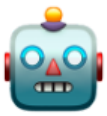
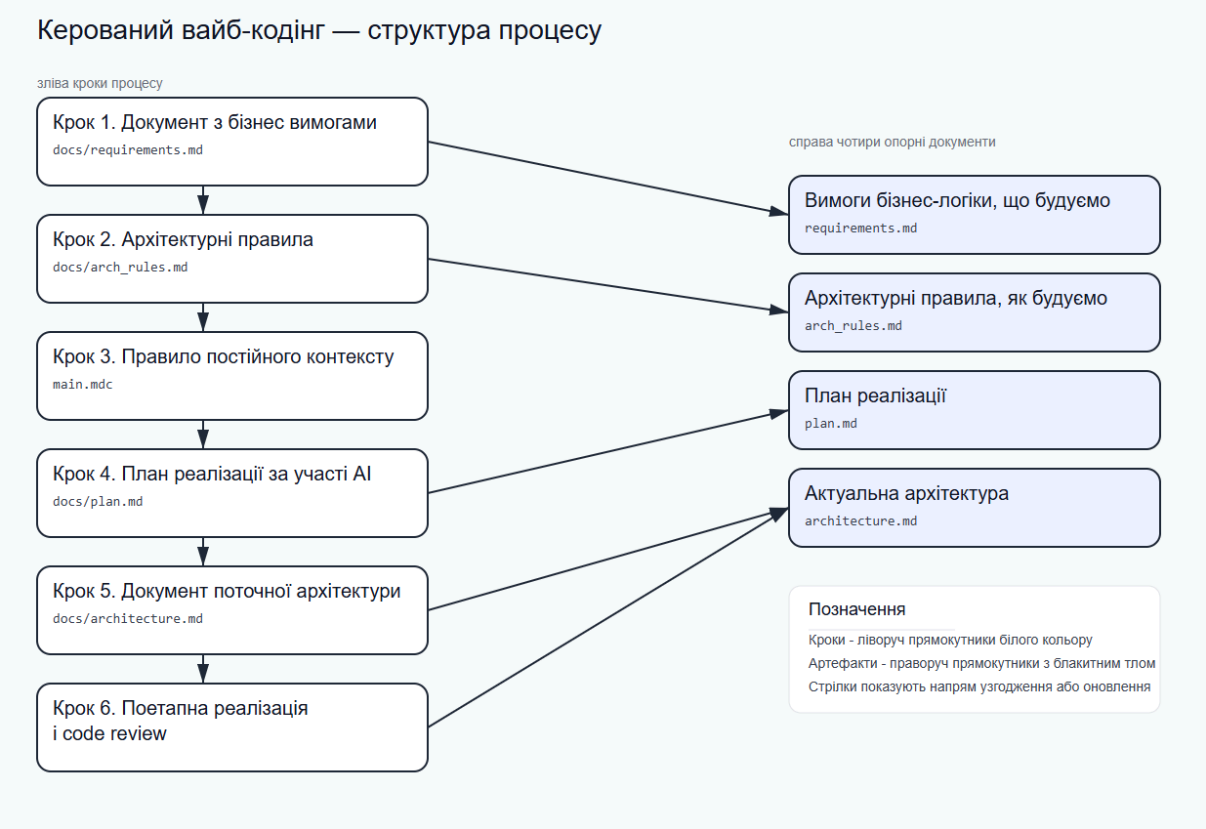


Керований вайб-кодінг

Цей документ фіксує послідовний процес розробки, у якому ми керуємо взаємодією з AI так, щоб код відповідав бізнес-логіці та архітектурі. Ми працюємо поетапно, підтримуємо єдине джерело інформації в проєкті та постійно уточнюємо правила, коли виявляємо системні помилки.



Перший крок — документ з бізнес вимогами

Мета полягає у створенні повного контексту продукту для AI і команди. Ми описуємо, що саме повинен робити сервіс, які обмеження приймаємо та як виглядатимуть зовнішні контракти.

Дія: Створіть файл `requirements.md` (наприклад, у папці `docs/`) та наповніть його. Ви можете написати вимоги вручну або скористатися допомогою AI

(наприклад, ChatGPT), щоб структурувати їх

Рекомендований вміст документа

- **Загальна інформація**. Базовий URL, принцип автентифікації та авторизації, формат відповідей, політика версіонування API, політика використання часових зон (timezone).
- **Обмеження**. Політики, такі як ліміти запитів (rate limiting) та ідемпотентність.
- **Формати помилок:** Стандартизований опис того, як сервіс повідомляє про помилки.
- **Ключові сценарії (Endpoints)**. Для кожної функції вказати HTTP-метод, шлях, тіло запиту та очікувану відповідь.
- **Додаткова інформація**. Модель даних у контексті використання, індекси БД на критичних шляхах, вимоги до трекінгу, логування та observability.

Результат — створений документ `docs/requirements.md`, який стає основою подальшої роботи.

Другий крок — документ з архітектурними правилами

Мета: Встановити технічні обмеження та вказати, **як** потрібно реалізовувати проєкт, щоб уникнути постійних виправлень коду, згенерованого AI.

Дія: Створіть файл `docs/arch_rules.md` та опишіть у ньому всі технічні та структурні вимоги

Рекомендований вміст документа

- **Технологічний стек**. Фреймворки та бібліотеки, наприклад NestJS, zod, pg або Prisma, інструменти для тестування і форматування коду.
- **Архітектурні патерни**. Підходи на кшталт CQRS, feature sliced design, залежності всередині модулів і правила імпорту.
- **Структурні правила**. Дерево проєкту, іменування файлів і директорій, організація модулів, принцип окремих пакетів для домену та транспорту.

- **Версії інструментів**. Вимоги до середовища, наприклад Node.js двадцята версія і вище, менеджер пакетів, версія TypeScript.
- **Заборони**. Технології та підходи, які не використовуємо, наприклад class validator у поєднанні з декораторами, без-типові обгортки над HTTP клієнтами.

Результат — створений документ `docs/arch_rules.md`, який діє як технічне завдання для AI.

Третій крок - налаштування правила для постійного контексту AI

Мета: Об'єднати створені документи в єдину систему, щоб AI-агент **обов'язково читав їх при кожному запиті**.

Дія: Створіть нове правило в налаштуваннях вашої IDE. В нашому випадку це Cursor.

Порядок налаштування:

1. Відкрийте **Preferences → Cursor Settings → Rules**.

2. Створіть **New Project Rule**.

3. Назвіть правило наприклад, `"main"`.

4. Встановіть прапорець `"Apply always: true"`, щоб правило застосовувалося до всіх запитів.

5. В описі правила **явно вкажіть шляхи до ваших документів**:

- `Усі правила архітектури описані в @docs/arch_rules.md`
- `Усі бізнес-вимоги до сервісу описані в @docs/requirements.md`

Результат: Налаштоване правило в IDE, яке змушує AI автоматично використовувати контекст з `requirements.md` та `arch_rules.md` у всіх подальших задачах.

Четвертий крок — план реалізації за участі AI

Мета: Уникнути хаотичної реалізації та структурувати роботу AI, змусивши його спочатку створити покроковий план. Цей крок є обов'язковим для нових систем. План зберігаємо у файлі `docs/plan.md`.

Порядок дій

- Створюємо порожній файл `plan.md` в `docs`.

- Дайте AI команду **проаналізувати файли** `requirements.md` **та** `arch_rules.md`**, створені вами на попередніх кроках**, і на їх основі згенерувати детальний план у файл `plan.md`.
- Перевірте, відредагуйте та **затвердьте згенерований план**. Наприклад, ви можете видалити зайві технології або уточнити етапи.

Результат: Затверджений вами покроковий план реалізації у файлі `plan.md`, якого AI буде дотримуватися.

Результат — затверджений покроковий план у `plan.md`, якого далі дотримуємося.

П'ятий крок — документ поточної архітектури

Мета полягає у створенні живого опису, який змінюється разом із кодом. Файл зберігаємо як `docs/architecture.md` і оновлюємо під час кожного значущого рефакторингу або додавання функції.

Дія:

1. Створіть новий файл `docs/architecture.md`.
2. **Поверніться до правила, створеного на Кроці 3, та доповніть його.** Тепер воно повинно виглядати так:
    - `Усі правила архітектури описані в @docs/arch_rules.md`
    - `Усі бізнес-вимоги до сервісу описані в @docs/requirements.md`
    - `Актуальний опис архітектури знаходиться в @docs/architecture.md`

3. Дайте AI одноразову команду, щоб він почав вести цей документ: **"При внесенні змін у проєкт, актуалізуй опис архітектури в документі** `@docs/architecture.md` **та використовуй його в роботі"**

Результат — **Створений та інтегрований у постійний контекст документ architecture.md**, який AI буде самостійно підтримувати.

Шостий крок — поетапна реалізація і код рев'ю

Мета полягає у контрольованій реалізації без стрибків між темами. Ми беремо задачі з `plan.md`, виконуємо їх послідовно і забезпечуємо технічну якість через рев'ю коду.

Робочий ритм

- Даємо AI лише одне завдання за раз з `plan.md`.
- Ретельно перевіряємо згенерований код. Ми маємо бути здатні написати такий самий код самостійно, тоді рев'ю буде змістовним.
- Виправляємо недоліки власноруч або формулюємо звернення до AI на переробку з посиланням на правила.
- За появи системної помилки (наприклад, використання застарілої версії Node.js) оперативно доповнюємо `arch_rules.md`, щоб помилка не повторювалася у наступних ітераціях.

Результат — реалізований поетапно і перевірений код, який відповідає бізнесу та архітектурі і базується на єдиному плані.

Підсумок

Ми маємо чотири опорні документи, які спрямовують роботу системи та команди. Бізнесові вимоги визначають що саме будуємо. Архітектурні правила визначають як саме будуємо. План реалізації встановлює послідовність. Опис поточної архітектури фіксує стан і причини рішень. Далі варто створити початкові версії усіх файлів у проєкті і підключити їх у вашому інструменті роботи з AI, щоб підтримувати спільний контекст під час кожної ітерації.