MEX Assistant Proposal - Sudo_Chill

Background

Grab's merchant-partners operate in a fast-paced market with large volumes of data from orders, menu items, and customer interactions. However, many small business owners lack the time or expertise to analyze this data for actionable insights. Grab's vision of empowering Southeast Asian micro-entrepreneurs through technology drives the need for an AI assistant that can bridge this gap. By leveraging recent advances in AI, especially conversational agents, we see an opportunity to turn raw data into a **personalized business coach** for every merchant. This background sets the stage for **MEX Assistant**, a multilingual conversational insights platform designed to help Grab merchant-partners make data-driven decisions and improve their livelihoods.

The Challenge & Opportunity

Challenges: Grab merchants need to stay on top of key metrics (sales trends, popular items, delivery times, etc.) in real time, but existing tools (dashboards, reports) can be too complex or static. Key challenges include:

- **Real-time Insight & Alerting:** How to deliver up-to-the-minute business insights and anomaly alerts (e.g. sudden drop in sales or delayed orders) without the merchant constantly monitoring reports.
- **Personalization:** Merchants vary in type, location, and scale, so insights must be tailored – a one-size-fits-all analysis won't resonate. For example, a small cafe in Manila has different patterns and needs than a large restaurant in Kuala Lumpur.
- **Diverse Language & Skill Levels:** Southeast Asian merchants speak multiple languages (Bahasa, Thai, Vietnamese, English, etc.), and not all are data-savvy. The solution must communicate insights in the merchant's preferred language and in an intuitive manner.

Opportunity: By addressing these challenges, we can empower merchants with a tool that not only answers their questions but **proactively guides them**. The opportunity is to transform data into a dialogue: instead of poring over charts, a merchant can simply ask "How were my sales this week?" or be automatically notified "Lunchtime orders are slower than usual today." This conversational approach lowers the barrier to understanding complex analytics. It aligns with Grab's mission to use AI for economic empowerment, turning data-driven guidance into a daily asset for merchants. A successful solution could improve merchants' revenue and efficiency, strengthen their loyalty to Grab, and differentiate Grab's merchant platform with innovative AI support.

Our Solution

MEX Assistant is an AI-powered conversational insights platform built as a chat-based assistant integrated with a merchant dashboard. It acts as a **virtual business analyst** for Grab merchant-partners. The assistant ingests the merchant's data (menu info, transactions, customer feedback, etc.) and provides insights, recommendations, and answers via a friendly chat interface. Key aspects of our solution include:

- **Conversational Q&A:** Merchants interact with MEX Assistant by chatting in natural language (text or voice). They can ask questions about sales, inventory, customer behavior or any business metric, and get an immediate, coherent answer. The assistant can handle questions like *"What were my top 3 selling items last month?"* or *"How did this week's revenue compare to last week?"* and provide answers grounded in the actual data (with figures, trends, even brief explanations).
- **Proactive Guidance:** MEX Assistant doesn't wait to be asked – it actively monitors the merchant's data and alerts them to significant insights or anomalies. For example, it might alert *"Lunch hour orders today are 20% below average – you might want to activate a promo to attract customers."* If food preparation times spike, it might warn the merchant of potential delays. This proactive insight delivery ensures the merchant is **never caught off guard** by negative trends or missed opportunities.

- **Integrated Visual Dashboards:** The solution combines chat with a dashboard view. The assistant not only tells insights but can **show** them. When appropriate, it presents charts or tables alongside its messages – e.g. a line chart of weekly sales or a bar chart of top-selling items. The interface allows merchants to visually verify and explore the data behind the assistant's advice.
- **Multilingual Support:** MEX Assistant communicates in the merchant's preferred language. A merchant can chat in Bahasa Indonesia, Thai, Vietnamese, English, etc., and the assistant will respond in that language. This is achieved by leveraging the multilingual capabilities of advanced LLMs and translation where needed – the AI's core logic remains language-agnostic, with localization handled in prompts and the UI. All static text in the interface is localized via the UI layer (using i18n libraries), so the overall experience feels native to the user's language without changing the underlying analytics.

In essence, our solution turns complex data analysis into a simple conversation. It empowers merchants to act on insights in real time, leading to better decision-making and improved business outcomes, all through a natural, human-like interaction.

Key Features & Innovation

Our MEX Assistant stands out through a combination of innovative features designed to impress both users and hackathon judges:

- **Real-Time Analytics & Alerts:** The system continuously analyzes incoming data (orders, deliveries, etc.) and immediately surfaces important findings. For example, if order preparation time exceeds the city average, the assistant will alert the merchant to this delay **in real time**, so they can take action (e.g. prep popular items in advance during a rush). All insights are updated continuously (or on a frequent schedule for heavier computations) to reflect the latest conditions.
- **Personalized Recommendations:** The assistant tailors its guidance to each merchant's context. It knows the merchant's top items, typical performance, and even peer benchmarks. Insights are not generic but specific (e.g. suggesting a menu tweak only if the data indicates it's relevant). It might recommend adding a new flavor of a popular dish if analysis shows that similar

merchants saw success with it. This personalization is powered by both **clustering** (grouping similar merchants to identify what top performers do) and content analysis (e.g., identifying if beverages drive most sales for a particular merchant).

- **Hybrid AI Intelligence:** MEX Assistant employs a hybrid of **rule-based analytics, traditional ML, and a cutting-edge Large Language Model (LLM)**. This is a key innovation: the LLM (e.g. GPT-4) provides flexible natural language understanding and generation, while behind the scenes we have structured models (forecasts, anomaly detectors, recommender systems) ensuring factual accuracy. The LLM functions as the "brain" orchestrating these components, calling on tools/functions to get exact data when needed. This synergy means the assistant's responses are both **conversational** and **data-aware** – a significant improvement over chatbots that only do one or the other.
- **Interactive Visualizations:** Data insights often are best understood with visuals. Our platform integrates chart components (e.g., line charts for trends, bar charts for comparisons). The **Dashboard** view (and even the chat) can display interactive charts for things like weekly sales, category breakdown, top N items, etc. For instance, when the assistant says *"Your sales are trending up"*, it can present a line chart of daily sales to back that up. Judges can see a seamless blend of dialogue and visualization, highlighting the assistant's capability to *both tell and show* insights.
- **Voice-Enabled & User-Friendly Interface:** The chat interface is designed for simplicity – short text responses and optional voice input. A busy merchant could even press a microphone button and ask a question verbally. This lowers the barrier further for users who find typing cumbersome. The UI also provides quick-click suggested questions (common queries) to guide users on what to ask, making the experience approachable from the first use.
- **Multilingual and Localized:** As noted, the assistant naturally supports multiple languages through the LLM's multilingual training and careful prompting. In addition, the app's UI is localized (buttons, labels, date formats, currencies) for Southeast Asian languages. This dual approach (LLM for free-form language, and UI localization for interface text) means the core analytical logic doesn't

change per language – we get internationalization with minimal additional engineering, which is a smart, scalable design.

- **Proactive Insight Delivery:** Uniquely, MEX Assistant behaves not just as a passive tool but as an active partner. It can send **unsolicited** but relevant messages, like a morning business summary or an alert about an unusual pattern forming. This is an innovation in the merchant space – turning a dashboard from something you have to check, into an assistant that checks on things for you and *pings you* when you need to know. It embodies a shift from reactive analytics to proactive coaching.
- **Data-Driven "What-If" Scenarios (Planned):** Looking beyond the prototype, our design allows for *scenario simulation* features (see the **Advanced Extension** section). This would let merchants ask hypothetical questions ("What if I increase my delivery fee by 10%?") and get forecasted outcomes. While not fully implemented in the initial prototype, planning for this shows the forward-thinking, innovative roadmap of the project.

Each of these features contributes to a solution that is not only technically impressive but also truly useful. The combination of conversational AI with real business analytics and multilingual support is innovative in the context of Grab's merchant tools. It directly addresses the challenges identified and leverages AI in a novel way to empower users.

Technical Execution & Architecture

Our implementation follows a **modular, scalable architecture** split between a React frontend and a Python backend. This separation of concerns ensures that the user experience is smooth and that heavy data processing and AI logic are handled server-side. Below we detail the frontend and backend design, including how key components from the codebase and data pipeline come together.

Frontend: React-Based Web Interface

The frontend is a single-page web application built with **React** (bootstrapped with Vite for fast dev). It delivers a modern, responsive UI that combines chat and dashboard elements seamlessly. Key aspects of the frontend implementation:

- **Component Structure:** We employed a component-based architecture for clarity and reuse. At a high level, a `ChatContainer` component encapsulates the entire chat interface. According to our code, `ChatContainer` wraps sub-components like `ChatHeader`, `ChatMessages`, `SuggestedQuestions`, and `ChatInput`, all managed within a React context (`ChatContext`) to easily share state (conversation history, loading status, etc.). The `ChatHeader` includes the assistant's title and controls (e.g. a language selector and a "clear chat" button), while `ChatMessages` renders the transcript of the conversation, and `ChatInput` provides an input box (with send button and optional voice-record feature).
- **Dashboard & Charts:** In addition to the chat, we built a `Dashboard` component that displays key metrics and visualizations. The Dashboard can be toggled from the main interface (our `MainPage.jsx` allows switching between the Chat view and Dashboard view). The Dashboard fetches data via the backend's insights API and renders charts such as a sales trend line chart and a bar chart of top items sold. We created reusable chart sub-components (collectively referred to here as **ChartVisualizations**), possibly including components like `ChartItem` for individual charts, using a library like Chart.js or Recharts under the hood. These charts are interactive (hover for details) and are designed to complement the chat – e.g., a user might see the dashboard for a broad overview and then ask the assistant a question for details on something they see.
- **State Management & Performance:** We use React hooks and context to manage state like current merchant selection, time range filters, and cached data. For example, the Dashboard uses a `useEffect` hook to fetch data whenever the selected merchant or time filter changes, and it caches results to avoid redundant calls. The code snippet below shows how we implemented a data fetch with caching for dashboard data and top items:

  Code (simplified): The dashboard fetch function calls `getDashboardData` (for summary stats), and in parallel fetches `getTopItems` and `getAiInsights`. It then combines these results into one `combinedData` object which is stored in state and cached. This ensures the UI remains responsive by avoiding unnecessary recomputation on each view.
- **User Experience Features:** The interface is designed to be clean and intuitive. We included a **suggested questions** bar (`SuggestedQuestions` component) that

surfaces common queries as buttons (e.g. "How are my sales this week?") to guide users. There's also basic navigation for future extensibility (e.g., placeholders for Settings or Notifications pages). Styling is done with CSS (and could be enhanced with a UI library like Material-UI for production). The app is fully responsive, so merchants can use the assistant on a tablet or small laptop in their shop. Importantly, all UI text and labels go through a localization module (`react-i18next` integration, with resource files for multiple languages), ensuring that a merchant sees the interface in Bahasa, Thai, etc., as per their preference, without altering the core logic.

Overall, the frontend acts as a smart presenter: it captures user input (text or voice), displays the assistant's replies and charts, and handles multi-language UI needs. It communicates with the backend via RESTful API calls, making the system architecture flexible (the frontend and backend can evolve or scale independently, e.g., deploying updates to the UI without touching backend code, and vice versa).

Backend: FastAPI Service & AI/Analytics Engine

The backend is the powerhouse of MEX Assistant, implemented in Python with **FastAPI** to handle web requests and a suite of data analytics modules and AI components. Its responsibilities include processing chat queries, running data analysis, interfacing with the LLM, and providing data to the frontend. Key details of the backend:

- **Service Architecture:** We structure the backend as a FastAPI application with clear modular design. For example, our project has routes defined in separate modules (e.g., `chat.py` for chat query endpoints and `insights.py` for dashboard data endpoints) and service classes that implement the logic. The directory layout:

```
/mex-assistant-frontend
├── public/
│   ├── index.html          # Main HTML file (Vite injects scripts here)
│   ├── favicon.ico         # App icon
│   └── locales/           # Translation files
│       ├── en/
```

```
│       │   └── translation.json # English strings
│       ├── id/
│       │   └── translation.json # Bahasa Indonesia strings
│       ├── ms/
│       │   └── translation.json # Bahasa Melayu strings
│       ├── th/
│       │   └── translation.json # Thai strings
│       ├── vi/
│       │   └── translation.json # Vietnamese strings
│       └── zh/
│           └── translation.json # Chinese strings
├── src/
│   ├── assets/             # Static assets
│   │   ├── images/           # Logos, icons (if not using font icons)
│   │   │   └── logo.png
│   │   └── fonts/            # Custom fonts (if any)
│   ├── components/           # Reusable UI components
│   │   ├── common/           # General-purpose shared components
│   │   │   ├── Button.jsx
│   │   │   ├── Card.jsx
│   │   │   ├── Loader.jsx      # Loading animation (like the dots)
│   │   │   ├── StatCard.jsx    # Display individual stats (Total Sales, etc.)
│   │   │   ├── TimeFilter.jsx  # Today/Week/Month/Year buttons
│   │   │   ├── MerchantSelector.jsx # Dropdown in sidebar footer
│   │   │   └── Modal.jsx       # For pop-ups, if needed
│   │   ├── layout/           # Structure components
│   │   │   ├── Header.jsx      # Top bar with search, profile
│   │   │   ├── Sidebar.jsx     # Left navigation
│   │   │   └── MainContent.jsx  # Wrapper for the main view area
│   │   ├── chat/             # Components specific to the AI Assistant chat [c
ite: 114]
│   │   │   ├── ChatContainer.jsx # Main chat section wrapper
│   │   │   ├── ChatHeader.jsx    # Header with Assistant info, lang selecto
r
│   │   │   ├── ChatMessages.jsx  # Displays the list of messages
│   │   │   ├── ChatMessage.jsx   # Individual message bubble (user/assis
```

```
tant)
│   │   ├── ChatInput.jsx     # Input field and send button
│   │   ├── LanguageSelector.jsx # Dropdown for language
│   │   └── ChatVisualization.jsx # Renders charts/tables within chat [c
ite: 117]
│   ├── dashboard/        # Components specific to the Dashboard vie
w [cite: 120]
│   │   ├── DashboardContainer.jsx # Main dashboard section wrapper
│   │   ├── StatsGrid.jsx      # Grid layout for StatCards
│   │   ├── SalesTrendChart.jsx # Specific chart component for sales/o
rders
│   │   ├── HourlySalesChart.jsx
│   │   ├── DailySalesChart.jsx
│   │   ├── TopItemsTable.jsx  # Table display for top items
│   │   └── AiInsights.jsx     # Section displaying AI insight cards
│   └── visualizations/    # Reusable chart components [cite: 121, 122]
│       ├── BaseChart.jsx    # Optional: Base wrapper for chart options
│       ├── LineChart.jsx
│       ├── BarChart.jsx
│       └── Table.jsx        # Reusable table component
├── contexts/             # React contexts for global state
│   ├── AppContext.jsx     # Global settings (theme, language, selected
merchant)
│   ├── ChatContext.jsx    # Manages chat messages, loading state
│   └── DashboardContext.jsx # Manages dashboard data, filters
├── hooks/                # Custom React hooks
│   ├── useChatApi.js      # Hook for interacting with chat service (moc
k or real)
│   ├── useDashboardData.js # Hook for fetching/managing dashboar
d data
│   ├── useTimeFilter.js   # Logic for handling time filter state
│   └── useLocalization.js # Hook for i18n functions
├── services/             # API call logic
│   ├── apiConfig.js       # Base API config (baseURL, headers - maybe
empty for proto)
│   ├── chatService.js     # Functions to call chat API endpoints
```

```
│   │   ├── insightsService.js # Functions to call dashboard data API endp
oints
│   │   └── mockService.js     # **Prototype:** Mock API functions returni
ng dummy data
│   ├── data/               # **Prototype:** Store mock data files
│   │   ├── mockChatResponses.js # Sample chat responses
│   │   ├── mockDashboardData.js # Sample dashboard stats and chart d
ata
│   │   └── mockMerchants.js   # List of merchants for the selector
│   ├── i18n/               # i18n configuration
│   │   └── i18n.js           # Setup for react-i18next [cite: 157]
│   ├── pages/              # Main page views (often routing targets)
│   │   └── MainPage.jsx       # Single main page containing Layout and sw
itching between Dashboard/Chat
│   │   └── SettingsPage.jsx   # Placeholder for future settings
│   │   └── NotificationsPage.jsx # Placeholder for future notifications
│   ├── styles/             # Global styles and themes
│   │   ├── index.css          # Main CSS entry point
│   │   ├── theme.js           # JS object defining theme variables (colors, fo
nts)
│   │   └── App.css            # App-level styles
│   ├── utils/              # Utility functions
│   │   ├── dateFormatter.js   # Format dates/times
│   │   ├── numberFormatter.js # Format currency, percentages
│   │   ├── chartUtils.js      # Helpers for chart configurations/data proces
sing
│   │   └── constants.js       # App-wide constants (e.g., API keys if neede
d, event names)
│   ├── App.jsx             # Main App component (routing, context provide
rs)
│   └── main.jsx            # Entry point (renders App into the DOM) - Vite u
ses main.jsx by default
├── .env                # Environment variables (API keys, base URLs)
├── .gitignore
├── package.json
```

```
├── vite.config.js          # Vite configuration file
└── README.md
```

This clean separation of concerns ensures maintainability and clarity. The frontend communicates with the backend via REST API calls over HTTP. During development, we run the React dev server and a Uvicorn server for FastAPI concurrently; in deployment, both could be containerized and served behind a common domain.

- **Data Ingestion & Analysis Pipeline:** The backend ingests and stores merchant data needed for insights. For the hackathon prototype, this data comes from provided CSV files (e.g., `items.csv`, `transactions.csv`, `transaction_items.csv`, `keywords.csv`, etc.). We use **pandas** to load and pre-process these datasets into in-memory tables (or lightweight SQLite for structured queries, if needed). Once data is loaded, we perform a series of analytic computations to derive insights:
  - *Sales Trends:* We aggregate transaction data by date/time to compute daily and weekly sales, identify peak hours, and detect any downward or upward trends. This yields metrics like week-over-week growth, best day of week, etc., which the assistant can report (e.g., *"Friday was your highest-grossing day last week"*).
  - *Top/Bottom Items:* Using `transaction_items.csv`, we calculate the frequency and revenue of each menu item. This identifies the top sellers as well as underperforming items. The assistant can then answer questions about top items or advise on phasing out poor sellers. These calculations are updated regularly so the insights remain current.
  - *Customer Search Insights:* From `keywords.csv` (which tracks what customers search for and whether those searches convert to orders), we glean demand gaps. For example, if many customers search for "spring rolls" but the merchant doesn't offer them or the conversion rate is low, the assistant can highlight this opportunity (either suggesting to add that item or optimize its listing). This helps merchants align their menu with customer demand.
  - *Operational Alerts (Delivery Lag Detection):* We analyze timestamps from order data (`transactions.csv` which includes order placed time, pickup time, delivered time, etc.). By comparing a merchant's average preparation or

delivery times with city-wide benchmarks, we flag anomalies. For instance, if a merchant's orders are consistently delivered later than the city average for the dinner hour, that might indicate a bottleneck. The assistant can alert the merchant: *"Orders at 7-8pm are being delivered 10min slower than average – maybe due to high traffic or kitchen load; consider prep adjustments."* This kind of insight helps merchants maintain service quality.

- *Forecasting:* We trained a simple time-series model (using Facebook's **Prophet** or ARIMA) on historical sales data to forecast future demand. Though rudimentary in the prototype, it provides a baseline for the assistant to say things like *"Next week's sales are forecasted to be 5% higher, likely due to an upcoming holiday."* More importantly, this model lays the groundwork for the **scenario simulation** feature (allowing "what-if" questions by adjusting the forecast inputs – discussed later).
- *Recommendations & Patterns:* We applied basic unsupervised learning for deeper insights. Using association rule mining (via `mlxtend` or `efficient-apriori` libraries), we looked at item combinations in orders to find patterns (e.g., customers who buy **Item A** often also buy **Item B**). This can drive cross-sell suggestions: the assistant might suggest pairing or promoting items frequently bought together. We also performed K-means clustering on merchant performance metrics to categorize the merchant (e.g., into a cluster of "breakfast-focused cafes" vs "late-night kitchens"), enabling peer comparison insights ("You are in the top 10% of your peer group for average order value"). These advanced analyses illustrate the hybrid AI approach: combining explicit data mining with the conversational AI.

All these analytics are computed either in real-time or periodically. For the hackathon demo, some computations run on-the-fly when the user asks (e.g., querying top items), whereas heavier ones (like training a forecast model or mining association rules) can be precomputed once and then looked up. This strategy ensures the assistant's responses stay snappy. Where necessary, we schedule background jobs (or simply cache results) so that each query doesn't redo expensive calculations.

- **LLM Integration via LangChain:** At the heart of the backend's intelligence is the **Large Language Model** integration. We use OpenAI's GPT-4 (accessed via

API) for its strong language and reasoning capabilities, orchestrated by **LangChain** to function as an agent. The LangChain framework allows us to define **tools** (functions in our code) that the LLM can invoke during conversation. We implemented tools such as:

- `get_sales_summary(merchant_id, period)`: returns a summary of sales for the given period (e.g., total sales, % change from previous period).
- `get_top_items(merchant_id, period)`: returns the top N items and their sales counts.
- `detect_delivery_issues(merchant_id)`: checks recent delivery times for anomalies.
- `forecast_sales(merchant_id, period)`: returns a forecast for the next period's sales (using our Prophet model).

These are just examples – the design allows adding many tools easily. When a user asks a question, the LLM is prompted with the user query plus a structured context that includes a description of available tools. Rather than hallucinate numbers, the LLM will decide to call one of these tools if it needs factual data. For instance, if the user asks *"Which items should I focus on?"*, the LLM might call `get_top_items` to get the actual top 3 items, then incorporate that into its answer: *"Your best sellers last month were Chicken Rice, Fried Spring Rolls, and Iced Tea – focusing on these or related items could boost sales."* The LangChain agent ensures the LLM can interface with our analytics logic, resulting in responses that are **grounded in real data** (a huge technical win, since it prevents the AI from making up answers). This approach essentially creates a **data-aware conversational agent**, combining formal analytics with natural language explanation in the answer.

- **Conversation & Context Management:** We maintain conversation context so that the assistant can handle follow-up questions. LangChain provides memory mechanisms to feed recent dialogue back into the LLM prompt, so the assistant remembers what was just discussed. For example, after asking "How were my sales this week?", the merchant could ask "What about compared to last month?" – the assistant will understand the reference and provide the comparison, perhaps by calling the appropriate tool again with adjusted parameters. We also prepend a system message with merchant-specific context at each query (e.g., a brief summary of the merchant's profile: type of cuisine, typical daily sales, etc.). This helps the LLM generate responses that

are more personalized and relevant. It's a lightweight form of personalization achieved via prompting rather than training a custom model from scratch.

- **Multilingual Handling:** As mentioned, if a query comes in a language that the chosen LLM model cannot handle natively (for example, if we had used an English-only model), we would integrate a translation step. In our design, we opted for GPT-4 which has strong multilingual abilities, meaning it can directly understand and reply in many languages. Thus, the core logic doesn't change per language – the LLM simply treats the non-English input appropriately. In cases where an explicit translation is needed, we have a translator module ready (could be an API like Google Translate or a smaller transformer model) to translate the user query to English before sending to the LLM, and then translate the response back to the local language. This two-step translation pipeline is abstracted such that it doesn't affect how tools are used or how logic runs; it's an add-on in the input/output stage. Overall, the backend ensures that language is not a barrier: whether through the LLM's own capabilities or auxiliary translation, the assistant's analytical power remains accessible in the user's tongue.
- **APIs and Response Format:** The backend exposes a `/chat` endpoint (for example) that the frontend calls with each new user query. This endpoint orchestrates the steps: identify the user/merchant, call the LLM agent which internally may call data functions, and then returns the assistant's response. For the dashboard, a separate `/insights` endpoint provides batched data needed to render charts (e.g., a JSON containing arrays of sales over time, top items, etc.). We designed the responses such that the chat endpoint can also return structured data or references for visuals when needed. For instance, if the assistant's answer includes a chart, the backend might include a payload like `"chart_data": {...}"` or a reference ID that the frontend knows how to render. In real deployment, we could also use WebSockets or Server-Sent Events for the chat endpoint to support streaming responses (allowing the assistant's answer to appear word-by-word, and enabling push notifications from server to UI for proactive alerts). For now, the prototype uses simple request-response cycles, but our architecture keeps room for these enhancements.

In summary, the backend combines a robust data pipeline with an intelligent LLM agent. It exemplifies a **hybrid AI architecture**: traditional data processing ensures

factual accuracy and relevance, while the LLM brings flexibility in understanding and responding to the user. This design is not only effective in the current implementation but is maintainable and extensible for future growth. We can independently improve any analytic component (upgrade the forecasting model or add a new insight function) and the LLM agent will seamlessly incorporate that new capability into conversations. The use of FastAPI, a modern async framework, means our service is high-performance and ready to scale (handling concurrent requests, etc.), which is crucial as this assistant could be serving thousands of merchants concurrently in production.

System Analysis Diagrams

System Architecture Diagram

The following diagram illustrates the high-level architecture of the MEX Assistant system, showing how the user, frontend, backend, AI agent, and data components interact:

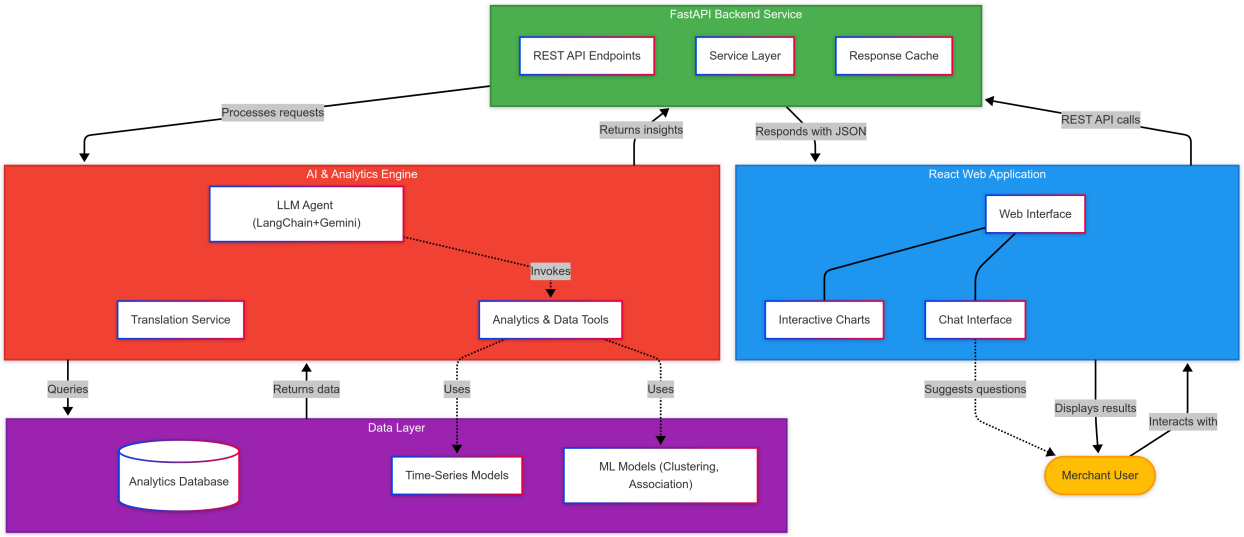


Diagram Description: The merchant interacts with the **React Web App** UI. When the merchant asks a question or requests an insight, the frontend makes a REST API call to the **FastAPI Backend**. The backend's AI engine (LLM agent) processes the query: the **LLM** may call upon **Analytics & Data Tools** (functions backed by the

Analytics Database/Data store) to fetch required information. Once the data is retrieved and the LLM formulates an answer, the backend returns the response to the frontend, which then displays it to the user (including any charts or visual elements if provided). This architecture cleanly separates concerns: the frontend handles interaction and presentation, the LLM agent handles language understanding and generation, and the data tools handle computations and retrieval, all communicating through well-defined interfaces.

Sequence Diagram of Assistant Query Flow

To demonstrate the runtime behavior, here is a sequence diagram of a typical query handled by MEX Assistant (e.g., the merchant asks a question and the assistant responds):

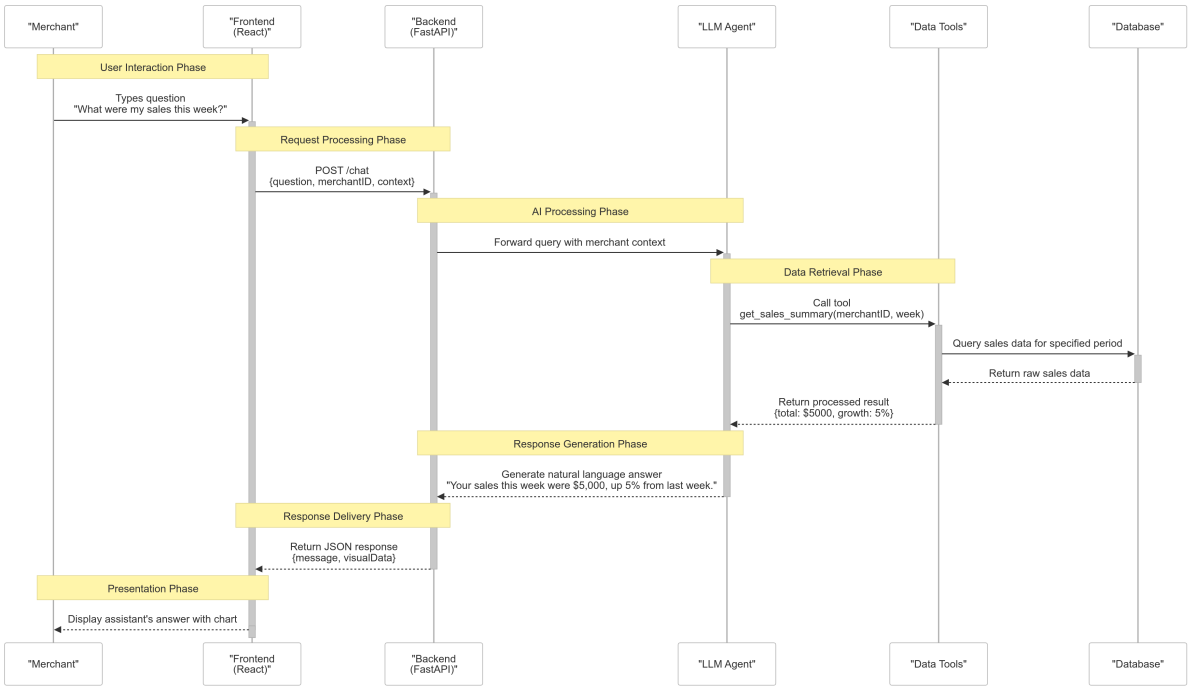


Diagram Description: The merchant's query flows from the frontend to backend, triggers data retrieval via the LLM agent's tools, and an answer flows back. Notably, the LLM agent intelligently decides to use `DataTools` to get factual data instead of guessing. In this sequence, after getting the data, the LLM formulates a natural language reply, possibly including a reference to a chart. The final

response is rendered in the UI for the merchant. This diagram also indicates how context (like `merchantID` and perhaps a snippet of recent history) is passed along, and how the system ensures the answer is grounded in the database results. Every step is asynchronous and non-blocking where possible, courtesy of FastAPI's async support (for instance, the DataTools call could be awaited while other requests are handled in parallel).

Use Case Diagram

Below is a use-case representation of how a Grab merchant interacts with the MEX Assistant and what functionalities the system provides:

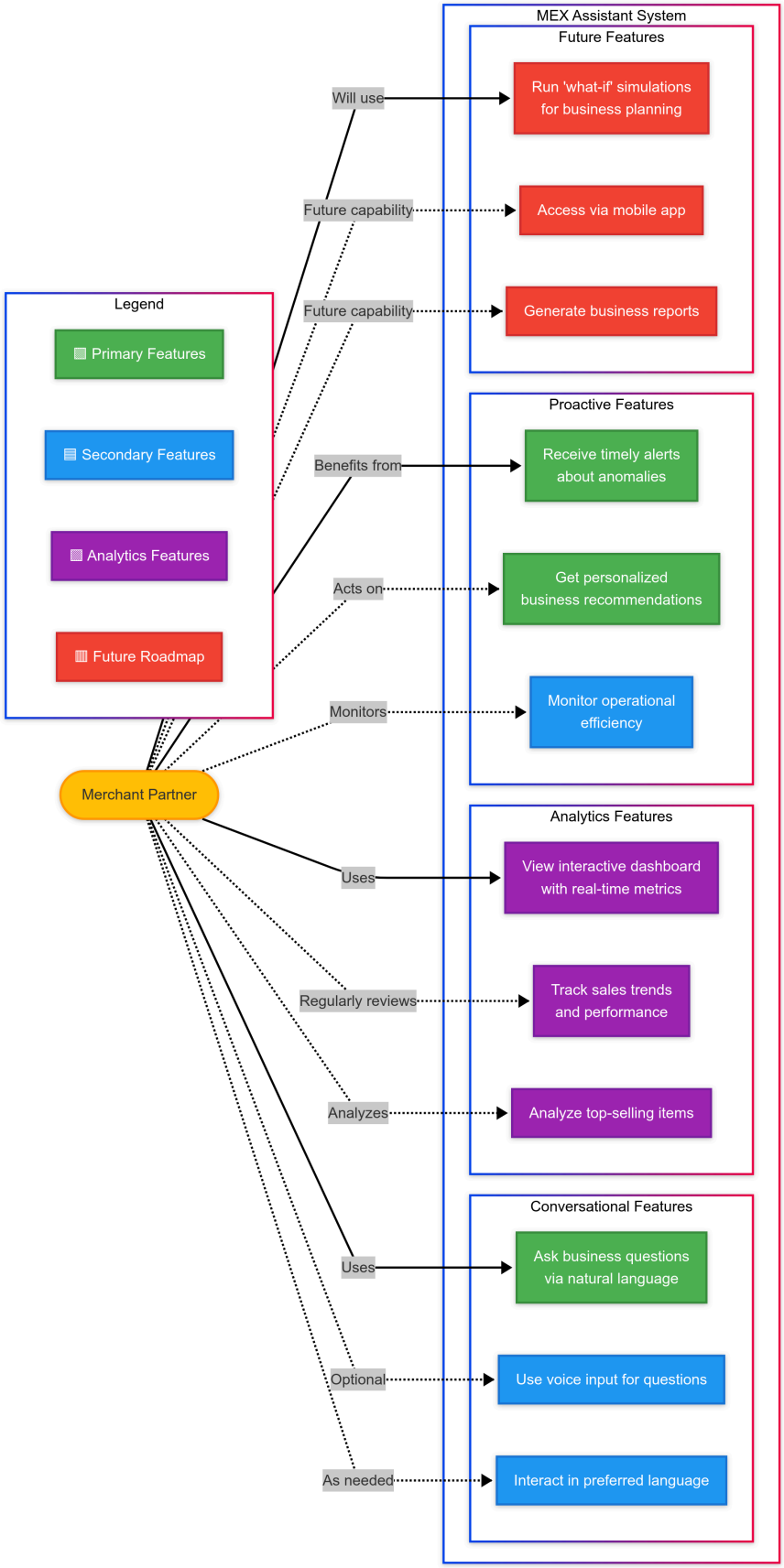
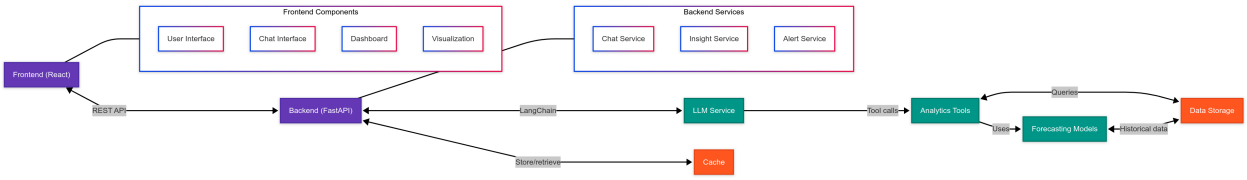

MEX Assistant System
Future Features
Run 'what-if' simulations for business planning
Access via mobile app
Generate business reports
Proactive Features
Receive timely alerts about anomalies
Get personalized business recommendations
Monitor operational efficiency
Analytics Features
View interactive dashboard with real-time metrics
Track sales trends and performance
Analyze top-selling items
Conversational Features
Ask business questions via natural language
Use voice input for questions
Interact in preferred language
Will use
Future capability
Future capability
Benefits from
Acts on
Monitors
Uses
Regularly reviews
Analyzes
Uses
Optional
As needed
Merchant Partner
Legend
Primary Features
Secondary Features
Analytics Features
Future Roadmap

Diagram Description: The **Merchant Partner** (user) can interact with the **MEX Assistant System** in multiple ways. **Use Case UC1**: The merchant asks questions via the chat interface (e.g., inquire about sales figures, best-selling items, etc.) and receives answers. **UC2**: The merchant views the dashboard to see key analytics at a glance (graphs, stats updated in real-time). **UC3**: The merchant benefits from proactive alerts/insights that the system sends without prompt (for example, a notification of an unusual drop in sales or an upcoming stockout risk). **UC4**: The merchant can run "what-if" scenario simulations (asking hypothetical questions to forecast outcomes). This diagram highlights the primary ways the system delivers value to the user: on-demand insights, visual analytics, proactive guidance, and scenario planning.

Component Interaction Diagram



Impact & Usefulness

The MEX Assistant is poised to create significant positive impact for Grab merchants:

- **Democratizing Data Insights:** It turns mountains of data into accessible advice. Merchants no longer need to be analytics experts – a simple question in their own language yields insights that would otherwise require a data analyst. This empowerment means even small family-run businesses can make informed decisions quickly.
- **Real-Time Decision Support:** By providing insights instantly and in real-time, the assistant enables merchants to react to issues or opportunities on the fly. For example, if lunchtime sales are dipping, the merchant can take action that very day (run a quick promo or alert staff) rather than finding out weeks later in a report. Fast feedback loops can improve business agility and revenue.

- **Personalized Coaching at Scale:** Each merchant gets personalized tips as if they had a business coach monitoring their performance. Whether it's recommending a popular item to add, pointing out a costly delay in deliveries, or congratulating them on a new monthly record, the assistant delivers a level of personalized attention typically not available to tens of thousands of small merchants. This can increase merchant satisfaction and loyalty to the platform.
- **Proactive Problem Prevention:** The proactive alerts help catch problems early. For instance, detecting a preparation delay trend before it leads to bad customer reviews, or warning about inventory running low on a popular item before it stocks out. Such timely interventions can improve end-customer experience (fewer canceled orders, faster deliveries) and thus the merchant's ratings and sales.
- **Multilingual Inclusivity:** By speaking the merchant's language, the tool ensures inclusivity. Merchants who are more comfortable in Thai or Bahasa Melayu get the same quality of insight as those who speak English, eliminating language barriers to using tech tools.
- **Alignment with Grab's Goals:** This solution directly supports Grab's mission of economic empowerment. It gives merchants an edge in optimizing their business, thus potentially increasing their earnings and success. By combining analyses across sales, operations, and customer behavior, the assistant provides **accurate, timely, and actionable** guidance, perfectly aligning with Grab's goal of instant trend identification and timely alerts for merchants. In the long run, widespread use of such an assistant could uplift the overall quality of service on the platform (as merchants become more efficient and customer-centric).
- **Potential Metrics of Success:** We anticipate measurable improvements if MEX Assistant is deployed. For example, merchants using it might see a X% increase in weekly sales (due to better inventory management and promotions), a Y% reduction in late orders (thanks to operational alerts), or a higher retention rate on the platform. Even qualitative feedback like "I feel more in control of my business" would validate the usefulness. In essence, the assistant's impact is twofold: **tangible business performance gains and intangible peace-of-mind** for merchants knowing AI has their back.

In summary, MEX Assistant isn't just a cool tech demo – it's a genuinely useful tool that can change how merchants run their businesses daily. By making advanced analytics *approachable*, it empowers merchants to act on insights that would otherwise remain hidden in spreadsheets. This kind of impact is what will impress judges (showing that our solution is not only technically sound but also meaningfully improves users' lives) and, more importantly, delight merchant-partners in the real world.

Conclusion

In this proposal, we presented **MEX Assistant** as a compelling solution to help Grab's merchant-partners leverage AI for economic empowerment. We began by identifying the challenges merchants face with data overload, and we turned those into an opportunity for an AI-driven conversational platform. Our solution marries a user-friendly chat interface with powerful analytics, delivering insights in a way that is proactive, personalized, and easy to understand.

From a technical perspective, our approach is ambitious yet grounded in feasibility. We combined state-of-the-art language AI (GPT-4 via LangChain) with proven data analytics methods (time-series forecasting, clustering, association rules) to create a **hybrid AI system** greater than the sum of its parts. The architecture we implemented – a React frontend and FastAPI backend – is modern, modular, and scalable, indicating that this prototype can evolve into a production system. We paid special attention to multilingual support and localization, ensuring the solution is region-ready for Southeast Asia from day one. The proposal also detailed an extension into scenario simulation, showing that we have a clear vision for future capabilities beyond the hackathon requirements.

Our goal was not only to build a working prototype, but to design a solution that *lasts* – one that could be deployed and make a real impact. We believe MEX Assistant achieves this by addressing real merchant pain points and aligning with Grab's vision. It's an innovative approach (conversational analytics in multiple languages), but also a practical one (using readily available tools and models, with a clear path to implementation). This balance of innovation and practicality strengthens the case for MEX Assistant.

In conclusion, MEX Assistant stands as a demonstration of how AI can transform business intelligence for small merchants. It turns data into dialogue, analyses into action, and confusion into clarity. We are confident that this project, with its motivating vision and solid technical underpinnings, will impress the hackathon judges and inspire them with its potential. More importantly, we are excited about the road ahead – refining this assistant and seeing it empower real merchants across Southeast Asia. MEX Assistant isn't just a hackathon project; it's a blueprint for the next generation of merchant support tools on the Grab platform.

Advanced Extension: Scenario Simulation (What-if Forecasting)

For a truly cutting-edge extension, we propose a **Scenario Simulation** feature that enables merchants to ask "what-if" questions and receive forecasted outcomes. This goes beyond reacting to data – it helps merchants plan for the future by exploring hypothetical situations in a risk-free manner.

What-if Scenarios Examples: A merchant could ask:

- *"What if I increase the price of my best-selling item by 10%? How might that affect my weekly revenue?"*
- *"What if I launch a 20% discount promo next Monday – how many extra orders could I expect based on past data?"*
- *"If the weather is rainy all next week, what will be the impact on my sales (given historically rain slows orders)?"*

In scenario simulation mode, the assistant would take such a hypothetical input, adjust the relevant parameters in its models, and produce a forecast or outcome for the merchant.

Implementation Approach: We leverage our forecasting model and data patterns to enable this:

- The time-series forecasting model (Prophet/ARIMA) we trained on historical sales can project future sales given recent trends. For a what-if scenario, we adjust the input to this model. For instance, to simulate a price increase, we might decrease the forecasted demand slightly based on an estimated price

elasticity (or we compare with similar items' data to infer the impact). For a promotion scenario, we might increase expected order count for the promo period based on past promotions.

- Our association rules and item popularity data can help estimate cross-effects. If the scenario is adding a new item or combo, we can find analogous historical data (e.g., when a similar item was introduced by the merchant or by a peer) to simulate its effect on sales.
- We would create a **"simulation tool"** accessible to the LLM agent, such as `simulate_scenario(parameters)` which encapsulates the logic of applying changes to the forecast. The parameters could include things like *price change*, *discount percentage*, *added cost*, *expected new customer traffic*, etc., depending on the scenario.
- The LLM will integrate the simulation output into a conversational answer. For example: *"According to my calculations, a 10% price increase on Item X might reduce its sales slightly, but overall weekly revenue could still increase by ~5%. Specifically, you'd earn an extra RM200 next week compared to the current pricing."* The assistant would couch the forecast with appropriate caveats, explaining assumptions (to maintain trust and transparency).

UI/UX for Simulation: In the chat interface, scenario questions would be handled like any other query. Additionally, we could offer a more guided UI for common scenarios – e.g., a "Scenario Simulator" panel where a merchant moves sliders (for price, promotion, etc.) and the assistant generates the outcome. This could be an extension of the dashboard: a sandbox mode where changes don't affect real settings but show predicted outcomes. It's a powerful visual tool for planning (e.g., forecasting charts that update based on scenario inputs in real-time).

Value of Scenario Simulation: This feature turns MEX Assistant from a reactive advisor into a forward-looking strategist for the merchant. Merchants can virtually "test" business decisions before committing to them. This reduces risk and encourages data-driven experimentation. Over time, as the system collects data on actual outcomes of changes, it can learn and refine its simulation accuracy (e.g., learning the true elasticity of demand for their products, improving future predictions).

Feasibility and Future Work: While full scenario simulation is advanced, we laid the groundwork with our forecasting and analytics models. The next step would be to gather more data on how certain factors (price, promotions, external events) historically affect sales. With more time, we'd incorporate machine learning models that can accept scenario variables as input. For example, a regression model might predict sales given features like "promotion = yes/no" or "price change = X%". During the hackathon, we demonstrate a simplified scenario (perhaps just forecasting next week normally, which is already a form of scenario: "status quo" projection). Post-hackathon, we would iterate to support interactive variables.

In summary, the scenario simulation extension showcases the forward-thinking potential of MEX Assistant. It transforms the assistant into not just an analyst, but a business planning aide. This level of foresight and interactivity would truly differentiate the platform – imagine a merchant virtually trying out a new strategy and immediately seeing a forecast of results. It brings the power of predictive analytics and "digital twin" experimentation into the hands of everyday entrepreneurs. By implementing this, Grab can pioneer a new kind of merchant support tool that helps users **not only understand the past and present but also prepare for the future** – all through a familiar conversational experience.