrics / KPIs

Rules:
- Do not choose technologies yet.
- Do not add features that I did not request unless you label them clearly as optional recommendations.
- Mark every unknown item as TBD.
- Separate business requirements from implementation suggestions.
- End with a short "Architecture Inputs" section summarizing the constraints an architect must consider.

Before finalizing, check the document for contradictions or missing critical requirements and list them.
```

Prompt: Architecture Options 3

هنا فقط نسمح للـAI يناقش Stack. المطلوب منه 3 اتجاهات واضحة بدل Recommendation واحدة تلقائية.

```
Now act as a senior Software Architect.

Use the approved PRD and Architecture Inputs. Do NOT assume that Next.js, Supabase, Firebase,
microservices, serverless, or a VPS is automatically the correct choice.

First, derive the architectural drivers from the requirements: scale, security, availability,
team capability, cost sensitivity, data complexity, integrations, deployment constraints, and
expected growth.

Then propose THREE architecture options:

OPTION A - Lean / MVP-first
Optimized for speed, simplicity, and low operational overhead.

OPTION B - Balanced / Production-ready
Optimized for maintainability, clear backend boundaries, reasonable scale, and a practical path
to growth.

OPTION C - Scale-first / High-control
Optimized for higher scale, stronger isolation, operational control, and future complexity
where justified.

For each option provide:
- architecture style
- frontend approach
- backend/API approach
- database choice and why
- authentication/authorization
- file/object storage
- caching strategy
- background jobs/queues
- realtime needs (if any)
- search strategy (if needed)
- third-party integrations
- deployment/infrastructure
- environments (dev/staging/prod)
- logging, monitoring, and alerts
- backups and disaster recovery
- security boundaries
- scalability path
- estimated operational complexity: Low / Medium / High
- estimated relative infrastructure cost: Low / Medium / High
- main advantages
- main disadvantages
- vendor lock-in considerations
- migration path if the product outgrows this option

Then create a comparison table and recommend ONE option.

Your recommendation must be based on the requirements, not personal preference.
```

```
Explain the trade-offs clearly enough for a non-technical business owner to understand.
If a managed platform such as Supabase/Firebase is recommended, explain what is managed, what
remains our responsibility, and the future migration strategy.
If a VPS/cloud VM is recommended, explain the operational responsibilities it introduces.

Do not write implementation code.
```

النقطة المهمة: لو ChatGPT اختار Supabase بعد المقارنة - مفيش مشكلة. الفرق إن الاختيار هنا أصبح Decision له أسباب وTrade-offs، مش Default تلقائي.

4 Prompt: Architecture Document جاهز للتصدير PDF

بعد اعتماد Option معينة، استخدم هذا الـPrompt. سيحول القرار إلى مستند يمكن مشاركته مع Developer أو Agency أو Coding Agent.

```
Architecture option approved: [OPTION NAME].

Create a professional Software Architecture Document based on the approved PRD and selected
architecture.
The audience is both a business owner and the engineering team.

Document structure:
1. Cover / project name / document version
2. Executive architecture summary
3. Business and technical context
4. Key architectural drivers and constraints
5. Scope and non-scope
6. System context diagram
7. Container / high-level component diagram
8. Modules and responsibilities
9. Main user flows / sequence diagrams for critical workflows
10. API boundaries and integration boundaries
11. Data architecture
    - main entities
    - ownership
    - relational boundaries
    - indexes that will likely matter conceptually
    - data retention considerations
12. Authentication and authorization model
13. Security architecture
14. File/media storage architecture
15. Caching strategy
16. Background jobs / queues
17. Notifications
18. Search / realtime architecture when relevant
19. Deployment topology
20. Environments: local, development, staging, production
21. CI/CD expectations
22. Logging, monitoring, metrics, tracing, and alerting
23. Backup, restore, and disaster recovery
24. Performance and scalability strategy
25. Cost and operational considerations
26. Major architectural decisions and trade-offs
27. Known risks and mitigations
28. Future evolution path
29. Architecture Decision Records (ADR) list
30. Open questions / TBD items

Diagram rules:
- Provide Mermaid diagrams where useful.
- Keep diagrams high-level and readable.
- Do not invent external systems, traffic numbers, SLAs, or compliance requirements.

Quality rules:
```

```
- Clearly separate confirmed requirements, assumptions, recommendations, and TBD items.
- Explain technical terms briefly for a non-technical reader.
- Do not over-engineer for hypothetical scale.
- Prefer modular boundaries and explicit responsibilities.
- Include a final architecture review checklist covering security, reliability, performance, operations, cost, and maintainability.

Output the document in polished Markdown suitable for export to PDF.
If your environment supports file generation, also create a PDF version named:
[PROJECT_NAME]_Software_Architecture_v1.pdf
```

5 Prompt: تقسيم المشروع إلى Phases

دي أهم خطوة قبل Claude Code. بدل "ابني المشروع"، نخلي الـAI يحدد Deliverables صغيرة لها Definition of Done.

```
Using the approved PRD and Software Architecture Document, create an implementation roadmap.

Break the work into phases that can be implemented and verified independently.

Use this structure:

PHASE 0 - Engineering foundation
Examples: repository structure, environments, configuration, database migration strategy, CI, code quality, test foundation, logging, documentation conventions.

PHASE 1 - Smallest usable MVP core
Only the minimum end-to-end workflow that proves the product's main value.

PHASE 2+ - Incremental business capabilities
Add modules in dependency order.

FINAL PHASE - Production readiness
Security review, performance checks, backups, monitoring, deployment runbook, recovery test, documentation, and launch checklist.

For every phase include:
- goal
- business value
- scope
- explicit out-of-scope
- dependencies
- affected modules
- database changes
- API changes
- UI/screens involved
- integrations involved
- security considerations
- testing requirements
- acceptance criteria
- Definition of Done
- rollback / migration considerations
- documentation that must be updated

Rules:
- Keep phases small enough that an AI coding agent can complete one phase without losing context.
- Do not mix unrelated features in one phase.
- Do not implement future-scale complexity before it is justified.
- Identify any decision that requires human approval before coding.
- Create a final dependency map showing which phases can run in parallel and which cannot.

At the end, generate a short handoff summary that I can paste into Claude Code before starting implementation.
```

الآن فقط: نبدأ Claude Code

ممنوع: ترمي الـPDF وتقول "Build everything". ده بيرجعنا لنفس الغلطة ولكن بمستند أكبر.

الصح: ادي Claude Code الـPRD + Architecture + Roadmap، وخليه يراجعهم أولاً، وبعدها نفذ Phase واحدة فقط.

Anthropic نفسها توصي بتعليمات واضحة، خطوات مرتبة، Context كافي، والتحقق من العمل في المهام الطويلة.
في مشاريع طويلة، تتبع الحالة والاختبارات وGit يساعد الـAgent يفضل فاهم هو وصل لفين.

6 Claude Code - Project Intake / No Coding Yet

```
You are the implementation engineer for this project.

Before writing any code, read all project documents provided to you:
- Product Requirements Document (PRD)
- Software Architecture Document
- Implementation Roadmap / Phases

FIRST TASK: review only. Do not implement anything yet.

Produce:
1. Your understanding of the product in 10-15 bullets.
2. The architecture and module boundaries you must preserve.
3. The approved technology choices and the reasons behind them.
4. Missing information or contradictions that could affect implementation.
5. High-risk areas: security, data migrations, payments, permissions, integrations, destructive
operations.
6. A proposed repository/folder structure consistent with the architecture.
7. The commands and tools you expect to use for linting, tests, builds, and local development.
8. A short checklist for Phase 0.

Rules:
- Do not replace architecture decisions silently.
- Do not introduce a new framework, database, service, or dependency without explaining why it
is necessary.
- Do not code until the open blockers are resolved or explicitly accepted as assumptions.
- Never speculate about existing code. Inspect files before making claims.
- Prefer simple, maintainable solutions over unnecessary abstractions.

End with: "READY FOR PHASE 0" only if the project is sufficiently clear.
```

7 Claude Code - تنفيذ Phase واحدة

انسخ الـPhase المطلوبة من الـRoadmap وضعها مكان [PHASE SPEC].

```
Implement ONLY the phase below.

[PHASE SPEC]

WORKFLOW:
1. Inspect the existing repository and relevant files first.
2. Restate the phase goal, dependencies, and Definition of Done.
3. Create a short implementation plan before editing.
4. Identify files/modules/database migrations/API contracts that will change.
5. Implement incrementally, preserving the approved architecture and existing conventions.
6. Add or update tests for the behavior introduced in this phase.
7. Run the relevant test suite, linting/type checks, and build checks.
8. For database changes, use migrations. Do not make destructive changes without explicit
approval.
9. For external services, keep secrets in environment configuration and never hard-code
credentials.
10. Do not add features outside this phase.
11. Do not refactor unrelated code unless the phase cannot be completed safely without it; if
so, explain first.
12. Update documentation and architecture notes if implementation details materially change.

BEFORE FINISHING:
- Verify each acceptance criterion one by one.
- List tests/checks actually run and their results.
- List files changed.
- List migrations and rollback notes.
- List any technical debt or follow-up work, but do not implement future phases.

If something in the phase conflicts with the architecture or requirements, STOP that part and
explain the conflict instead of guessing.
```

Claude Code - Architecture & Quality Review 8

استخدمه بعد كل 2-3 Phases أو قبل Production. الهدف مش إضافة Features، بل اكتشاف الانحراف عن التصميم والمخاطر.

```
Perform a review of the current codebase against the approved PRD and Software Architecture
Document.

This is a REVIEW task first. Do not start broad refactoring automatically.

Check:
1. Architecture conformance and module boundaries
2. Business logic placement
3. API contracts and validation at system boundaries
4. Authentication, authorization, and permission enforcement
5. Sensitive data handling and secrets
6. Database schema quality, migrations, query patterns, indexes, and transaction boundaries
7. Error handling and idempotency where relevant
8. Queue/background-job reliability where relevant
9. Caching correctness and invalidation where relevant
10. File upload/storage security where relevant
11. Logging, monitoring, and auditability
12. Test coverage for critical business flows
13. Performance bottlenecks and N+1/query issues
14. Deployment/configuration risks
15. Backup and restore readiness
16. Vendor lock-in or architecture drift
17. Dead code, duplicated business logic, and unnecessary abstractions

Output:
- Critical issues
- High-priority issues
- Medium-priority issues
- Low-priority improvements
- Architecture deviations
- Recommended fixes in dependency order

For each issue include evidence from specific files or code paths.
Do not claim a problem exists without inspecting the relevant code.
```

9 Prompt: Change Request بدون تكسير المشروع

العميل دائمًا هيغير رأيه. قبل ما تقول لـClaude "زود Feature"، استخدم Impact Analysis.

```
We have a new change request:

[DESCRIBE THE NEW FEATURE OR BUSINESS CHANGE]

Do NOT implement it yet.
Act as a software architect and perform an impact analysis against the current PRD,
architecture, roadmap, database, APIs, and completed phases.

Analyze:
- business value
- affected user roles and workflows
- affected modules
- data model changes
- API changes
- permission/security impact
- integrations impact
- migration/backfill needs
- performance/scalability impact
- deployment impact
- testing impact
- backward compatibility
- risks
- whether this belongs in the current phase or a new phase

Provide 2-3 implementation approaches when meaningful, with trade-offs.
Recommend the smallest safe change that preserves architectural consistency.

Then produce an updated phase specification and Definition of Done.
Wait for approval before implementation.
```

Prompt: Production Readiness Gate 10

آخر Prompt قبل Launch. لا نريد "بيشتغل عندي" فقط.

```
Act as a production-readiness reviewer.

Using the approved architecture and the current repository, create a launch gate for this
product.
Do not deploy or make destructive changes unless explicitly authorized.

Review and verify where possible:
- production environment configuration
- secrets management
- authentication and authorization
- rate limiting / abuse protection where relevant
- input validation and file upload controls
- database migrations and rollback plan
- backups and a tested restore procedure
- logging and alerting
- health checks
- performance baseline and capacity assumptions
- critical user journey tests
- payment/webhook idempotency where relevant
- email/SMS/notification failure handling
- queues and retry/dead-letter behavior where relevant
- storage permissions and lifecycle
- DNS/TLS/domain configuration
- CI/CD and deployment rollback
- privacy/data retention requirements
- admin/audit access
- monitoring dashboards
- incident/runbook documentation

Output a table with:
Item | Status (PASS / BLOCKED / NOT APPLICABLE) | Evidence | Required Action

Then provide:
1. Launch blockers
2. Non-blocking risks
3. Rollback plan
4. First 24-hour monitoring checklist
5. First 7-day post-launch checklist

Never mark something PASS without evidence.
```

Workflow مختصر تحفظه

المرحلة	الأداة	الناتج	ممنوع تعمل إيه؟
1. Discovery	ChatGPT	فهم الفكرة + أسئلة مفتوحة	اختيار Stack
2. Requirements	ChatGPT	PRD + Scope + Acceptance Criteria	اختراع Features
3. Architecture	ChatGPT	3 Options + Trade-offs	الاعتماد على Default واحد
4. Architecture Doc	ChatGPT	مستند Architecture + Diagrams	إخفاء Unknowns
5. Phases	ChatGPT	Roadmap + Definition of Done	Phase ضخمة جدًا
6. Intake	Claude Code	فهم الـRepo + المخاطر	بدء Coding فورًا
7. Implementation	Claude Code	Phase واحدة + Tests	Scope Creep
8. Review	Claude Code	Architecture/Quality Audit	Refactor شامل بلا سبب
9. Change	ChatGPT/Claude	Impact Analysis	إضافة Feature مباشرة
10. Launch	Claude Code	Production Readiness Gate	PASS بدون Evidence

النتيجة: حتى لو مش Developer، هتدخل على التنفيذ وأنت عندك Scope واضح، Architecture مبررة، مراحل صغيرة، ومعيار واضح لمعنى "خلصت".

Checklist قبل ما تسلم أي **Phase** للـ**AI**

- هل الهدف التجاري للـPhase واضح؟
- هل الـScope والـOut-of-scope مكتوبين؟
- هل الـDependencies معروفة؟
- هل الـUser Roles والصلاحيات واضحة؟
- هل تغييرات الـDatabase والـAPI محددة؟
- هل Acceptance Criteria قابلة للاختبار؟
- هل في عمليات حساسة تحتاج Approval بشري؟
- هل في Rollback/Migration Plan؟
- هل الـAI ممنوع من إضافة Features خارج المرحلة؟
- هل لازم يحدث Docs بعد التنفيذ؟

متى تحتاج **Developer / Architect** حقيقي؟

استخدام الـAI لا يلغي الحاجة لخبرة هندسية عندما يكون المشروع عالي المخاطر: بيانات طبية/مالية، متطلبات قانونية، Payments معقدة، أنظمة حرجة، Scale كبير، Integrations حساسة، أو عمليات قد تسبب خسارة مالية أو Data Loss. في الحالات دي استخدم الـAI لتسريع التحليل والتنفيذ، وليس كبديل عن المراجعة المتخصصة.

ليه الـ**Workflow** ده أفضل من **Prompt** واحد؟

قرارات **Architecture** مبررة

بدل Default Stack، الاختيار يرتبط بالمتطلبات والـTrade-offs.

وضوح أعلى

كل مرحلة لها Output محدد، فأسهل تراجعها وتصححها قبل ما الخطأ يتحول إلى Code.

أسهل في التغيير

أي Feature جديدة تمر على Impact Analysis بدل ما تتضاف عشوائي.

أقل **Scope Creep**

Claude Code ينفذ Phase واحدة ويقف عند Definition of Done.

أقوى للعميل

حتى غير التقني يقدر يشوف Scope، قرارات، مراحل، مخاطر ومتى يدفع لتنفيذ كل جزء.

أسهل في الـ**Scaling**

التوسع يبقى Evolution Path معروف، مش إعادة كتابة مفاجئة.

المصادر التي بُني عليها الـWorkflow

الدليل عملي ومصمم للاستخدام مع مشاريع حقيقية. اعتمد في منهجيته على مبادئ حديثة من مصادر رسمية:

- **OpenAI - Prompt Engineering**: وضوح التعليمات، إعطاء Context كافي، تقسيم المهام المعقدة، والعمل بشكل Iterative.
- **Anthropic - Prompting & Agentic Workflows**: تعليمات مباشرة، خطوات منظمة، إدارة الحالة في المهام الطويلة، استخدام Tests وGit، وعدم التخمين عن Code لم يتم فحصه.
- **AWS Well-Architected**: تقييم القرارات عبر الأمن، الاعتمادية، الأداء، التشغيل، التكلفة والاستدامة بدل التركيز على Technology واحدة.

OpenAI: https://help.openai.com/en/articles/10032626-prompt-engineering-best-practices-for-chatgpt
OpenAI API Prompt Engineering: https://developers.openai.com/api/docs/guides/prompt-engineering
Anthropic Prompting Best Practices: https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/prompt-templates-and-variables
Anthropic Claude Code: https://docs.anthropic.com/en/docs/claude-code/getting-started
AWS Well-Architected: https://docs.aws.amazon.com/wellarchitected/latest/userguide/waf.html

---

هذا الدليل ليس Architecture جاهزة لمشروع بعينه. الهدف هو إجبار الـAI على فهم المشروع، توثيق الافتراضات، مقارنة البدائل، ثم التنفيذ تدريجيًا.