# SUPPLYMIND

## SMART SUPPLY CHAIN INTELLIGENCE PLATFORM

JONATHAN GLAAB
MINH TRIET VU
ELIZABETH MOKRUSA
FATEMEH ZAREZADEH MEHRIZI

HTTPS://SUPPLYMIND-FRONTEND-7C89888C6700.HEROKUAPP.COM/LOGIN

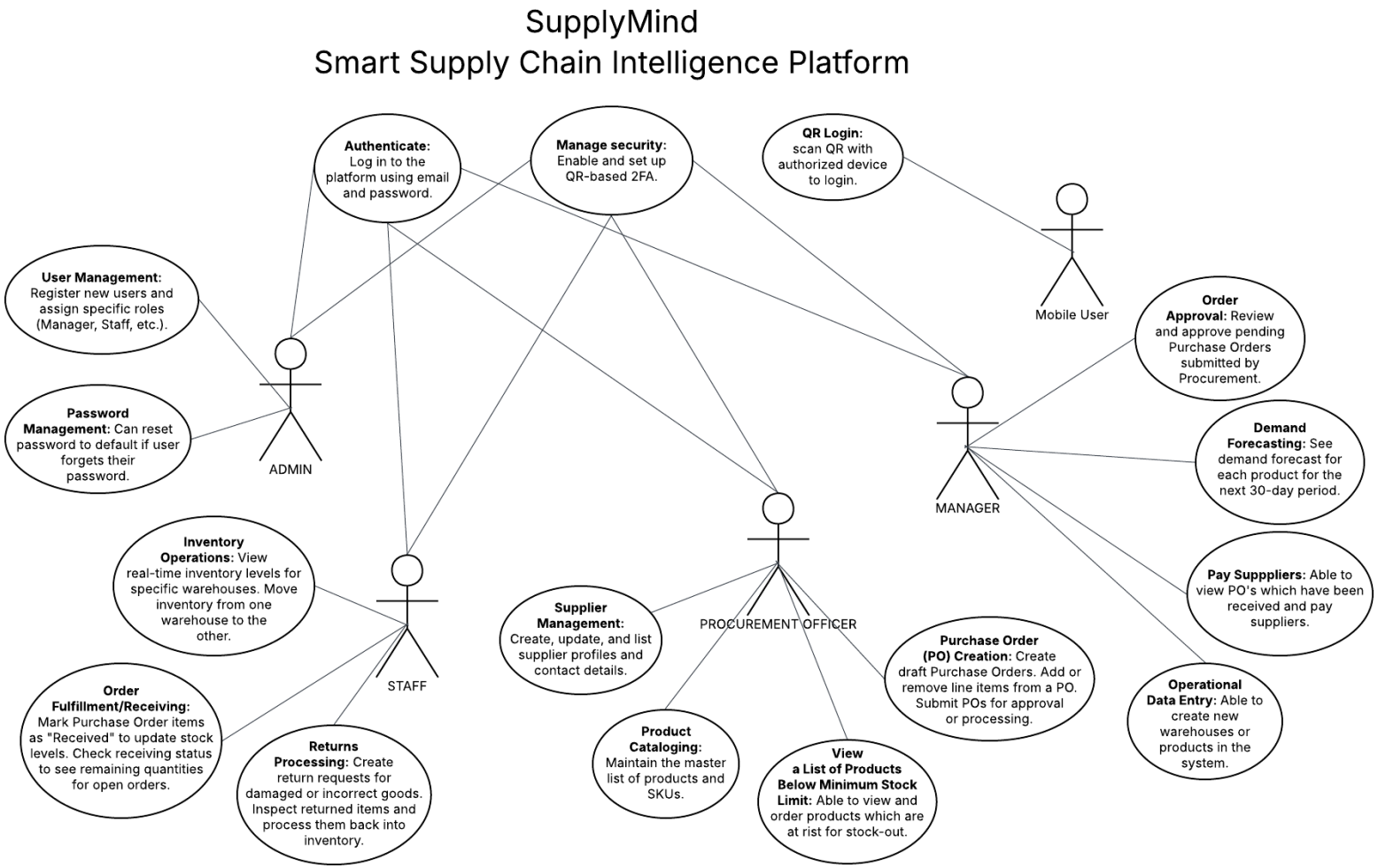
# *BACKGROUND*

- Supply Chain Management is a Challenge for SMEs
  - **Disconnected Systems:** Businesses often rely on separate tools for inventory, sales, and procurement, leading to data silos and operational inefficiencies.
  - **Reactive "Fire-Fighting":** Without a unified view, managers are forced to react to problems like stockouts or overstocking only after they occur, causing lost sales and increased costs.
  - **Lack of Data-Driven Insight:** Manual tracking and guesswork make it difficult to forecast demand, identify risks, or optimize purchasing decisions, hindering growth.
  - **Complex Financial Reconciliation:** Managing payments to suppliers and from customers across different platforms is time-consuming and prone to error.

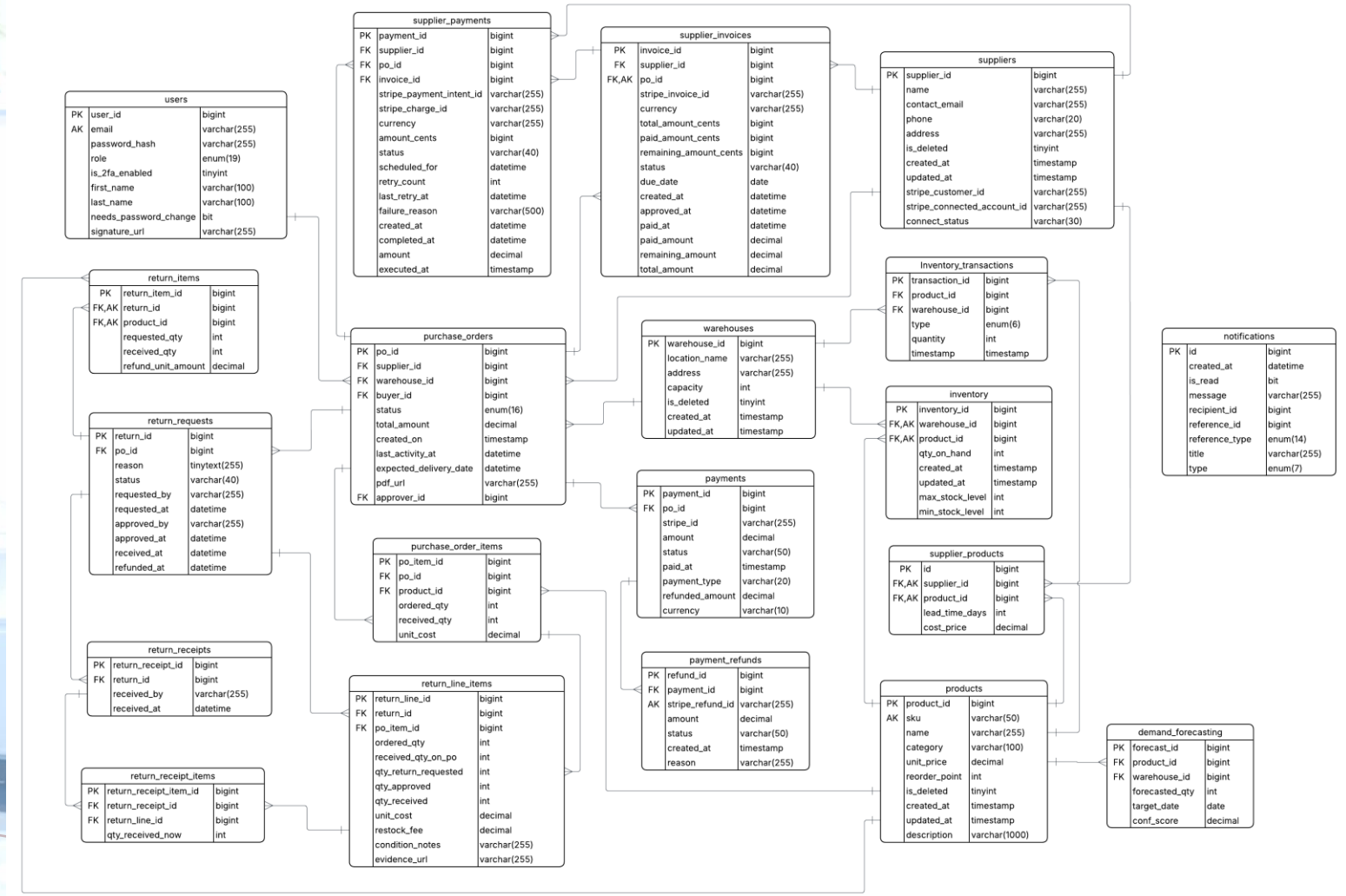
# *SOLUTION:* SUPPLYMIND, A CENTRALIZED COMMAND CENTER

- **End-to-End Integration:** SupplyMind provides a single platform to manage the all warehouse operations from forecasting and procurement, to inventory reception and supplier payment.
- **Proactive, Data-Driven Management:** Our app transforms operational data into actionable insights, enabling real-time visibility and smarter, proactive decision-making.
- **Streamlined Core Operations:**
  - **Entity Management:** Easily track products, suppliers, and warehouses.
  - **Procurement:** Automate purchase orders.
  - **Logistics & Inventory:** Gain real-time control over inventory movements, including receiving and returns.
  - **Financial Integration:** Simplify transactions with secure, integrated payment processing.

# SOLUTION


SupplyMind
Smart Supply Chain Intelligence Platform
Authenticate: Log in to the platform using email and password.
Manage security: Enable and set up QR-based 2FA.
QR Login: scan QR with authorized device to login.
Mobile User
User Management: Register new users and assign specific roles (Manager, Staff, etc.).
Password Management: Can reset password to default if user forgets their password.
ADMIN
Order Approval: Review and approve pending Purchase Orders submitted by Procurement.
Demand Forecasting: See demand forecast for each product for the next 30-day period.
MANAGER
Pay Suppliers: Able to view PO's which have been received and pay suppliers.
Operational Data Entry: Able to create new warehouses or products in the system.
Inventory Operations: View real-time inventory levels for specific warehouses. Move inventory from one warehouse to the other.
Order Fulfillment/Receiving: Mark Purchase Order items as "Received" to update stock levels. Check receiving status to see remaining quantities for open orders.
Returns Processing: Create return requests for damaged or incorrect goods. Inspect returned items and process them back into inventory.
STAFF
Supplier Management: Create, update, and list supplier profiles and contact details.
Product Cataloging: Maintain the master list of products and SKUs.
PROCUREMENT OFFICER
Purchase Order (PO) Creation: Create draft Purchase Orders. Add or remove line items from a PO. Submit POs for approval or processing.
View a List of Products Below Minimum Stock Limit: Able to view and order products which are at rist for stock-out.

# CHALLENGES AND SOLUTIONS



# *CHALLENGES AND SOLUTIONS: AI OR NOT AI*

- **The Challenge:** Everyone expects "AI" to solve everything. But for supply chain forecasting, AI are often "black boxes"—they require massive datasets to train and can hallucinate numbers.
- **The Solution:** Holt-Winters Exponential Smoothing algorithm
  - **Why:** It explicitly handles **Seasonality** (holidays/spikes) and **Trends** (growth) using standard statistical mathematics
  - **Implementation:** A custom Java implementation (HoltWinters.java) that calculates Level, Trend, and Seasonality components
  - **Result:** Deterministic forecasts that a warehouse manager can trust, without the overhead or unpredictability of an LLM

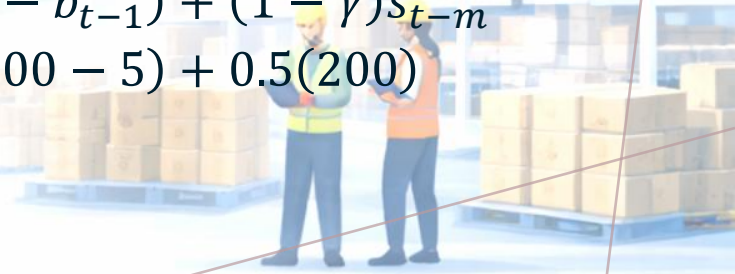
# HOLT-WINTER METHOD

- Level: $\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$
- Trend: $b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1}$
- Seasonal: $s_t = \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$
  - **α** : Controls level smoothing—equation weight on current observation vs prior level+trend. At 0.5, equally balances recent data and historical smoothed level.
  - **β** Controls trend smoothing—weights current trend change vs prior trend. 0.5 gives moderate responsiveness to trend shifts
  - **γ** Controls seasonal smoothing—weights current seasonal residual vs prior seasonal factor. 0.5 adapts steadily to seasonal pattern changes.

# HOLT-WINTER METHOD

- Baseline: 100 shovels/day
- Trend: +5 shovels/day
- Seasonality: In December, we sell extra +200 shovel
- Prediction: Baseline (100) + Trend (5) + Seasonality (200) = **305 Predicted Sales**
- Reality: blizzard hits, we sold 400 shovels

- Level update:
  - $\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1})$
  - $\ell_t = 0.5(400 - 200) + 0.5(100 + 5) = 152.5$
- Trend update:
  - $b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1}$
  - $b_t = 0.5(152.5 - 100) + 0.5(5) = 28.75$
- Seasonality update:
  - $s_t = \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}$
  - $s_t = 0.5(400 - 100 - 5) + 0.5(200) = 247.5$



# *CHALLENGES AND SOLUTIONS: FRICTIONLESS WORKFLOW*

- **The Challenge:** Workers wear gloves and share terminals. Passwords are slow and bringing a laptop everywhere is not helpful
- **The Solution:** A dedicated Mobile-First Experience & Touchless Auth
  - **Mobile Mode:** streamlined mobile interface (MobileLayout.jsx) that strips away navigation clutter, focusing purely on tasks: Scan, Receive, Transfer, Return.
  - **QR Login**: WebSocket-driven login flow (DesktopLoginQR.jsx). A user scans a code on the desktop with their logged-in phone, and the desktop instantly logs them in via a secure token transfer.
  - **Battery preservation**: 20 seconds timeout on all camera related task, dark theme by default

# *REALIZING COMPLEX ABSTRACT SYSTEMS*

- **The Challenge:** When making a complex multi-layered system, minor decisions affect the options of users and flow of business
- **The Solution:** Regular board meetings to discuss business decisions
  - Realizing complex abstract systems has many downstream effects
  - Could be alleviated with direction during the creation and planning phases
  - How we shape the interfaces shapes the business capabilities
  - More options are not always desirable
  - Some limits are good but have larger implications

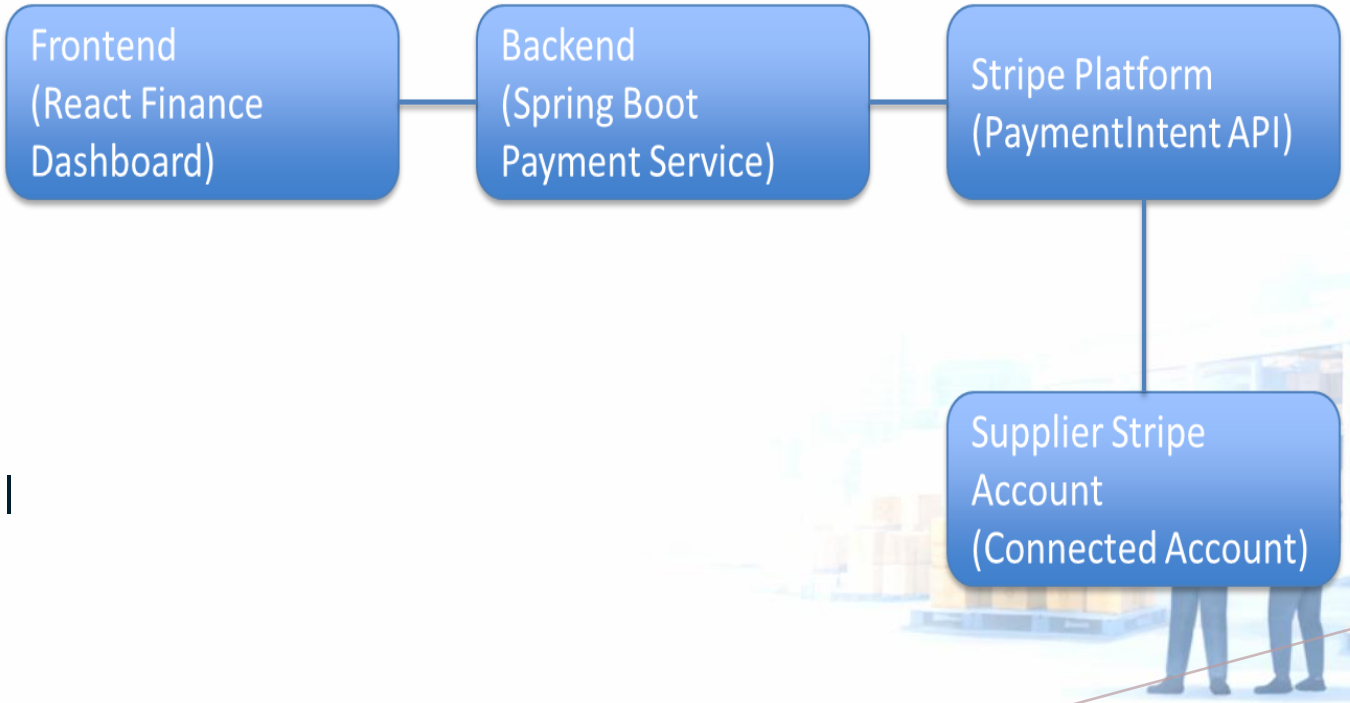
# *WORKING ON HALF A PIPELINE*

- **The Challenge:** Our first project with fully dedicated frontend and backend teams
- **The Solution:** Following the schematics of our project proposal, communication
  - This is our first project with full separation between backend and frontend
  - Following an architecture of intention
  - Attempting seamless connection without having written the methods
  - Drift from changes in scope and complexity of project

# CHALLENGES AND SOLUTIONS

The Payment Challenge

- SupplyMind is a platform connecting:
  - Buyers
  - Finance team
  - Multiple suppliers
- The platform must:
  - Pay multiple suppliers
  - Track payment lifecycle
  - Support partial payments
  - Maintain financial audit trail



# CHALLENGES AND SOLUTIONS

Why We Chose Stripe Connect for Supplier Payouts

- **Goal:** Pay external suppliers, not just collect payments from customers.
- **Problem with standard Stripe (single account):**
  - Payments stay in one platform Stripe account.
  - No built-in “per supplier” money flow and compliance for marketplaces.
- **Why Connect fits SupplyMind:**
  - Each supplier can be represented as a connected account
  - Clear separation of platform vs supplier funds
  - Scales to many suppliers and supports payout workflows

```
@Column(name = "stripe_connected_account_id")
private String stripeConnectedAccountId;

@Enumerated(EnumType.STRING)
@Column(name = "connect_status", nullable = false)
@Builder.Default
private SupplierConnectStatus connectStatus = SupplierConnectStatu
```

# CHALLENGES AND SOLUTIONS

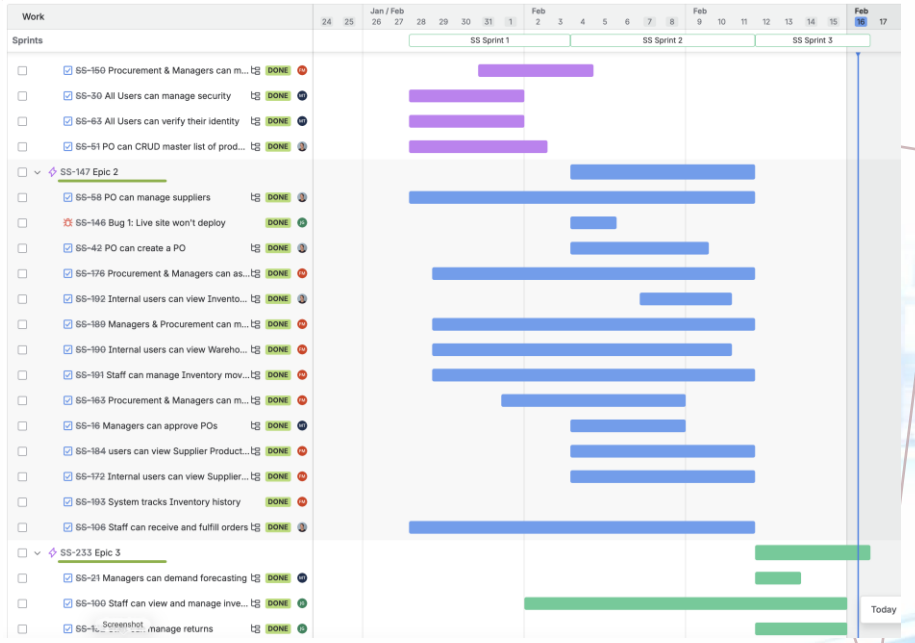
PaymentIntent Redirect Rule Broke “Execute Payment"

- **Issue:** Clicking “Execute Payment” returned a Stripe error
  - PaymentIntent could accept redirect-based payment methods
  - Stripe required a return_url (for off-site redirects)
- **Impact:** Our backend “execute” call couldn’t reliably complete payment
- **Solution:** Force no-redirect payment methods
  - automatic_payment_methods[allow_redirects] = never
  - Keeps checkout inside our app and avoids return_url requirement

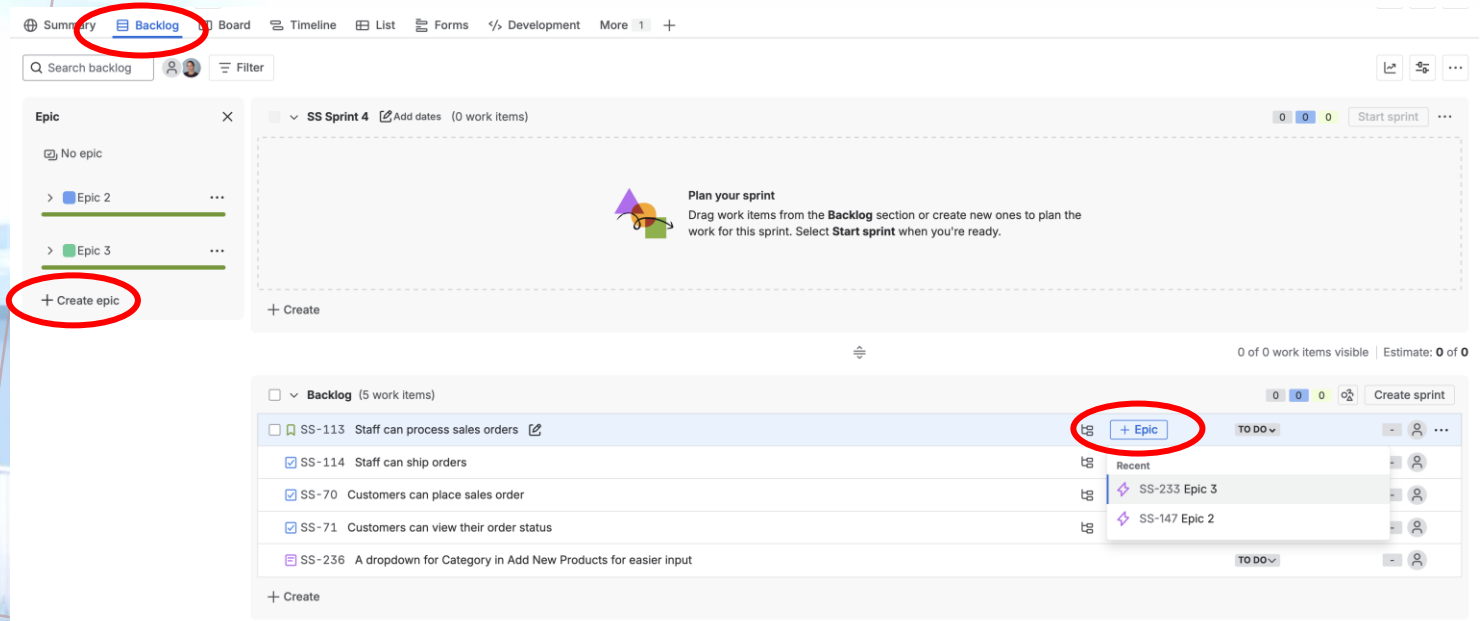
```
String testPaymentMethod = "pm_card_visa";
PaymentIntentCreateParams params =
        PaymentIntentCreateParams.builder()
                .setAmount(amountCents)
                .setCurrency(defaultCurrency.toLowerCase())
                // prevent redirect-based PMs → no return_url needed
                .setAutomaticPaymentMethods(
                        PaymentIntentCreateParams.AutomaticPaymentMethods.builder()
                                .setEnabled(true)
                                .setAllowRedirects(
                                        PaymentIntentCreateParams.AutomaticPaymentMethods.AllowRedirects.NEVER
                                )
                                .build()
                )
                // test payment method + confirm now
                .setPaymentMethod(testPaymentMethod)
                .setConfirm(true)

                .setDescription("Supplier payment #" + sp.getSupplierPaymentId())
```

# WHAT WE LEARNED

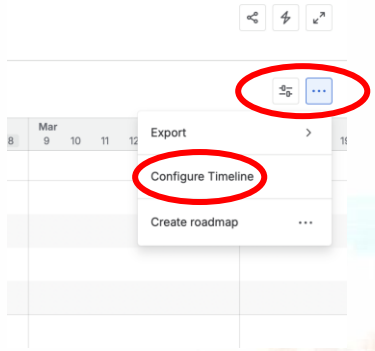
- How to create make Jira timeline look like a Gaant chart:



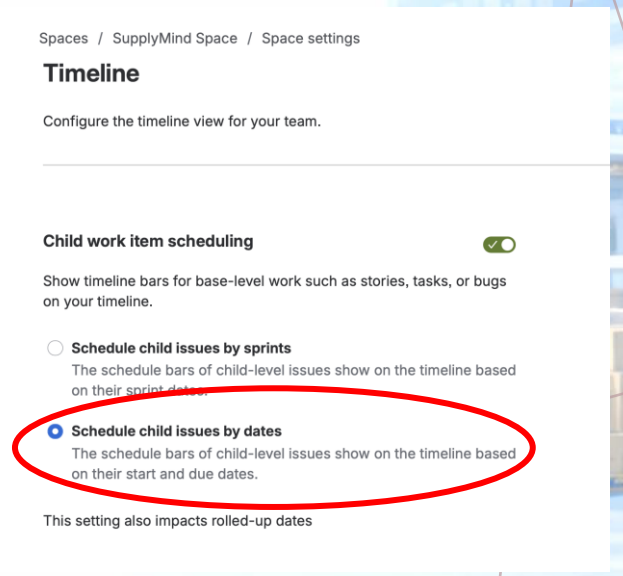

#1




#2




#3



# WHAT WE LEARNED

- Using AI for code documentation:
  - Javadoc for java and JSDoc for JavaScript and React – or Gemini Code Plugin to replace both

```java
// ----------------------------
// CREATE DRAFT
// ----------------------------
@Override 1 usage  fzmehrizi +2
@Transactional
public PurchaseOrderResponse createDraft( @NotNull PurchaseOrderCreateRequest req) {

    Supplier supplier = supplierRepo.findById(req.supplierId())
            .orElseThrow(() -> new NotFoundException("Supplier not found: " + req.supplierId()));

    Warehouse warehouse = warehouseRepo.findById(req.warehouseId())
            .orElseThrow(() -> new NotFoundException("Warehouse not found: " + req.warehouseId()));
```

```java
/**
 * Creates a new Purchase Order in a DRAFT status.
 *
 * @param req The request object containing the supplier and warehouse IDs.
 * @return A response object representing the newly created Purchase Order.
 */
@Override 1 usage  fzmehrizi +2
@Transactional
public PurchaseOrderResponse createDraft( @NotNull PurchaseOrderCreateRequest req) {

    Supplier supplier = supplierRepo.findById(req.supplierId())
            .orElseThrow(() -> new NotFoundException("Supplier not found: " + req.supplierId()));

    Warehouse warehouse = warehouseRepo.findById(req.warehouseId())
            .orElseThrow(() -> new NotFoundException("Warehouse not found: " + req.warehouseId()));
```

PurchaseOrderServiceImpl.java

```jsx
export function ApprovalModal({ poId, isOpen, onOpenChange, onPoUpdated }) {  Show usages  Elizabeth Mokrusa
    const [po, setPo] = useState( initialState: null);
    const [loading : boolean , setLoading] = useState( initialState: true);
    const [error : string , setError] = useState( initialState: '');
    const [selectedStatus : string , setSelectedStatus] = useState( initialState: '');

    // NEW: State to control the separate SendEmailModal
    const [isEmailModalOpen : boolean , setIsEmailModalOpen] = useState( initialState: false);
```

```jsx
/**
 * A modal dialog for viewing and managing the details of a Purchase Order.
 * It allows users with appropriate permissions to approve, reject,
 * manually update the status, and email the PO to a supplier.
 *
 * @param {object} props - The component's props.
 * @param {number} props.poId - The ID of the Purchase Order to display.
 * @param {boolean} props.isOpen - Controls whether the modal is open or closed.
 * @param {function} props.onOpenChange - Callback function to handle modal visibility changes.
 * @param {function} props.onPoUpdated - Callback function to execute after a PO has been successfully updated.
 */
export function ApprovalModal({ poId : number , isOpen : boolean , onOpenChange, onPoUpdated }) {  Show usages  Elizabeth Mokrusa *
    const [po, setPo] = useState( initialState: null);
    const [loading : boolean , setLoading] = useState( initialState: true);
    const [error : string , setError] = useState( initialState: '');
    const [selectedStatus : string , setSelectedStatus] = useState( initialState: '');
```

ApprovalModal.jsx

# *WHAT WE LEARNED: WORKING WITH EMAIL*

- **The Insight:** Real-world clients use diverse, legacy email servers
- **Architecture:** We decoupled our logic from specific vendors (like Google)
  - **Reading:** Uses IMAP Standard (InboxProvider interface)
  - **Sending:** Uses SMTP Standard (EmailProvider interface)
- **Result:** The entire system is "Plug-and-Play." We can switch from Gmail to Outlook just by changing the host and port in the config file—**zero code changes**

# *WHAT WE LEARNED*
# *AI: "CO-PILOT" VS. "AUTOPILOT"*

- **Insight:** Unchecked AI is dangerous (hallucinations), that's why we implemented guardrails and human in the loop.
- **The Strategy**:
  - **Inbound (Autopilot):** We use strict **JSON Guardrails** to parse incoming status updates safely
  - **Outbound (Co-Pilot):** We use **Generative AI** to *draft* emails, but we enforce a **Human-in-the-Loop** to review and send
- **Lesson:** AI is best used to *augment* human speed, not replace human judgment

# *WHAT WE LEARNED*

Robust Payment Pipeline (Schedule → Intent → Finalize → Ledger)

- We built a full payment lifecycle:
  - **Schedule Payment:** Creates SupplierPayment record (audit + timeline)
  - **Create PaymentIntent:** Generates `clientSecret` for Stripe Elements
  - **Finalize / Execute:** Updates invoice balances and statuses
  - **Key learning:** Payments are not a "single click"
  - **Must handle states**: SCHEDULED → PROCESSING → PAID/FAILED
    - Ensure DB remains the source of truth (invoice remaining, paid amount)
  - **Outcome:** Professional UX + traceable finance history
    - Payment timeline is searchable, filterable, and supports partial payments

# WHAT WE LEARNED

```jsx
<Card className="pay-card pay-card--dark">
  <CardHeader>
    <CardTitle className="text-white text-2xl">Card Payment</CardTitle>
  </CardHeader>

  <CardContent>
    {!clientSecret || creatingIntent ? (
      <div className="flex items-center gap-2 text-slate-200">
        <Loader2 className="h-4 w-4 animate-spin" />
        Preparing secure payment…
      </div>
    ) : (
      <Elements stripe={stripePromise} options={options}>
        <CheckoutCardForm poId={poId} amountLabel={amountLabel} />
      </Elements>
    )}
```

```java
322     public Long scheduleSupplierPayment(ScheduleSupplierPaymentRequestDTO dto) {
355
356         SupplierPayment sp = SupplierPayment.builder()
357                 .invoice(inv)
358                 .po(inv.getPo())
359                 .supplier(inv.getSupplier())
360                 .amount(amountToPay)
361                 .amountCents(amountCents)
362                 .currency(inv.getCurrency() == null ? "cad" : inv.getCurrency())
363                 .status(SupplierPaymentStatus.SCHEDULED)
364                 .scheduledFor(dto.getScheduledFor() == null ? Instant.now() : dto.getScheduledFor())
365                 .retryCount(0)
366                 .build();
367
368         sp = supplierPaymentRepo.save(sp);
369
370         inv.setStatus(SupplierInvoiceStatus.SCHEDULED);
371         invoiceRepo.save(inv);
372
373         return sp.getSupplierPaymentId();
```

# SUMMARY OF THE TECHNOLOGIES

- Backend: Java (Spring Boot, Spring Security)
- Frontend: React (Vite)
- Database: MySQL (Aiven.io)
- Payments: Stripe Connect
- Hosting & Storage: Heroku, Backblaze B2
- Architecture: MVC
- AI: OpenRouter (Arcee AI)
- Email: Gmail Services

# *FUTURE OF WORK*

- Have a customer storefront
- Customer ability to view orders
- Notifications for Staff or Procurement Officers
- Database efficiency

# *DISTRIBUTION OF WORK*

- Jonathan
  - Set up Git repository
  - Configuration Management
  - Application Monitoring & Logging
  - Warehouse and Staff views
  - Supplier and supplier product lists

- Minh
  - Email in app implementation
  - AI in the automatic updates to PO status
  - Models in back end
  - Component library in the front end
  - Forecasting model and view

# *DISTRIBUTION OF WORK*

- Fatemeh
  - Setup and configuration of hosting, database, and storage infrastructure
  - Design and implementation of the backend architecture
  - Development of the majority of backend services and business logic
  - Research and implementation of Stripe Connect integration on the backend
  - Development of the Financial Dashboard frontend and integration with purchase orders ready for payment
- Elizabeth
  - Admin dashboard and functionality
  - Manager PO approval process
  - Connect forecasting view for manager
  - Products pages and CRUD
  - JIRA MASTER

# *THANK YOU*