EMMA API – Swagger Testing Guide (Grouped by Area)

*This guide organizes EMMA API endpoints by functional area and explains each endpoint’s purpose, inputs, outputs, dependencies, and database schema attributes. Use it to drive step‑by‑step Swagger testing before validating web-form UX flows.*

# Suggested Swagger Test Order

1. Mint a dev token: POST /api/auth/dev-token (Development only)
2. Check health endpoints (optional sanity checks)
3. Register/verify via invitation (if testing onboarding)
4. Create or identify Organization context (orgId in JWT or returned from dev-token)
5. Create a Contact: POST /api/Contact
6. Get Contact by ID: GET /api/Contact/{id}
7. List Contacts by Org: GET /api/Contact?orgId={orgId}
8. Assign Contact to a user: PUT /api/Contact/{id}/assign
9. Create an Interaction for a Contact: POST /api/contacts/{contactId}/interactions
10. List Interactions for a Contact: GET /api/contacts/{contactId}/interactions
11. Agent follow-up suggestion (dev stub): POST /api/agent/suggest-followup
12. Enum lookup for dynamic values: GET /api/enums/{type}

# Authentication & Account

## Dev Auth – POST /api/auth/dev-token

**Purpose:** Issue a development JWT for local testing; optionally returns orgId/userId in payload.

**Input:** Body (optional): { orgId?: Guid, userId?: Guid, email?: string }

**Output:** JSON: { token, orgId, userId, email }

**Dependencies:** Development environment only; no auth required to call.

**Schema Attributes:** No direct DB writes; used to populate Authorization header and orgId claim.

## Account – GET /api/Account/profile

**Purpose:** Return current user profile/org. Anonymous allowed; uses identity if authenticated.

**Input:** None

**Output:** User/Organization context (shape may vary by environment).

**Dependencies:** Optional Bearer token to enrich context.

**Schema Attributes:** Read-only access paths; no direct writes.

## Verification – POST /api/auth/verify-email

**Purpose:** Verify email address via a verification token; idempotent if already verified.

**Input:** Body: { "token": "..." }

**Output:** 204 No Content on success; ProblemDetails on error.

**Dependencies:** Anonymous call; relies on a valid verification token.

**Schema Attributes:** Updates user verification status fields.

# Organization / Onboarding

## Register from Invitation – POST /api/Organization/invitations/{token}/register

**Purpose:** Register a user from an invitation token and (dev stub) send verification email.

**Input:** Path: token; Body: RegisterFromInvitationDto { firstName?, lastName?/fullName?, password? }

**Output:** 204 No Content on success; ProblemDetails on revoked/expired/not found/validation.

**Dependencies:** Public token-gated call; invitation must be valid and not revoked/expired.

**Schema Attributes:** Creates/updates User and Organization membership; writes verification state.

# Contacts

## Create Contact – POST /api/Contact

**Purpose:** Create a new contact under an organization.

**Input:** Body: ContactCreateDto { organizationId: Guid, firstName: string, lastName: string, ownerId?: Guid, ... }

**Output:** Created Contact with Id, CreatedAt, UpdatedAt.

**Dependencies:** Bearer (VerifiedUser). orgId claim must match OrganizationId; organization must exist.

**Schema Attributes:** Table: Contacts (Id, OrganizationId, FirstName, LastName, RelationshipState, OwnerId, CreatedAt, UpdatedAt, ...). Indexes: IX\_Contacts\_OrganizationId, IX\_Contacts\_RelationshipState, etc.

## Get Contact – GET /api/Contact/{id}

**Purpose:** Retrieve a contact by id within org scope.

**Input:** Path: id (Guid)

**Output:** Full Contact

**Dependencies:** Bearer (VerifiedUser). orgId claim must match contact’s OrganizationId.

**Schema Attributes:** Reads Contacts; may include navigation to Interactions, Assignments, Collaborators.

## List Contacts – GET /api/Contact?orgId={orgId}

**Purpose:** List contacts for an organization (supports filtering/pagination via query).

**Input:** Query: orgId (Guid) (must match JWT claim); optional paging/filter params.

**Output:** List<Contact> with pagination metadata.

**Dependencies:** Bearer (VerifiedUser). orgId claim validation enforced.

**Schema Attributes:** Reads Contacts by OrganizationId; leverages indexes on OrganizationId/IsActiveClient/etc.

## Assign Contact – PUT /api/Contact/{id}/assign

**Purpose:** Assign a contact to a specific user (owner/agent).

**Input:** Path: id (Guid); Body: ContactAssignDto { "userId": Guid }

**Output:** 204 No Content (typical) or updated contact/assignment state.

**Dependencies:** Bearer (VerifiedUser). User and Contact must share organization context.

**Schema Attributes:** Updates Contacts.OwnerId or related assignment table columns and UpdatedAt.

# Contact Collaborators (Placeholders – 501)

## Create Collaborator – POST /api/contacts/{contactId}/collaborators

**Purpose:** Placeholder for granting collaboration access to a contact (returns 501).

**Input:** Path: contactId; Body: collaborator details (TBD).

**Output:** 501 Not Implemented.

**Dependencies:** N/A (scaffolding).

**Schema Attributes:** Future: ContactCollaborator table with role/permissions columns.

## List Collaborators – GET /api/contacts/{contactId}/collaborators

**Purpose:** Placeholder for listing collaborators (returns 501).

**Input:** Path: contactId

**Output:** 501 Not Implemented.

**Dependencies:** N/A

**Schema Attributes:** Future: reads ContactCollaborator.

# Interactions

## Create Interaction – POST /api/contacts/{contactId}/interactions

**Purpose:** Log a communication event (email, call, sms, note, meeting, task) for a contact.

**Input:** Path: contactId (Guid); Body: { type, direction, subject?, content?, consentGranted?, occurredAt }

**Output:** Created Interaction entry

**Dependencies:** Bearer (VerifiedUser). Contact and JWT orgId must align. AI analysis currently disabled in this path.

**Schema Attributes:** Table: Interactions (Id, ContactId, OrganizationId, Type, Direction, Timestamp/OccurredAt, AgentId, Content, Channel, Status, Tags, CreatedAt, etc.). Indexes: IX\_Interactions\_ContactId, IX\_Interactions\_OrganizationId, IX\_Interactions\_Timestamp, composite Type/Status.

## List Interactions – GET /api/contacts/{contactId}/interactions

**Purpose:** Return all interactions for a specific contact (scoped by org).

**Input:** Path: contactId (Guid)

**Output:** List<Interaction>

**Dependencies:** Bearer (VerifiedUser). orgId claim required; results filtered by it.

**Schema Attributes:** Reads Interactions by ContactId/OrganizationId.

# Agent / Orchestration (Dev Stub)

## Suggest Follow-up – POST /api/agent/suggest-followup

**Purpose:** Development-only stub returning follow-up suggestion payload given optional contact/context.

**Input:** Body (optional): { contactId?: Guid, context?: string }

**Output:** Suggestion payload for UI testing (shape may vary).

**Dependencies:** Bearer (VerifiedUser); Development environment only.

**Schema Attributes:** No direct DB writes; used to exercise orchestration/LLM pathways in dev.

# Enums

## Get Enum Values – GET /api/enums/{type}

**Purpose:** Return dynamic enum values by type (Cosmos-backed with JSON fallback).

**Input:** Path: type (e.g., PlanType, RelationshipState, etc.)

**Output:** Array of enum values (key/value/displayName/metadata).

**Dependencies:** Anonymous allowed (by design); may be tightened in production.

**Schema Attributes:** Reads from Cosmos-backed config store or JSON fallback; no relational writes.

# Health & Diagnostics

## Health – GET /api/health

**Purpose:** Basic liveness probe for the API process.

**Input:** None

**Output:** 200 OK with health info

**Dependencies:** None

**Schema Attributes:** None

## Cosmos Health – GET /api/health/cosmos

**Purpose:** Connectivity test to Azure Cosmos DB.

**Input:** None

**Output:** 200 OK if reachable; ProblemDetails on failure.

**Dependencies:** Cosmos DB configuration must be valid.

**Schema Attributes:** Read test item/metadata from Cosmos.

## Postgres Health – GET /api/health/postgres

**Purpose:** Connectivity test to PostgreSQL.

**Input:** None

**Output:** 200 OK if reachable; ProblemDetails on failure.

**Dependencies:** PostgreSQL connection string must be valid.

**Schema Attributes:** Executes trivial read query.

## Cosmos Sample Item – GET /api/health/cosmos/item

**Purpose:** Reads a development sample item for quick verification.

**Input:** None

**Output:** Sample item content

**Dependencies:** Cosmos DB must be accessible; sample item present.

**Schema Attributes:** Reads single item by known id/partition.

# Appendix – Standardized Error Shape (RFC7807 ProblemDetails)

All error responses use application/problem+json with consistent fields (type, title, status, detail).