hw2

Початкові умови

- моноліт
- 20 розробників
- 1 бд (уявимо монгоДБ)
- структура як на картинці
- проект на nodejs

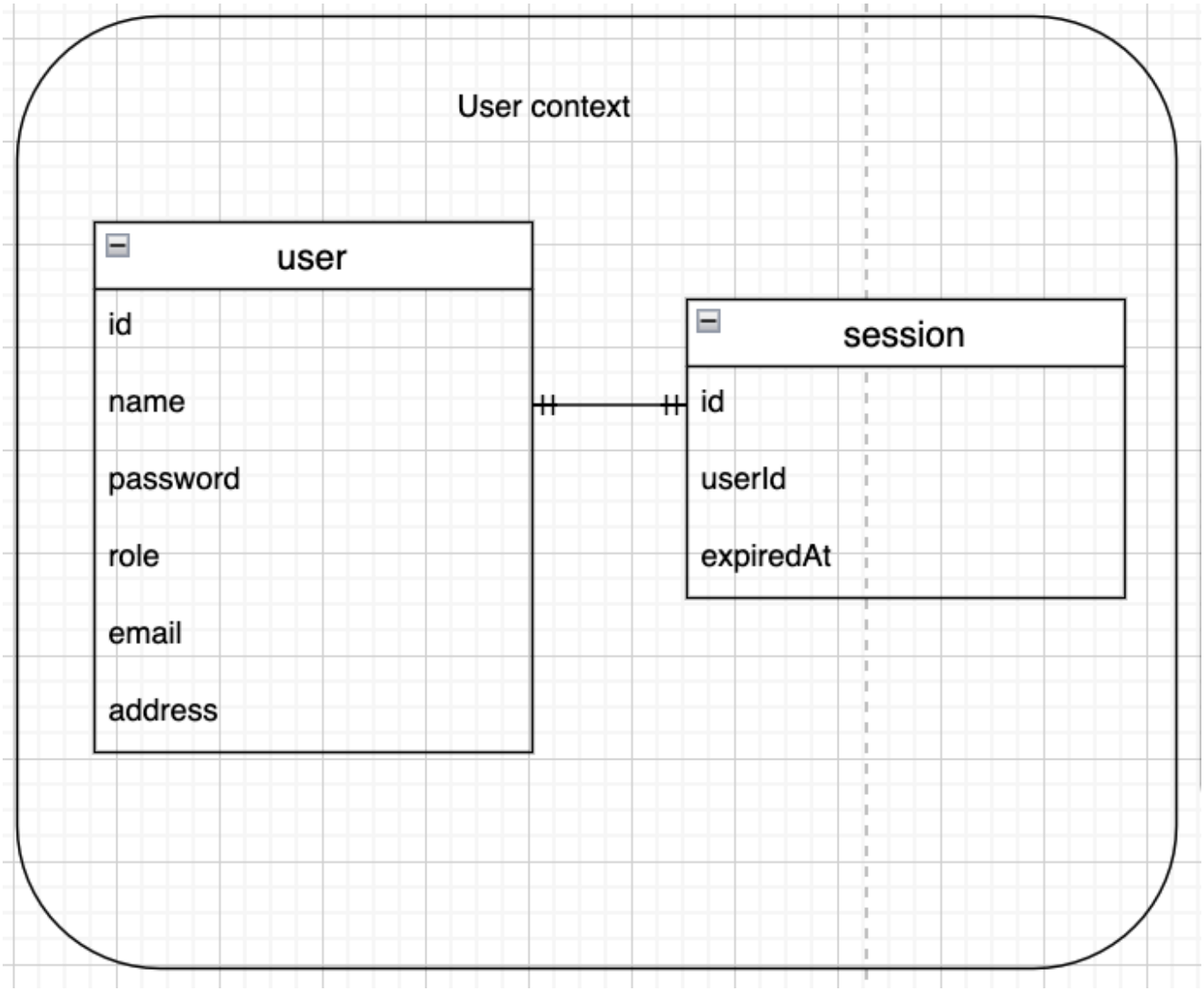
---

1. Першим ділом я би провів підготовку до міграції проекта на мікросервіси саме з людьми:
    - переход на МС проходить паралельно деліверінгу фітч, 20-30% спрінт капасіті виділяється на наш реакторинг. (`Strangler application` паттерн)
    - поділив би команду розробників на 3 підкоманди по 3 людини (кожній підкоманді по 2 мікросервіси для виокремлення)
    - розділення на команди відбулося б на основі експертизи, щоб кожна команда добре розумілася в предметній області того, що она виокремлює
    - в кожній команді повинен бути надійний розробник ака лідер (щоб міг приймати рішення самостійно та направляти тіммейтів)
    - всіх зібрав, провів би мотиваційну бесіду, чому це важно нам і бізнесу. Поставив би ціль, та час до коли то потрібно нам виконати, вислухав би їх побажання.
    - дав би книжки та матеріале для підготовки

---

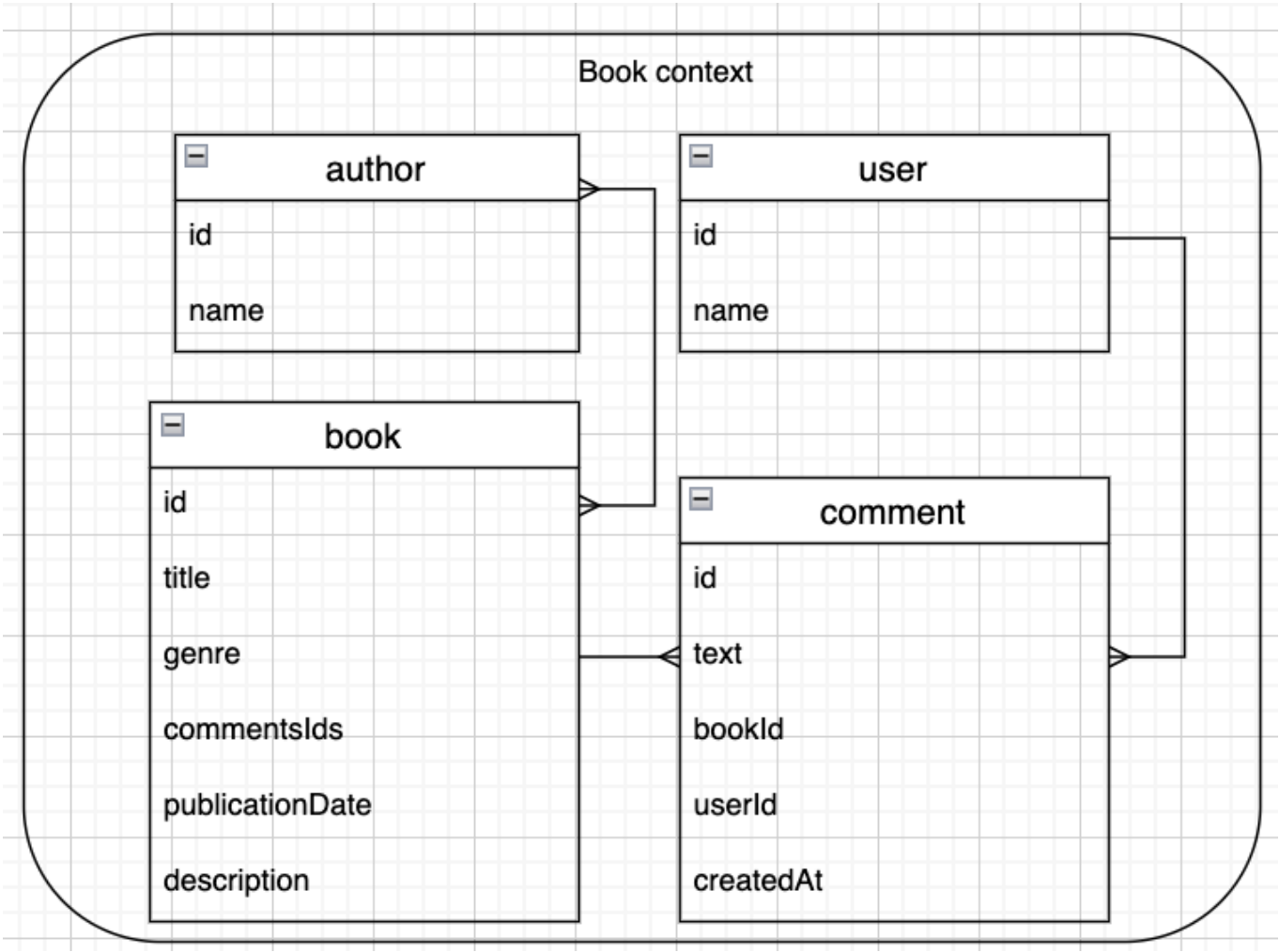
2. Технічна підготовка проекту.
    1. підготовув би шаблон мікросервісу
        - гіт
        - докер
        - конфіги
        - змінні оточення
        - логування
        - залежності

2. Shared components (libraries):
   на основі брейн-штормінгу, евент-сорсінгу, та на розумінні яка в нас бізнес-логіка та компоненти можуть переюзатися - повиносив би у бібліоткеки (умовні npm пакети), такі речі що можна перевикористовувати все ще і в моноліті, так і в майбутніх мікросервісах
3. Імплементування Anti-corruption layer для спілкування майбутніх МС з монолітом
4. Вибір та налаштування брокера (Apache Kafka). Щоб це не стало єдиною точкою відмови, впевнитись та налаштувати все так, щоб досягнути `High Availability` та `Fault tolerance`
5. створення технічної адмінки та БД для подальшого використання для `Feature Flag` підходу при переїзді на МС



3. Перший МС буде `User MS` який в собі буде реалізовувати AuT/AuZ, та source of truth для об'єкту юзера. `User MS` перший - бо його легше всього виокремити, майже на усіх проектах це не дуже велика, та добре виокремлена сутність.
- першим ділом ми тестуємо на тестовому інвайроменті дамп данних та перенос, та

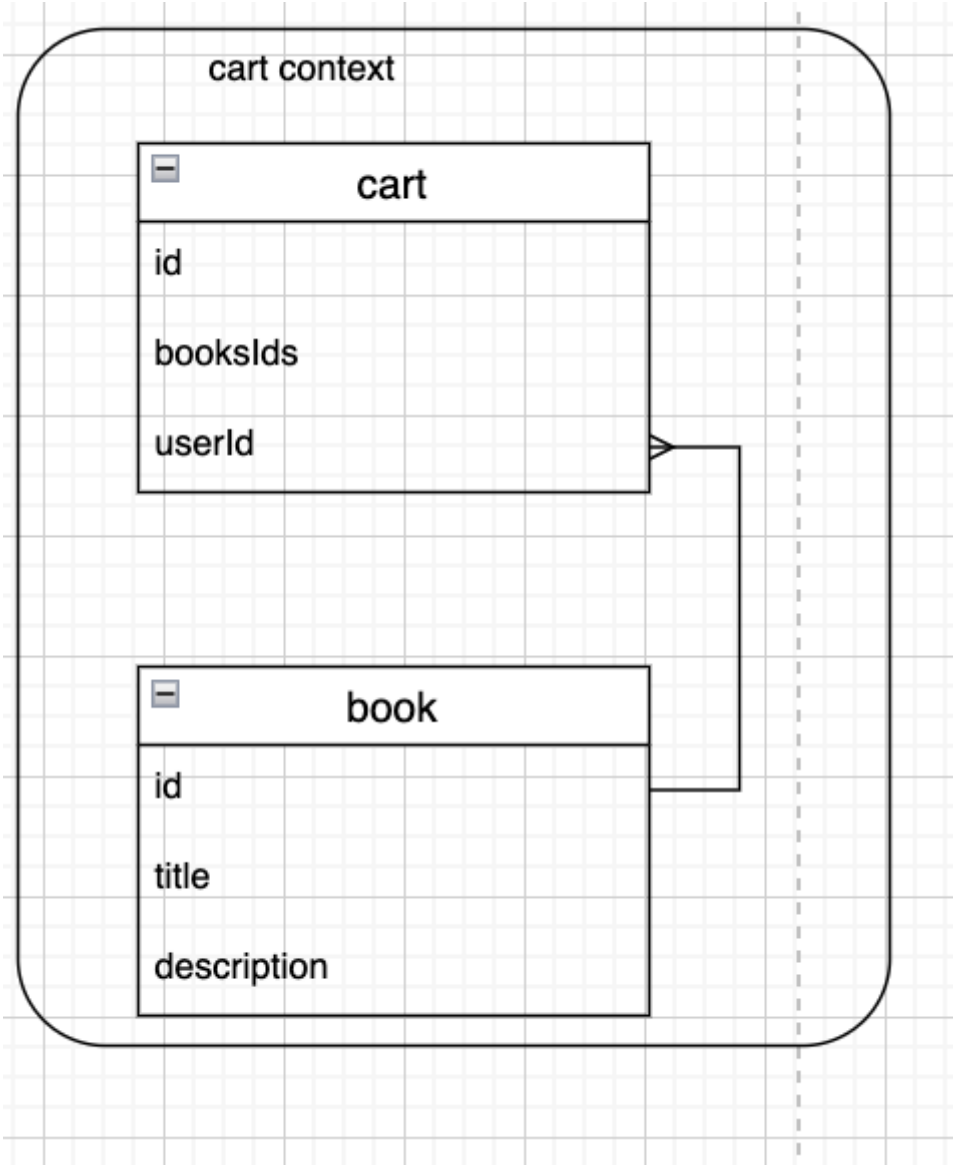
впевнюємось що наш база витримає , бо ми майже завжди маємо окремі коллекції user, session та щось схоже на це, саме тому спочатку дані, потім код.
- коли перенос завершено, ми пишемо МС код,
- коли ми впевнелись що все ок, ми готові до нашого переключення на роботу через МС.
- спілкуємось з монолітом через ACL, викоритовуємо брокер.
- якийсь час можуть використовуватись обидві БД. нові токени, та сессії записуємо в МС, старі використовуємо поки не заекспайряться (та не видаємо більше рефреш токени) потім можемо видаляти зайві колекції.



4. Другий МС `Book MS`. Бо це доволі окрема сутність яка не дуже пов’язана в данних з іншими таблицями. Пошук книжок за автором чи назвою, та комменти, це доволі такі окремі процесси, які не відносяться до корзини, чи оформлення замовлення (які використовують лише id якизось там книжок)
1. тож в цьому випадку я також би виносив спочатку данні
2. виносимо код (також ACL, брокер)
3. налаштовуємо перші саги, по типу Юзер зареєструвався, і до цього сервісу дійшли месседи про нового юзера, чи зміну імені.

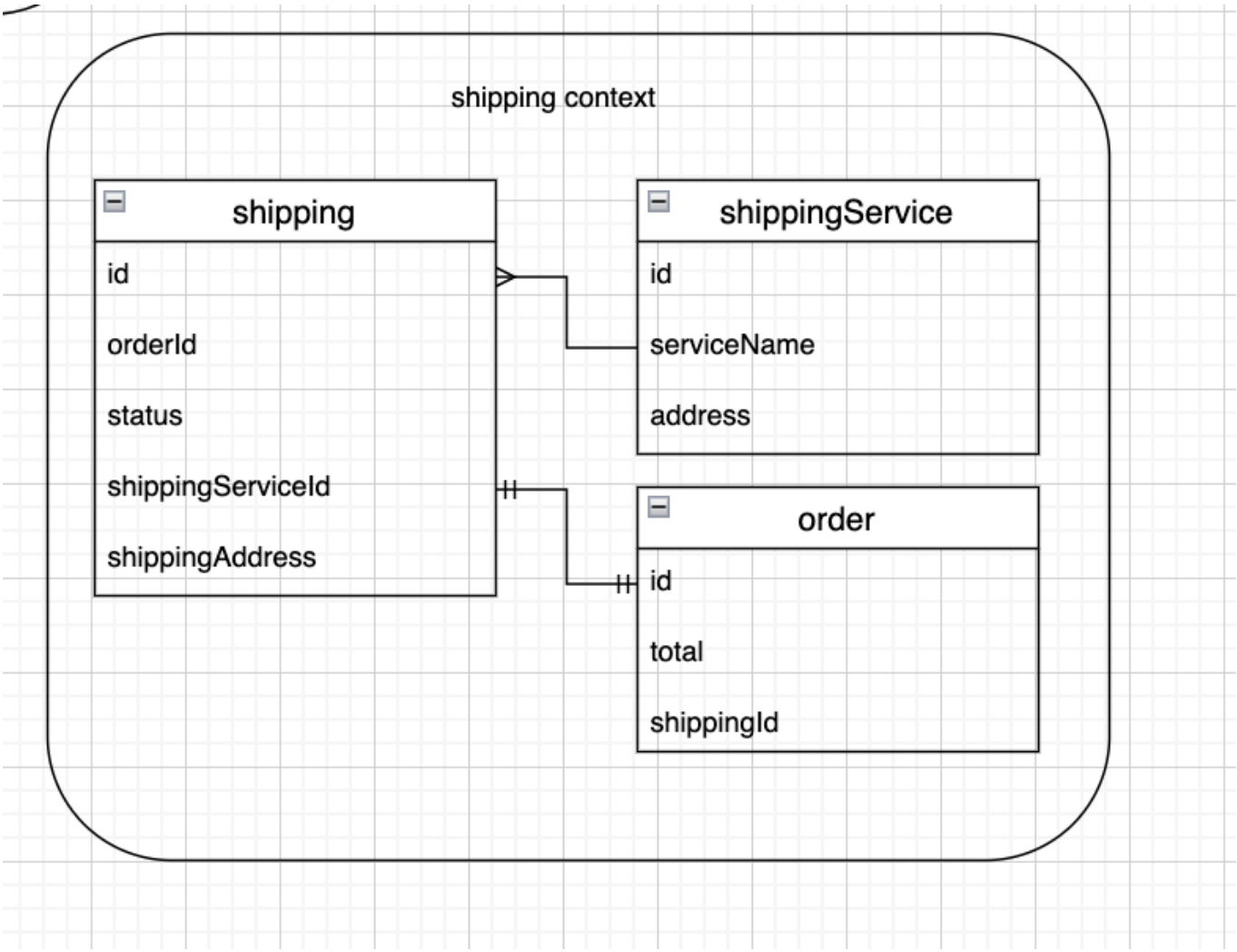
4. За допомогою `Feture Flag` в один момент перемикаємо для всіх юзерів флоу на МС, коли все відтестовано, під час перемикання, буде на декілька міллісекунд недоступний ElasticSearch з продуктами, але це не є критичною частиною аплікухи, тому можливо цим можна знехтувати

по суті в нас залишився МАКРО-сервіс який включає ордери, шиппінг, та карт, тож його можна залишати як таким деякий час, поки триває виокремлення зазначених вище МС.

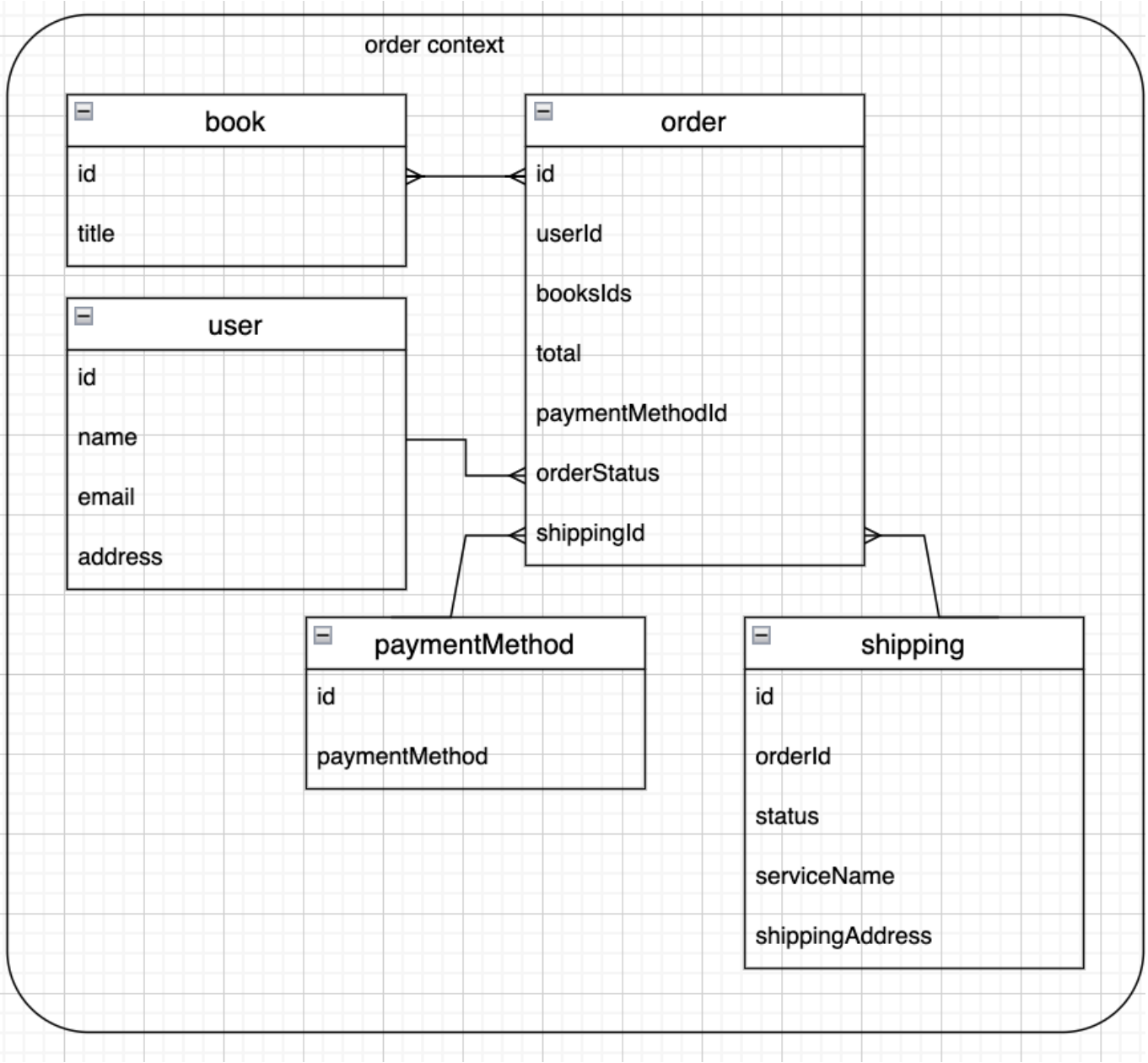


5. Далі виносив би `Cart MS`
1. почав би з кодової бази, бо це критичний вузол всього додатку, тож потрібно пересвідчитись що працює все дуже добре, без багів, саме тому починаємо з коду та використовуємо `Parallel Run`. Працюємо все ще на моноліті, але декілька днів слідкуємо за логами з МС, щоб пересвідчитись що кодова база в нормі.
2. переїзд БД дуже важливий також, бо не повинні бути втрачені жодні корзини та можливі покупки. Для кожної корзини в нас вже імплементовано інкрементне версіонування . Максимально перестраховуючись в коді робимо перемички які потім повирубаємо за допомогою `Feature Flag` які дозволять нові корзини створювати на

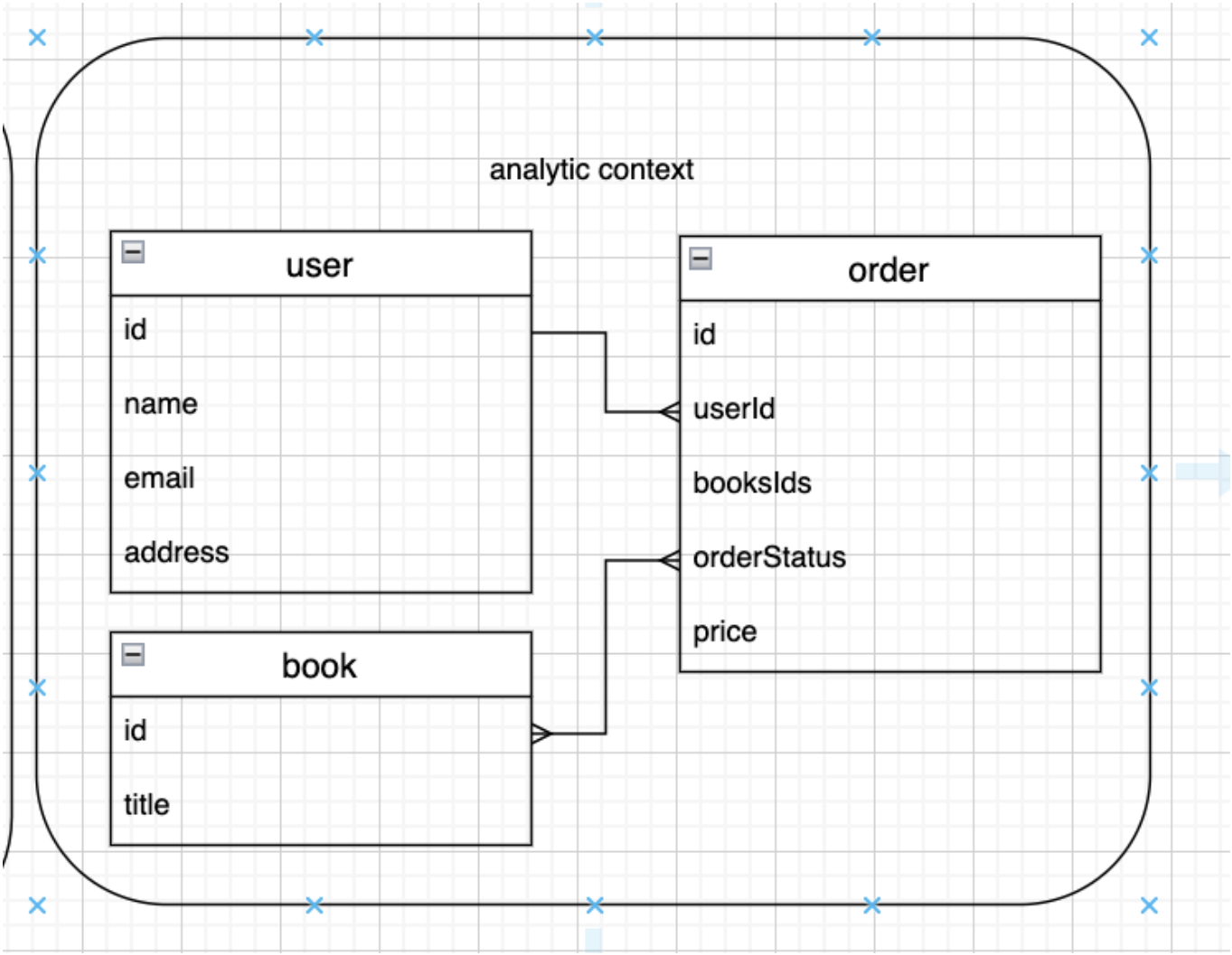
лише МС, почати перелив історичних даних до МС, після переливу лочимо оновлення історичних даних в моноліті, і якщо під час переливу, вони будуть оновлюватись, то ми їх додамо до МС із збільшеною версією версії корзини (якщо під час переливки версію корзини з моноліту менша за ту зо в МС, то не оновлюємо)
3. далі вирубаємо фічею взагалі монолітну частну та випілюємо



6. Shipping MS. Його легше виокремити. Він залежить лише від Ордеру. Але він менший ордеру, тож він Shipping MS буде перший.
1. спочатку кодова база
2. перевіряється працездатність
3. вона тестується
4. `Parallel run`
5. БД виноситься.
6. вирубаємо з моноліту



7. Order MS. це буде найжирніший та найскладніший МС.
1. тож починав я би також з кодової бази тут.
2. тестування `Parallel run`
3. тестування e2e.
4. гарне тестування спілкування з іншими МС, від яких залежить робота ордера
5. пренос БД



8. Analytic MS. він останній бо є наменш значушім, від нбого не залежить працезданість жодного МС. він лише в режимі рід слідкує за оновленнями деяких колонок з декількох таблиць, та агрегує данні для аналітики.

1. наповнюємо БД МС,
2. оновлюємо БД МС
3. Виносимо кодову базу до МС
4. прогоняємо тести,
5. `Parallel run`