11. Vẽ và giải thích góc nhìn logic và góc nhìn process của thiết kế Event Sourcing được đề xuất trong bài lab 03? Giải thích các công cụ sử dụng và từng bước cần thực hiện để viết mã nguồn cho một tính năng ghi dữ liệu vào hệ thống, và một tính năng xuất báo cáo từ dữ liệu của hệ thống, theo góc nhìn logic và process đề xuất.

**Trả lời**

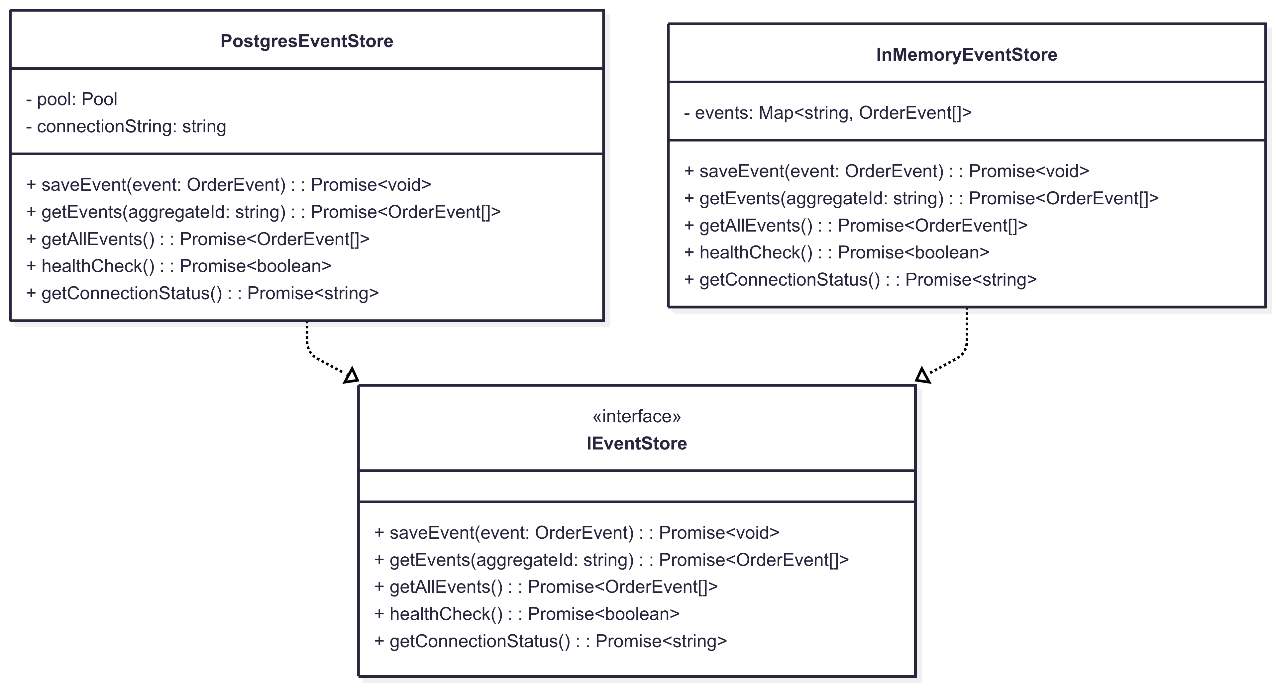
1. **Góc nhìn Logic**

A diagram of a process

AI-generated content may be incorrect.

**A diagram of a diagram

AI-generated content may be incorrect.**

****

**A diagram of a computer

AI-generated content may be incorrect.**

**1) Bức tranh tổng thể theo lớp (layers)**

**1. Frontend (React/Next.js)**

* **OrderPage / OrderForm**: thu thập input, hiển thị kết quả.
* **useOrders hook + ApiClient**: đóng gói gọi API (POST/GET), xử lý lỗi, loading.  
  **Trách nhiệm**: chỉ trình bày (presentation) và phát lệnh/đặt câu hỏi ra backend; **không chứa logic nghiệp vụ**.

**2. Presentation Layer (API)**

* **Routes/Middleware**: định tuyến, auth, validation, rate limit, idempotency key (nếu cần).
* **OrderController**: chuyển HTTP request → **Command** (ghi) hoặc **Query** (đọc), chuẩn hóa DTO và trả HTTP response.  
  **Trách nhiệm**: “dịch” giữa thế giới HTTP và ứng dụng; **không quyết định nghiệp vụ**.

**3. Application Layer**

* **CommandHandler**: xử lý các lệnh thay đổi trạng thái (*Create, Cancel, AddItem, …*).
  + Tải trạng thái **Order Aggregate** (replay từ Event Store + snapshot).
  + Gọi phương thức domain (enforce invariant).
  + Nhận **Domain Events** và yêu cầu lưu xuống Event Store.
* **QueryHandler**: phục vụ API đọc; lấy dữ liệu từ **ReadModelRepository** (đã được denormalize/projection).  
  **Trách nhiệm**: điều phối ca nghiệp vụ (use case orchestration). **Không lưu dữ liệu trực tiếp**—chỉ gọi port.

**4. Domain Layer**

* **Order Aggregate**: mô hình hoá luật nghiệp vụ (vd: không huỷ đơn đã shipped, tính tổng tiền, …).
* **OrderItem, OrderStatus** (value objects/enums).
* **Domain Events** (*OrderCreated, ItemAdded, OrderCanceled, …*): kết quả bất biến của hành động.  
  **Trách nhiệm**: **trái tim nghiệp vụ**. Chỉ sinh events; **không biết DB là gì**.

**5. Infrastructure Layer**

* **PostgresEventStore**: port/adapter ghi–đọc **events** (append-only), đảm bảo version, idempotency.
* **SnapshotService**: định kỳ lưu **snapshot** để giảm chi phí replay.
* **KafkaProducer (tuỳ chọn)**: phát tán events ra bus để dịch vụ khác/subscribers tiêu thụ.
* **ProjectionUpdater**: subscribe event stream (trực tiếp hoặc qua Kafka) để **cập nhật Read Models**.
* **ReadModelRepository**: truy vấn **PostgreSQL (read models)** cho QueryHandler.

**Trách nhiệm**: hiện thực các **adapter** nói chuyện với công nghệ (DB, message bus).

**6. PostgreSQL (DB vật lý)**

* Lưu **events**, **snapshots**, và **read models** (projections).

**Trách nhiệm**: bền vững dữ liệu; tách **write model (event log)** và **read model**.

**2) Luồng LOGIC chính (dạng “hợp đồng” giữa các thành phần)**

**A. Luồng Command (ghi: Create/Cancel/AddItem…)**

1. **Frontend** gửi POST → **OrderController**.
2. **OrderController** tạo **CreateOrderCommand** → **CommandHandler**.
3. **CommandHandler** nạp **Aggregate** (EventStore + Snapshot), gọi order.create(...).
4. **Aggregate** kiểm tra invariants, **phát sinh Domain Events** (vd. OrderCreated).
5. **CommandHandler** yêu cầu **EventStore.append(events)**; (tùy chọn) **KafkaProducer.publish(events)**.
6. **ProjectionUpdater** nhận events → cập nhật **Read Models**.
7. **Controller** trả **201** + orderId.

**Giá trị logic**:

* Tính **auditability** (events bất biến), **integrity** (version), **recoverability** (replay/snapshot), **scalability** (read/write tách rời).

**B. Luồng Query (đọc: GetOrder, Export Report…)**

1. **Frontend** gọi GET → **OrderController/ReportController**.
2. **Controller** → **QueryHandler**.
3. **QueryHandler** → **ReadModelRepository** lấy dữ liệu từ **Read DB** (đã tối ưu).
4. (Nếu là report) **ReportService** đóng gói CSV/PDF.
5. **Controller** trả về dữ liệu/file.

**Giá trị logic**:

* **Hiệu năng đọc cao** nhờ projection; **eventual consistency** được chấp nhận theo thiết kế CQRS.

**3) Biên giới & phụ thuộc (rất quan trọng ở Logical View)**

* **UI ↔ API**: hợp đồng là **HTTP + DTO** (Request/Response).
* **API ↔ Application**: hợp đồng là **Command/Query DTO** (loại ca nghiệp vụ).
* **Application ↔ Domain**: hợp đồng là **method/domain contract** của Aggregate (đổi lệnh thành events).
* **Application ↔ Infrastructure**: qua **ports** (interfaces) như EventStore, SnapshotRepo, ReadModelRepo.
  + EventStore.append(streamId, eventType, payload, expectedVersion?)
  + EventStore.load(streamId) -> Event[]
  + ReadModelRepository.getOrderView(orderId) -> OrderView
* **Domain** không phụ thuộc hạ tầng; **hạ tầng** hiện thực interfaces do **application/domain** định nghĩa (Dependency Inversion).

**4) Các quyết định thiết kế thể hiện trên sơ đồ (và “vì sao”)**

* **Event Store append-only + version** → đảm bảo **tính bất biến** & **traceability**.
* **SnapshotService** → giảm độ trễ replay cho aggregate lớn.
* **ProjectionUpdater tách rời** → **scale** đọc/ghi độc lập, thêm projection mới **không ảnh hưởng** dữ liệu cũ (schema evolution).
* **KafkaProducer (tùy chọn)** → mở rộng tích hợp liên dịch vụ; nếu chưa cần, ProjectionUpdater có thể đọc trực tiếp từ EventStore.
* **ReadModelRepository** riêng → cô lập truy vấn phức tạp/denormalize, tránh rò rỉ SQL vào application.

**5) Cách dùng sơ đồ để “dẫn đường” viết mã**

* Khi **thêm tính năng ghi**:
  1. Thêm **Command + DTO** (Presentation ↔ Application).
  2. Mở rộng **Aggregate** + **Domain Event** tương ứng (Domain).
  3. Gọi **EventStore.append** (Infrastructure).
  4. Viết handler cho **ProjectionUpdater** để update **Read Model** (Infrastructure).
  5. Bổ sung **route/controller** (Presentation) + **UI** (Frontend).
* Khi **thêm tính năng đọc/báo cáo**:
  1. Thiết kế **Read Model**/view phù hợp (Infrastructure/DB).
  2. Viết **ReadModelRepository** query → **QueryHandler** (Application).
  3. (Nếu xuất file) thêm **ReportService** đóng gói CSV/PDF (Infrastructure).
  4. Thêm **endpoint** và tích hợp **UI**.

1. **Góc nhìn Process**

A screenshot of a computer

AI-generated content may be incorrect.

Sơ đồ này minh họa luồng hoạt động khi một người dùng tạo một đơn hàng trong một hệ thống sử dụng kiến trúc Event Sourcing và CQRS.

**Các thành phần tham gia:**

* **User**: Người dùng cuối tương tác với hệ thống.
* **Frontend**: Giao diện người dùng, nơi người dùng điền thông tin và tương tác với ứng dụng.
* **API Controller (controller.ts)**: Tầng API của backend, nhận các yêu cầu HTTP từ Frontend và điều phối chúng.
* **Application Layer (Command Handler)**: Tầng ứng dụng chịu trách nhiệm xử lý các lệnh (Command), điều phối logic nghiệp vụ và tương tác với Domain Layer.
* **Domain Layer (Order Aggregate)**: Lõi nghiệp vụ của hệ thống, nơi định nghĩa các thực thể và quy tắc nghiệp vụ (Order Aggregate là một tập hợp các đối tượng liên quan được coi là một đơn vị).
* **Infrastructure Layer (Event Store)**: Tầng cơ sở hạ tầng chịu trách nhiệm lưu trữ và truy xuất các sự kiện.
* **PostgreSQL DB**: Cơ sở dữ liệu vật lý được sử dụng để lưu trữ các sự kiện.

**Luồng hoạt động "Tạo đơn hàng":**

1. **Người dùng điền thông tin đơn hàng**: Người dùng nhập các chi tiết cần thiết cho đơn hàng trên giao diện Frontend.
2. **Frontend gửi yêu cầu POST**: Frontend gửi một yêu cầu HTTP POST đến API Controller tại /orders với dữ liệu đơn hàng (orderData).
3. **API Controller gọi Command Handler**: API Controller nhận yêu cầu và gọi phương thức handleCreateOrder() của Application Layer (Command Handler), truyền vào một createOrderCommand.
4. **Command Handler tạo/cập nhật Order Aggregate**: Command Handler yêu cầu Domain Layer (Order Aggregate) xử lý lệnh. Trong quá trình này, Order Aggregate áp dụng lệnh và sinh ra một sự kiện (Event), cụ thể là OrderCreatedEvent.
5. **Order Aggregate trả về Event**: Order Aggregate trả lại OrderCreatedEvent cho Command Handler.
6. **Command Handler lưu Event vào Event Store**: Command Handler gọi phương thức appendEvent() của Infrastructure Layer (Event Store) để lưu OrderCreatedEvent.
7. **Event Store ghi Event vào PostgreSQL**: Event Store thực hiện một lệnh INSERT INTO events trong cơ sở dữ liệu PostgreSQL để lưu OrderCreatedEvent.
8. **PostgreSQL xác nhận lưu Event**: PostgreSQL xác nhận rằng sự kiện đã được lưu thành công.
9. **Event Store trả về xác nhận**: Event Store trả lại thông báo "Event đã lưu thành công" cho Command Handler.
10. **Command Handler trả về kết quả thành công**: Command Handler thông báo cho API Controller rằng lệnh đã được xử lý thành công.
11. **API Controller trả về phản hồi**: API Controller gửi phản hồi HTTP "201 Created" cùng với OrderCreatedEvent về Frontend.
12. **Frontend hiển thị đơn hàng đã tạo**: Frontend nhận phản hồi và hiển thị thông tin về đơn hàng mới tạo cho người dùng.
13. **Các công cụ sử dụng**

* **Runtime & ngôn ngữ**: Node.js + TypeScript
* **Presentation (API)**: Fastify (nhẹ, nhanh) hoặc NestJS (có DI mạnh)
* **Validation/DTO**: zod hoặc class-validator
* **Persistence (Event Store + Read Models)**: PostgreSQL (pg hoặc Prisma/Drizzle cho Read Model)
* **Event Store**: bảng append-only + optimistic concurrency (expectedVersion)
* **Message Bus (tuỳ chọn)**: Kafka (kafkajs) cho phát tán events
* **Projection/Read Model**: tiến trình ProjectionUpdater (consumer) cập nhật order\_projections, order\_items
* **Report**: CSV (csv-stringify/fast-csv), PDF (pdfkit) hoặc HTML→PDF (Puppeteer)
* **Idempotency**: middleware đọc Idempotency-Key + bảng log idempotency
* **Testing**: Jest + supertest (API), test aggregate theo “given-when-then” (event sourcing)

1. **Các bước thực hiện**
   1. **Tính năng ghi dữ liệu vào hệ thống – Create Order**

**Bước 1: Frontend - Tạo Form và nút "Tạo đơn hàng"**

1. **Mục tiêu**: Cung cấp giao diện cho người dùng nhập thông tin đơn hàng và gửi đi.
2. **File cần sửa**: frontend/components/ui/[order-management-clean.tsx](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html).
3. **Thực hiện**:
   * Tạo một form với các trường nhập liệu cho customerId và danh sách items (sản phẩm, số lượng, giá).
   * Thêm một nút "Create Order".
   * Viết hàm xử lý sự kiện **onSubmit** cho form. Hàm này sẽ thu thập dữ liệu từ form, đóng gói thành một object **orderData**, và gọi phương thức **createOrder từ ApiClient**.

// Trong component OrderManagementClean

const handleCreateOrder = async (event) => {

  event.preventDefault();

  // Lấy dữ liệu từ state của form

  const newOrderData = { customerId, items };

  try {

    await apiClient.createOrder(newOrderData);

    // Tải lại danh sách đơn hàng hoặc hiển thị thông báo thành công

    refetchOrders();

  } catch (error) {

    console.error("Failed to create order:", error);

  }

};

**Bước 2: Frontend - Cập nhật API Client**

1. **Mục tiêu**: Gửi yêu cầu tạo đơn hàng mới đến backend.
2. **File cần sửa**: [api-client.ts](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html).
3. **Thực hiện**:
   * Tạo một phương thức mới là **createOrder**(orderData: NewOrderData).
   * Phương thức này sẽ gửi một yêu cầu POST đến endpoint **/api/orders** của backend, với **orderData** được gửi trong body của request.

// Trong class ApiClient

async createOrder(orderData: NewOrderData): Promise<void> {

  await this.axiosInstance.post('/orders', orderData);

}

**Bước 3: Presentation Layer (Backend) - Route và Controller Method**

1. **Mục tiêu**: Nhận yêu cầu HTTP, xác thực đầu vào, và chuyển nó thành một Command để Application Layer xử lý.
2. **File cần sửa**: Order-management/src/api/routes.ts và Order-management/src/api/controller.ts.
3. **Thực hiện**:
   * Trong routes.ts, đảm bảo route POST **/orders** đã được định nghĩa và trỏ đến controller.

// Đã có sẵn trong file routes.ts

router.post('/orders', (req, res) => this.controller.createOrder(req, res));

* + Trong controller.ts, phương thức **createOrder** sẽ nhận request, tạo **CreateOrderCommand**, và gọi **CommandHandler**.

// Trong class OrderController

async createOrder(req: Request, res: Response) {

  try {

    // Validation có thể được thêm ở đây hoặc trong middleware

    const { customerId, items } = req.body;

    const command = { customerId, items };

    const orderId = await this.commandHandlers.handleCreateOrder(command);

    res.status(201).json({ success: true, data: { orderId } });

  } catch (error) {

    res.status(400).json({ success: false, error: error.message });

  }

}

**Bước 4: Domain Layer - Định nghĩa Logic nghiệp vụ trong Aggregate**

1. **Mục tiêu**: Mô hình hóa các quy tắc nghiệp vụ cốt lõi khi một đơn hàng được tạo.
2. **File cần sửa**: Order-management/src/domain/Order.ts.
3. **Thực hiện**:
   * Logic tạo đơn hàng nằm ngay trong constructor của class **Order**.
   * constructor sẽ nhận customerId và items, sau đó:
     + Kiểm tra các *invariants* (bất biến): ví dụ, danh sách items không được rỗng.
     + Tự động gán một id (UUID) duy nhất.
     + Tính toán tổng số tiền (totalAmount) từ danh sách items.
     + Thiết lập trạng thái ban đầu là PENDING.

// Trong class Order

constructor(customerId: string, items: OrderItem[], status: string = 'PENDING', id?: string) {

  if (!items || items.length === 0) {

    throw new Error('Order must have at least one item.');

  }

  this.id = id || uuidv4();

  this.customerId = customerId;

  this.items = items;

  this.status = status;

  this.totalAmount = this.calculateTotal(); // Phương thức private để tính tổng tiền

}

**Bước 5: Application Layer - Xử lý Command**

1. **Mục tiêu**: Điều phối use case "Tạo đơn hàng": tạo aggregate, sinh ra event, và yêu cầu lưu trữ.
2. **File cần sửa**: Order-management/src/queries/handlers.ts.
3. **Thực hiện**:
   * Phương thức **handleCreateOrder** nhận command từ Controller.
   * **Tạo Aggregate**: Gọi new Order(...) để tạo một instance mới của Order aggregate. Logic domain trong constructor sẽ được thực thi.
   * **Tạo Event**: Dựa trên Order object vừa tạo, sinh ra một **OrderCreatedEvent**. Đây là "sự thật" bất biến về những gì đã xảy ra.
   * **Lưu Event**: Gọi **this.eventStore.saveEvent()** để lưu sự kiện này.

// Trong class OrderCommandHandlers

async handleCreateOrder(command: { customerId: string; items: OrderItem[] }): Promise<string> {

  // 1. Tạo Aggregate, logic domain được thực thi trong constructor

  const order = new Order(command.customerId, command.items);

  // 2. Tạo Event từ trạng thái của aggregate

  const event: OrderEvent = {

    type: 'OrderCreated',

    aggregateId: order.id,

    timestamp: new Date(),

    data: {

      orderId: order.id,

      customerId: order.customerId,

      items: order.items,

      status: order.status,

      totalAmount: order.totalAmount,

    },

  };

  // 3. Yêu cầu Infrastructure Layer lưu Event

  await this.eventStore.saveEvent(event);

  return order.id;

}

**Bước 6: Infrastructure Layer - Lưu Event vào Event Store**

1. **Mục tiêu**: Ghi **OrderCreatedEvent** vào PostgreSQL một cách an toàn và nhất quán.
2. **File cần sửa**: Order-management/src/infrastructure/[postgres-event-store.ts](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html).
3. **Thực hiện**:
   * Phương thức **saveEvent** sẽ được gọi.
   * Nó sử dụng Connection Pooling để có kết nối hiệu quả.
   * Nó gọi hàm **append\_event** trong PostgreSQL, hàm này sẽ xử lý việc tăng version và ghi event vào bảng events trong một TRANSACTION để đảm bảo tính nguyên tử.

**Bước 7: Infrastructure Layer - Cập nhật Read Model (Projection)**

1. **Mục tiêu**: Ngay sau khi đơn hàng được tạo, cập nhật bảng order\_summaries để các truy vấn đọc có thể thấy ngay đơn hàng mới.
2. **File cần sửa**: Order-management/src/infrastructure/projection-updater.ts (giả định).
3. **Thực hiện**:
   * Bộ xử lý sự kiện sẽ lắng nghe **OrderCreatedEvent**.
   * Khi nhận được event, nó sẽ thực hiện một câu lệnh INSERT vào bảng **order\_summaries** với dữ liệu từ event.

-- Câu lệnh SQL mà Projection Updater sẽ chạy

INSERT INTO order\_summaries (order\_id, customer\_id, status, total\_amount, created\_at)

VALUES ($1, $2, $3, $4, $5);

* 1. **Tính năng xuất báo cáo dữ liệu của hệ thống** - **Export Orders Report**

**Bước 1: Infrastructure Layer - Thiết kế Read Model**

1. **Mục tiêu**: Tạo một bảng (hoặc view) được tối ưu cho việc truy vấn báo cáo.
2. **File cần sửa**: : Order-management/database/schema.sql.
3. **Thực hiện**:
   * Thiết kế một bảng **daily\_sales\_summary** để lưu tổng doanh thu mỗi ngày.

CREATE TABLE daily\_sales\_summary (

    sale\_date DATE PRIMARY KEY,

    total\_revenue NUMERIC(12, 2) NOT NULL,

    order\_count INTEGER NOT NULL

);

* + Cập nhật **ProjectionUpdater** để xử lý **OrderCreatedEvent**và **OrderCancelledEvent**, sau đó INSERT hoặc UPDATE vào bảng **daily\_sales\_summary**.

**Bước 2: Infrastructure Layer - Tạo Read Model Repository**

1. **Mục tiêu**: Đóng gói logic truy vấn vào Read Model.
2. **File cần tạo**: Order-management/src/infrastructure/read-model-repository.ts.
3. **Thực hiện**:
   * Tạo class **ReadModelRepository.**
   * Viết phương thức **getMonthlySalesReport**(month: number, year: number) để truy vấn bảng **daily\_sales\_summary**.

// ...

const result = await this.pool.query(

  'SELECT sale\_date, total\_revenue FROM daily\_sales\_summary WHERE EXTRACT(MONTH FROM sale\_date) = $1 AND EXTRACT(YEAR FROM sale\_date) = $2 ORDER BY sale\_date',

  [month, year]

);

return result.rows;

**Bước 3: Application Layer - Tạo Query Handler**

1. **Mục tiêu**: Điều phối việc lấy dữ liệu báo cáo.
2. **File cần tạo**: Order-management/src/queries/handlers.ts.
3. **Thực hiện**:
   * Tạo class **OrderQueryHandlers.**
   * Viết phương thức **handleGetMonthlySalesReport**(query: { month: number, year: number }).
   * Phương thức này gọi **readModelRepository.getMonthlySalesReport()** và trả về dữ liệu.

**Bước 4: Infrastructure Layer - Tạo Report Service**

1. **Mục tiêu**: Chuyển đổi dữ liệu từ DB thành định dạng file CSV.
2. **File cần tạo**: Order-management/src/infrastructure/report-service.ts.
3. **Thực hiện**:
   * Cài đặt thư viện **fast-csv**: npm install fast-csv.
   * Tạo class **ReportService** với phương thức **generateSalesReportCsv**(data: any[]): Promise<string>.
   * Sử dụng **fast-csv** để chuyển mảng JSON (dữ liệu từ repository) thành một chuỗi CSV.

**Bước 5: Presentation Layer (Backend) - Tạo Endpoint xuất báo cáo**

1. **Mục tiêu**: Cung cấp một endpoint để người dùng tải file báo cáo.
2. **File cần sửa**: routes.ts và controller.ts.
3. **Thực hiện**:
   * Trong routes.ts, thêm route mới:

router.get('/reports/monthly-sales', (req, res) => this.controller.getMonthlySalesReport(req, res));

* + Trong controller.ts, tạo phương thức **getMonthlySalesReport**:

async getMonthlySalesReport(req: Request, res: Response) {

  try {

    const { month, year } = req.query; // Lấy tháng, năm từ query params

    const data = await this.queryHandlers.handleGetMonthlySalesReport({ month, year });

    const csv = await this.reportService.generateSalesReportCsv(data);

    res.header('Content-Type', 'text/csv');

    res.attachment(`sales-report-${year}-${month}.csv`);

    res.send(csv);

  } catch (error) {

    res.status(500).json({ success: false, error: 'Failed to generate report' });

  }

}

**Bước 6: Frontend - Thêm chức năng tải báo cáo**

1. **Mục tiêu**: Cho phép người dùng chọn tháng/năm và nhấn nút để tải file.
2. **File cần sửa**: frontend/components/ui/[order-management-clean.tsx](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (hoặc một trang báo cáo riêng).
3. **Thực hiện**:
   * Thêm các ô input để người dùng chọn tháng và năm.
   * Thêm nút "Export Report".
   * Khi nhấn nút, tạo một thẻ <a> trỏ đến URL của endpoint báo cáo (ví dụ: /api/reports/monthly-sales?year=2025&month=8), rồi tự động click vào thẻ đó để trình duyệt tải file về.

**Giai đoạn 0: Khởi tạo nền tảng (The "Chưa có gì" State)**

Trước khi viết tính năng, chúng ta cần dựng lên bộ khung của hệ thống.

1. **Tạo cấu trúc thư mục:**
   * Tạo thư mục gốc, ví dụ event-sourcing-system.
   * Bên trong, tạo hai thư mục con: [frontend](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) và [order-management](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (backend).
2. **Khởi tạo Project:**
   * **Backend (**[order-management](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)**):**
     + Chạy npm init -y.
     + Cài đặt các thư viện cần thiết: npm install express typescript ts-node-dev pg @types/express @types/pg uuid @types/uuid.
     + Tạo file tsconfig.json (npx tsc --init).
     + Tạo cấu trúc thư mục con bên trong src: api, application, domain, infrastructure.
   * **Frontend:**
     + Chạy pnpx create-next-app@latest frontend để tạo một dự án Next.js với TypeScript.
3. **Thiết lập Database ban đầu:**
   * Tạo file [schema.sql](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html).
   * Viết lệnh CREATE TABLE events (...) trong file này. Đây là "sự thật" duy nhất của hệ thống ở phía ghi (write-side).

-- order-management/database/schema.sql

CREATE TABLE events (

    id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

    aggregate\_id VARCHAR(255) NOT NULL,

    event\_type VARCHAR(100) NOT NULL,

    event\_data JSONB NOT NULL,

    version INTEGER NOT NULL,

    timestamp TIMESTAMPTZ DEFAULT current\_timestamp,

    UNIQUE(aggregate\_id, version)

);

**Tính năng 1: Tạo Order (Write Flow - Xây dựng từ trong ra ngoài)**

Chúng ta sẽ xây dựng từ lõi nghiệp vụ (Domain) ra đến giao diện (UI).

**Bước 1: Domain Layer - Định nghĩa "Order" là gì**

1. **Mục tiêu**: Mô hình hóa các quy tắc và dữ liệu cốt lõi của một đơn hàng.
2. **File cần tạo**:
   * [Order.ts](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
   * [types.ts](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
3. **Thực hiện**:
   * Trong types.ts, định nghĩa cấu trúc của một OrderEvent và sự kiện cụ thể OrderCreated.
   * Trong Order.ts, tạo class Order. Viết constructor để nhận customerId và items, kiểm tra các quy tắc (ví dụ: items không được rỗng), tự gán id, và tính totalAmount.

**Bước 2: Infrastructure Layer - Cách lưu trữ Event**

1. **Mục tiêu**: Hiện thực hóa việc ghi Event xuống PostgreSQL.
2. **File cần tạo/sửa**:
   * [event-store.ts](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (Interface - Port)
   * [postgres-event-store.ts](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (Implementation - Adapter)
   * [schema.sql](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
3. **Thực hiện**:
   * Trong event-store.ts, định nghĩa interface IEventStore với hai phương thức: saveEvent(event) và getEvents(aggregateId).
   * Trong schema.sql, thêm FUNCTION append\_event(...) để xử lý việc ghi event và tăng version một cách an toàn (atomic).
   * Trong postgres-event-store.ts, tạo class PostgresEventStore implement IEventStore. Viết logic kết nối DB (connection pool) và gọi hàm append\_event trong phương thức saveEvent.

**Bước 3: Application Layer - Điều phối nghiệp vụ**

1. **Mục tiêu**: Kết nối Domain và Infrastructure để thực hiện một use case.
2. **File cần tạo**: order-management/src/application/command-handlers.ts (đổi tên từ commands/handlers.ts để rõ ràng hơn).
3. **Thực hiện**:
   * Tạo class OrderCommandHandlers.
   * Viết phương thức handleCreateOrder(command). Bên trong, nó sẽ:
     1. Tạo một Order aggregate mới (new Order(...)).
     2. Tạo một OrderCreatedEvent từ Order aggregate đó.
     3. Gọi this.eventStore.saveEvent(...) để lưu event.

**Bước 4: Presentation Layer - Mở cổng API**

1. **Mục tiêu**: Cung cấp một endpoint HTTP để bên ngoài có thể gọi vào.
2. **File cần tạo**:
   * [controller.ts](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
   * [routes.ts](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
   * [index.ts](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (Entry point của server)
3. **Thực hiện**:
   * Trong index.ts, khởi tạo Express server, áp dụng các middleware cần thiết (như express.json()) và gắn các routes.
   * Trong routes.ts, định nghĩa route POST /api/orders và trỏ nó đến một phương thức trong controller.
   * Trong controller.ts, viết phương thức createOrder(req, res) để nhận request, tạo command, gọi commandHandler, và trả về response 201 Created.

**Bước 5: Frontend - Xây dựng giao diện người dùng**

1. **Mục tiêu**: Tạo form cho người dùng nhập liệu.
2. **File cần tạo/sửa**:
   * [api-client.ts](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
   * [page.tsx](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (hoặc một component riêng)
3. **Thực hiện**:
   * Trong api-client.ts, viết hàm createOrder để gọi API POST /api/orders.
   * Trong page.tsx, dựng một form đơn giản với các trường cần thiết và một nút "Create".
   * Viết hàm xử lý onSubmit để gọi apiClient.createOrder.

**Tính năng 2: Xuất Báo cáo (Read Flow - Xây dựng từ DB ra ngoài)**

Bây giờ chúng ta xây dựng phía đọc (Query), độc lập với phía ghi.

**Bước 1: Infrastructure Layer - Tạo Read Model và Projection**

1. **Mục tiêu**: Tạo một cấu trúc dữ liệu được tối ưu cho việc đọc và cơ chế cập nhật nó.
2. **File cần tạo/sửa**:
   * [schema.sql](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)
   * order-management/src/infrastructure/projection-updater.ts
3. **Thực hiện**:
   * Trong schema.sql, CREATE TABLE daily\_sales\_summary (...) để lưu doanh thu theo ngày.
   * Trong projection-updater.ts, viết một service lắng nghe các event (ban đầu có thể là đọc trực tiếp từ bảng events). Khi có OrderCreatedEvent, nó sẽ INSERT hoặc UPDATE vào bảng daily\_sales\_summary.

**Bước 2: Infrastructure Layer - Repository cho Read Model**

1. **Mục tiêu**: Đóng gói logic truy vấn dữ liệu báo cáo.
2. **File cần tạo**: order-management/src/infrastructure/read-model-repository.ts.
3. **Thực hiện**:
   * Tạo class ReadModelRepository.
   * Viết phương thức getMonthlySalesReport(month, year) để SELECT dữ liệu từ bảng daily\_sales\_summary.

**Bước 3: Application Layer - Query Handler**

1. **Mục tiêu**: Điều phối việc lấy dữ liệu báo cáo.
2. **File cần tạo**: order-management/src/application/query-handlers.ts.
3. **Thực hiện**:
   * Tạo class ReportQueryHandlers.
   * Viết phương thức handleGetMonthlySalesReport(query) để gọi readModelRepository.

**Bước 4: Infrastructure Layer - Dịch vụ tạo file CSV**

1. **Mục tiêu**: Chuyển đổi dữ liệu từ DB thành file CSV.
2. **File cần tạo**: order-management/src/infrastructure/report-service.ts.
3. **Thực hiện**:
   * Cài đặt thư viện fast-csv.
   * Viết class ReportService với phương thức generateCsv(data) để tạo chuỗi CSV.

**Bước 5: Presentation Layer - Endpoint xuất báo cáo**

1. **Mục tiêu**: Cung cấp API để tải file.
2. **File cần sửa**: routes.ts và controller.ts.
3. **Thực hiện**:
   * Trong routes.ts, thêm route GET /api/reports/monthly-sales.
   * Trong controller.ts, thêm phương thức getMonthlySalesReport để:
     1. Gọi queryHandler.
     2. Gọi reportService để tạo CSV.
     3. Trả về response với header Content-Type: text/csv và file đính kèm.

**Bước 6: Frontend - Giao diện tải báo cáo**

1. **Mục tiêu**: Cho phép người dùng chọn tháng/năm và tải về.
2. **File cần sửa**: [page.tsx](vscode-file://vscode-app/c:/Users/Admin/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html).
3. **Thực hiện**:
   * Thêm các ô chọn tháng/năm và nút "Export".
   * Khi nhấn nút, tạo một link đến endpoint báo cáo và tự động click để tải file.
4. **Liên kết hai góc nhìn**

* **Logic View**: Cho thấy các thành phần và mối quan hệ tĩnh → giúp xác định viết code ở đâu.
* **Process View**: Cho thấy trình tự tương tác động → giúp hiện thực đúng luồng dữ liệu.
* Kết hợp: Lập trình viên vừa biết vị trí module, vừa hiểu flow khi triển khai.