PMs onboarding

Tami 2 – Technical Overview For PMs

A concise, structured map of the system architecture, core flows, and supporting features.

---

1. Core Flows

1.1 Meeting Lifecycle

One-liner:
A meeting starts as audio, becomes a cleaned transcript, then generates a summary, action items, embeddings, entities, and graph relationships.

```
flowchart LR
  A[Create session] --> B[Upload audio (Supabase Storage)]
  B --> C{Language routing}
  C -->|Hebrew| D[RunPod Ivrit ASR async]
  C -->|English| E[OpenAI Whisper sync]
  D --> F[Save transcript + segments]
  E --> F[Save transcript + segments]
  F --> G[Deep refinement (GPT-4o) + mark deletions]
  G --> H[Auto summary + action items (GPT-4o-mini)]
  H --> I[Auto embeddings (text-embedding-3-small)]
  I --> J[Auto entities (GPT-4o-mini)]
  J --> K[Auto relationships → Neo4j]
  K --> L[Session completed]
```

Code locations:

- Orchestration: `api/sessions/[id]/transcription-status/route.ts`
- ASR routing: `lib/transcription/service.ts`
- Ivrit ASR: `lib/transcription/ivrit.ts`

- Whisper ASR: `lib/transcription/whisper.ts`
- Refinement: `lib/transcription/refinement.ts`
- Auto summary: `lib/ai/auto-summary.ts`
- Embeddings: `lib/ai/embeddings.ts`
- Entities: `lib/ai/entities.ts`
- Relationships: `lib/ai/relationships.ts`

1.2 Session Q&A (RAG per meeting)

One-liner:
A question is embedded, matched against that meeting's embeddings, and answered using the best context chunks.

```
sequenceDiagram
  participant U as User
  participant API as /api/sessions/:id/chat
  participant DB as Supabase (memory_embeddings)
  participant AI as OpenAI (gpt-4o-mini)

  U->>API: Ask question
  API->>AI: Embed question
  API->>DB: match_embeddings (top-k)
  API->>AI: Answer with context + transcript fallback
  API->>DB: Save chat_messages
  API->>U: Final answer
```

Code: `api/sessions/[id]/chat/route.ts`

1.3 Global Memory Chat (cross-meeting Q&A)

One-liner:
The user asks a question across all meetings; the system retrieves top-K chunks from the global vector store and answers with citations.

```
flowchart LR
  A[Question] --> B[Embed]
  B --> C[search_embeddings (global)]
  C --> D[Group by session]
  D --> E[Generate answer + sources]
  E --> F[Store memory_chat_messages]
```

Code: `api/memory/chat/route.ts`

1.4 Attachments → Memory Embeddings

One-liner:
Uploaded files are parsed, chunked, embedded, and added to the unified vector store.

```
flowchart LR
  A[Upload attachment] --> B[Store in Supabase]
  B --> C[Parse (PDF/DOC/Excel/TXT)]
  C --> D[Chunk + embed]
  D --> E[Save to memory_embeddings]
```

Code:

- Upload: `api/sessions/[id]/attachments/route.ts`
- Processing: `lib/ai/document-processor.ts`

1.5 Entities & Knowledge Graph

One-liner:
Entities and relationships extracted from transcripts are stored in Supabase and Neo4j for graph views and dedupe.

```
flowchart LR
  A[Transcript] --> B[Extract entities]
  B --> C[Save entities + mentions]
  C --> D[Extract relationships]
  D --> E[Save to Neo4j]
  E --> F[Graph visualization + dedupe]
```

Code:

- Entities: `api/sessions/[id]/entities/route.ts`
- Relationships: `api/sessions/[id]/relationships/route.ts`
- Graph APIs: `api/graph/*`
- Neo4j client: `lib/neo4j/client.ts`

---

2. Product Features (One-Liners)

- Sessions: Create, list, and update meetings.
- Transcript segments: Speaker-aligned segments with timestamps.
- Action items: Auto-generated + user-editable.
- Tags: Manual or auto-generated labels.
- Search (keyword): SQL search over transcript segments.
- Search (semantic): Vector search over embeddings.
- Exports: Summary and transcript to HTML/Markdown.
- Reprocessing: Re-run all pipeline stages.
- Speaker tools: Rename or merge speaker IDs.

- Attachments: Upload → extract text → embed.
- Knowledge graph: Neo4j relationships + dedupe.
- Account deletion: Fully cascaded delete.

---

3. Data Model Cheat Sheet (Supabase)

- `sessions` – Meeting metadata.
- `transcripts` – Full transcript per session.
- `transcript_segments` – Speaker-timed segments.
- `summaries` – Overview, key points, decisions.
- `action_items` – Summary-based tasks.
- `entities` – Canonical entities across sessions.
- `entity_mentions` – Occurrences of entities.
- `tags` / `session_tags` – Topic classification.
- `memory_embeddings` – Chunks + embeddings + metadata.
- `attachments` – Uploaded files + storage references.
- `chat_messages` – Per-session Q&A logs.
- `memory_chat_messages` – Global Q&A logs.

---

4. AI Services

- **Whisper (OpenAI):** English ASR.
- **Ivrit ASR:** Hebrew speech-to-text with diarization.
- **GPT-4o-mini:** Summaries, Q&A, entities, relationships.
- **GPT-4o:** Deep refinement + diarization cleanup.
- **text-embedding-3-small:** All vector search.

---

5. Search & Retrieval

- `match_embeddings`: Vector search scoped to a session.
- `search_embeddings`: Global multi-session vector search.
- Keyword search: SQL `ilike` on segments.

---

6. Knowledge Graph (Neo4j)

- Nodes: Person, Organization, Topic, etc.
- Edges: Relation type, confidence score, session source.
- Dedupe: Alias, fuzzy, semantic similarity, LLM fallback.
- API returns nodes/edges for UI visualization.

---

7. Operational / Infra Notes

- Next.js App Router for backend logic.
- Supabase: Auth, Postgres, RLS, storage, vector DB.
- Neo4j: Graph storage + analytics.
- Audio & attachments: Stored in Supabase buckets.
- RLS: All user data protected with `auth.uid()`.

---

8. Feature Landscape (Mind Map)

```
mindmap
  root((Tami 2))
    Core Flows
      Meeting lifecycle
      Transcribe + refine
      Summaries + action items
      Embeddings + memory
      Entities + relationships
    Retrieval
      Keyword search
      Semantic search
      Session Q&A
      Global memory chat
    Knowledge Graph
      Entities
      Relationships
      Dedupe
      Visualize
    Content
      Attachments
      Transcript segments
      Exports
    Admin/Utility
      Reprocess pipeline
```

`Speaker merge/rename`
`Account delete`

---

9. Key API Endpoints

- **Sessions**
  `GET/POST /api/sessions`
  `GET/PATCH /api/sessions/:id`

- **Transcription**
  `POST /api/sessions/:id/transcribe`
  `GET /api/sessions/:id/transcription-status`

- **Summary**
  `POST/PATCH /api/sessions/:id/summarize`

- **Q&A**
  `POST/GET /api/sessions/:id/chat`
  `POST /api/memory/chat`

- **Embeddings**
  `GET/POST/DELETE /api/sessions/:id/embeddings`

- **Entities / Relationships**
  `POST /api/sessions/:id/entities`
  `POST /api/sessions/:id/relationships`

- **Search**
  `GET /api/search`
  `POST /api/search/semantic`

- **Attachments**
  `POST /api/sessions/:id/attachments`

- **Speakers**
  `GET/PATCH/POST /api/sessions/:id/speakers`

- **Reprocess**
  `POST /api/sessions/:id/reprocess`

- **Export**
  `GET /api/sessions/:id/export`

Technical Onboarding for dev

Tami 2 – Technical Onboarding (Deep Dive)

Audience: new engineers (backend + frontend).
A structured, concise map of system architecture, pipelines, data model, and API surface.

---

1. System Overview

Tami ingests meeting audio, transcribes it (Hebrew/English), refines the transcript, generates summaries and action items, builds semantic memory embeddings, extracts entities and relationships, and exposes search + Q&A across meetings and attachments.

```
flowchart TB
  subgraph Client
    UI[Web UI]
  end
  subgraph Backend
    API[Next.js API Routes]
    AI[AI Pipeline]
  end
  subgraph Data
    PG[Supabase Postgres + pgvector]
    ST[Supabase Storage]
    GDB[Neo4j Graph]
  end
  UI --> API
  API --> PG
  API --> ST
  API --> AI
  AI --> PG
  AI --> GDB
```

---

2. Backend

2.1 Major Services (one-liners)

- **API layer:** Next.js API routes exposing sessions, AI actions, search, and utilities.
- **Transcription routing:** Ivrit (Hebrew, async) or Whisper (English, sync).
- **Refinement:** deep cleanup and speaker correction.
- **Summarization:** overview, key points, decisions, action items.
- **Embeddings:** chunking + vector storage for semantic search.
- **Entities:** extract people/orgs/projects/topics + mentions.
- **Relationships:** write to Neo4j knowledge graph.
- **Search:** keyword search (SQL) + vector search (pgvector).

2.2 AI Models

- **whisper-1:** English transcription.
- **Ivrit ASR:** Hebrew transcription + diarization (RunPod).
- **gpt-4o:** deep transcript refinement.
- **gpt-4o-mini:** summaries, Q&A, entities, relationships, light refinement.
- **text-embedding-3-small:** embeddings for transcript + attachments.

2.3 Core Pipelines

A) Meeting Transcription & Enrichment

```
sequenceDiagram
  participant UI as Client
  participant API as Backend
  participant ASR as ASR Provider
  participant AI as LLM
  participant DB as Postgres
  participant KG as Neo4j

  UI->>API: Start transcription
  API->>ASR: Transcribe (Ivrit async | Whisper sync)
  ASR-->>API: Transcript + segments
  API->>DB: Save transcript + segments
  API->>AI: Deep refinement
  API->>DB: Save refined transcript
  API->>AI: Generate summary + action items
  API->>DB: Save summary data
  API->>AI: Generate embeddings
  API->>DB: Save embeddings
  API->>AI: Extract entities + relationships
  API->>DB: Save entities + mentions
  API->>KG: Save relationships
```

B) Per-Meeting Q&A (RAG)

```
flowchart LR
  Q[Question] --> E[Embed]
  E --> R[Vector search (session)]
  R --> A[Answer with GPT + context]
  A --> S[Store chat message]
```

C) Global Memory Chat (Cross-Meeting)

```
flowchart LR
  Q[Question] --> E[Embed]
  E --> R[Vector search (global)]
  R --> G[Group by session]
  G --> A[Answer + citations]
```

```
A --> S[Save memory chat]
```

2.4 Data Model (Supabase)

Core Tables

- **sessions**: meeting container.
- **transcripts**: one transcript per session.
- **transcript_segments**: timestamped, speaker-aligned text.
- **summaries**: overview + key points + decisions.
- **action_items**: tasks derived from summaries.
- **entities**: canonical people/orgs/projects/topics.
- **entity_mentions**: mapping of entities → sessions.
- **tags / session_tags**: manual + auto tagging.
- **memory_embeddings**: transcript/attachment vectors.
- **attachments**: files + storage references.
- **chat_messages**: per-session Q&A.
- **memory_chat_messages**: global Q&A.

Key Indexes

- **pgvector HNSW** on memory_embeddings.embedding.
- Standard user/session indexes for RLS performance.

2.5 Knowledge Graph (Neo4j)

- **Nodes:** Person, Organization, Project, Topic, etc.
- **Edges:** WORKS_AT, MANAGES, RELATED_TO, USES, etc.
- **Use cases:** graph visualization, navigation, deduping.

2.6 API Surface (Grouped)

Sessions

- `GET /api/sessions`
- `POST /api/sessions`
- `GET /api/sessions/:id`
- `PATCH /api/sessions/:id`

Transcription

- `POST /api/sessions/:id/transcribe`
- `GET /api/sessions/:id/transcription-status`

Refinement

- `POST /api/sessions/:id/refine`
- `DELETE /api/sessions/:id/refine`

Summaries

- `POST /api/sessions/:id/summarize`

- `PATCH /api/sessions/:id/summarize`

Action Items

- `GET /api/sessions/:id/action-items`
- `POST /api/sessions/:id/action-items`
- `PATCH /api/sessions/:id/action-items/:itemId`

Embeddings & Memory

- `GET /api/sessions/:id/embeddings`
- `POST /api/sessions/:id/embeddings`
- `DELETE /api/sessions/:id/embeddings`
- `POST /api/memory/chat`

Q&A + Search

- `POST /api/sessions/:id/chat`
- `GET /api/sessions/:id/chat`
- `GET /api/search`
- `POST /api/search/semantic`

Entities & Relationships

- `GET /api/sessions/:id/entities`
- `POST /api/sessions/:id/entities`

- `GET /api/sessions/:id/relationships`
- `POST /api/sessions/:id/relationships`

Graph APIs

- `GET /api/graph/entities`
- `POST /api/graph/entities`
- `GET /api/graph/visualize`
- `GET /api/graph/entities/duplicates`
- `POST /api/graph/entities/:id/merge`
- `GET /api/graph/entities/:id/similar`
- `GET /api/graph/relationships`

Tags

- `GET /api/tags`
- `POST /api/tags`
- `GET /api/sessions/:id/tags`
- `POST /api/sessions/:id/tags`
- `DELETE /api/sessions/:id/tags`

Attachments

- `GET /api/sessions/:id/attachments`

- `POST /api/sessions/:id/attachments`
- `GET /api/sessions/:id/attachments/:attachmentId`
- `DELETE /api/sessions/:id/attachments/:attachmentId`

Utilities

- `POST /api/sessions/:id/reprocess`
- `GET /api/sessions/:id/export`
- `GET /api/sessions/:id/adjacent`
- `GET/PATCH /api/sessions/:id/speakers`
- `POST /api/sessions/:id/speakers/merge`
- `PATCH /api/sessions/:id/segments/bulk`
- `DELETE /api/user/delete`

---

3. Frontend

3.1 Architecture (one-liners)

- Next.js App Router pages for sessions, memory, and entities.
- UI built with Radix primitives + utility-layer styling.
- Full Hebrew RTL + English LTR support.
- Server-side data loading for core pages; client-side for actions.

3.2 User Journeys (one-liners)

- Create a meeting: start session → upload audio → track processing.
- Review: transcript, summary, speakers, action items, attachments.
- Q&A: per-meeting or global memory chat.
- Explore: entities + knowledge graph.
- Organize: tags, search, export.

3.3 UI Modules

- **Meetings dashboard**
- **Meeting detail:** transcript, summary, action items, speakers, attachments, chat
- **Memory:** global Q&A + semantic search
- **Entities:** table view + graph view
- **Search:** keyword + semantic
- **Auth & settings**

3.4 Frontend Packages

- React 19 + Next.js 16
- Radix UI / shadcn components
- Tailwind CSS

- next-intl (i18n + RTL)
- sonner (toasts)
- react-force-graph-2d

---

4. Security & Access

- Supabase Auth for authentication.
- Full RLS enforcing user ownership on all tables.
- API routes validate ownership before mutations.

---

5. Operational Notes

- Hebrew transcription is async → requires polling.
- Deep refinement is expensive → executed after ASR.
- Auto-summary, auto-embeddings, auto-entities run post-transcription.
- Reprocessing endpoints exist for manual reruns.

---

6. Mental Model (one-liner)

Tami converts meetings into structured, persistent memory: audio → trusted transcript → enriched knowledge → searchable over time.
