DEVX360

Sipho Sehlapelo
Kesley Hamann
Sibusiso Mngomezulu
Lwando Msindo
David Musa-Aisien

DevX360

Contents

API SERVICE CONTRACTS

This document defines the service contracts for the DevX360 platform, outlining the expected behavior, inputs, outputs, and error handling for each API endpoint. The contracts serve as a formal agreement between the backend services and their consumers (frontend applications, external systems, or other services), ensuring consistency, reliability, and maintainability of the system.

Each service contract includes the endpoint URL, HTTP method, request parameters, authentication requirements, response structure, and potential errors.

Service contracts

Common types / shared info

- Auth: cookie token (HTTP-only) created by generateToken. All authenticated routes expect cookie token.
- Common error shape:

```json
{
  "message": "A short description",
  "error": "optional technical message",
  "suggestion": "optional user-facing suggestion",
  "details": {}
}
```

- ObjectId validation: endpoints using teamId or id validate mongoose.Types.ObjectId.
- Rate limiting: global limiter configured (15min window, max 100 requests per IP).

- CORS: only allowed origins (localhost test origins included).
- File uploads: multipart/form-data, avatar file field, using multer storing files in uploads/.

Health check

Method: GET
Path: /api/health
Auth: none

Request: none

Success (200):

```json
{
  "status": "OK",
  "database": "Connected" | "Disconnected",
  "ollama": "Operational" | "Unavailable",
  "timestamp": "ISO8601 string"
}
```

Errors (500):

- Unexpected server exception

{ "status": "Degraded", "error": "message", "timestamp": "ISO8601" }

Side effects: none

Register (create new user)

Method: POST
Path: /api/register
Auth: none

Request (application/json):

```json
{
  "name": "string",
  "email": "string",
  "password": "string",
  "role": "string (optional)",
  "inviteCode": "string (optional)"
}
```

Validation:

- name, email, password required.
- password minimum length 6.
- email normalized to lowercase.

Success (201):

```json
{
  "message": "Registration successful",
  "user": {
    "_id": "ObjectId",
    "name": "string",
    "email": "string",
    "role": "string",
    "inviteCode": "string|null",
    "isEmailVerified": true,
    // password NOT returned
  }
}
```

Side effects:

- Creates User document (password hashed).
- Sets cookie token (httpOnly, SameSite Lax, secure in prod).

Errors:

- 400 — missing fields, password too short, or email already exists.
- 500 — internal server error.

Idempotency: not idempotent (creates new user).

Start GitHub OAuth

Method: GET
Path: /api/auth/github
Auth: none

Request: none

Success: Redirect to GitHub OAuth URL:
302 ->
https://github.com/login/oauth/authorize?client_id=...&scope=user

Errors: none (unless env misconfigured).

Side effects: none.

GitHub OAuth callback

Method: GET
Path: /api/auth/github/callback
Auth: none (GitHub provides code)

Query params:

- code (required)

Behavior / side effects:

- Exchanges code for access_token with GitHub.
- Fetches GitHub user profile.
- Locates existing User by githubId, githubUsername, or email OR creates new User.
    - Created user gets random hashed password and isEmailVerified: true.
- Sets cookie token for the created/found user.
- (Optional) Update existing user fields (githubId, githubUsername).

Success: sets cookie and redirects to frontend or returns success (current code sets cookie; redirection commented out).

Errors:

- Redirects to ${FRONTEND_URL}/login?error=github_auth_failed on OAuth errors.
- 500 for server errors.

Notes: ensure FRONTEND_URL, GITHUB_CLIENT_ID, GITHUB_CLIENT_SECRET configured.

Login

Method: POST
Path: /api/login
Auth: none

Request (application/json):

```json
{ "email": "string", "password": "string" }
```

Success (200):

```json
{
  "message": "Login successful",
  "user": { /* user object without password */ }
}
```

- Sets cookie token with 7 days expiry.

Errors:

- 400 — missing email/password
- 401 — invalid email or invalid password
- 500 — server error

Side effects: updates user.lastLogin timestamp.

Get profile

Method: GET
Path: /api/profile
Auth: required (cookie token)

Request: none

Success (200):

```json
{
  "user": {
    "_id": "ObjectId",
    "name": "string",
    "email": "string",
    "role": "string",
    "avatar": "/uploads/filename.jpg" | undefined,
    "teams": [
      {
        id: "ObjectId",
        name: "string",
        creator: { "_id": "...", "name": "..." },
        members: [{ "_id":"...","name":"...","email":"..." }],
        doraMetrics: { /* metrics object or null */ },
        repositoryInfo: { /* repositoryInfo object or null */ }
      }
    ]
  }
}
```

Errors:

- 404 — user not found
- 500 — internal server error

Side effects: none

Update profile

Method: PUT
Path: /api/profile
Auth: required

Request (application/json) — at least one field required:

{ "name": "string (optional)", "email": "string (optional)", "password": "string (optional)" }

Validation:

- If email provided — ensure uniqueness (excluding user themself).
- password minimum length 6 if provided.

Success (200):

```json
{
  "message": "Profile updated successfully",
  "user": { /* updated user without password */ }
}
```

Errors:

- 400 — no fields provided, or email conflict, or password too short
- 404 — user not found
- 500 — server error

Side effects: updates User document.

Idempotency: The same update call is idempotent.

Get all users (admin-only)

Method: GET
Path: /api/users
Auth: required; role === "admin"

Request: none

Success (200):

```json
{ "users": [ /* users list without passwords */ ] }
```

Errors:

- 403 — admin access required
- 500 — server error

Delete user by ID (admin-only)

Method: DELETE
Path: /api/users/:id
Auth: required; role === "admin"

Path params:

- id (ObjectId)

Success (200):

```json
{ "message": "User deleted successfully" }
```

Errors:

- 400 — invalid user ID format
- 404 — user not found
- 403 — cannot delete admin user (safeguard)
- 500 — server error

Side effects: removes User doc.

Logout

Method: POST
Path: /api/logout
Auth: required

Success (200):

```json
{ "message": "Logged out" }
```

Side effects: clears cookie token.

Upload avatar

Method: POST
Path: /api/avatar
Auth: required
Content-Type: multipart/form-data (field name avatar)

Request:

- Body: file form field avatar (single file)

Success (200):

```json
{ "message": "Avatar uploaded", "avatarUrl": "/uploads/{filename}" }
```

Errors:

- 404 — user not found
- 500 — upload error

Side effects:

- Saves file to uploads directory.
- Deletes previous avatar file from disk (if exists).
- Updates user.avatar to filename.

Create team

Method: POST
Path: /api/teams
Auth: required

Request (application/json):

```
json

{ "name": "string", "password": "string", "repoUrl": "string" }
```

Validation:

- name, password, repoUrl required.
- name uniqueness enforced.

Success (201):

```
{
  "message": "Team created successfully",
  "team": { "id": "ObjectId", "name": "string", "repoUrl": "repo url" },
  "repositoryInfo": { /* repositoryInfo object */ }
}
```

Errors:

- 400 — missing fields, team exists
- 500 — repository analysis failed or DB error
    - On repo analysis failure returns:

{ "message":"Repository analysis failed", "error": "message", "suggestion": "Check repository accessibility..." }

- On validation / duplicate key returns specialized error responses (400 with details)

Side effects:

- Hashes and stores team password.
- Calls analyzeRepository(repoUrl) and getRepositoryInfo(repoUrl) — these may call GitHub and compute DORA metrics.
- Creates Team doc and RepoMetrics doc containing metrics and repositoryInfo.
- Triggers runAIAnalysis(team._id) asynchronously (setTimeout 0) to perform AI analysis.

Idempotency: not idempotent.

Join team

Method: POST
Path: /api/teams/join
Auth: required

Request (application/json):

```
{ "name": "string", "password": "string" }
```

Success (200):

```
{ "message": "Joined team", "teamId": "ObjectId" }
```

Errors:

- 404 — team not found
- 401 — incorrect password
- 500 — joining error

Side effects:

- Adds user to team.members if not already present.
- If user has githubUsername and team has RepoMetrics.repositoryInfo.url, calls collectMemberActivity(owner, repo, githubUsername) and stores stats in repoData.memberStats keyed by userId.

Notes: membership join attempts log user info for debugging.

Check team membership

Method: GET
Path: /api/teams/:teamId/membership
Auth: required

Path params:

- teamId (ObjectId)

Success (200):

```
json

{ "isMember": true|false }
```

Errors:

- 400 — invalid team ID format
- 404 — team not found
- 500 — internal error

Team search

Method: GET
Path: /api/teams/search
Auth: required

Query params:

- q (required) — search string

Success (200):

{ "results": [{ "name": "string", "creator": ObjectId, "members": [ObjectId] }] }

Errors:

- 400 — q missing
- 500 — internal server error

Notes: case-insensitive regex search on name.

Team details (RBAC)

Method: GET
Path: /api/teams/:name
Auth: required
Middleware: authorizeTeamAccess (ensures membership and attaches req.team, sets req.user.teamRole)

Path params:

- name (team name)

Success (200): two variants:

- **Creator / full permissions:**

```json
{
  "team": { "id": "ObjectId", "name": "string" },
  "doraMetrics": {...} | null,
  "repositoryInfo": {...} | null,
  "lastUpdated": "Date|null",
  "members": [ /* populated members (name,email) */ ],
  "creator": { "name": "..."},
  "memberStats": { /* per-user stats map */ },
  "permissions": "full"
}
```

- **Non-creator (read-only):**

```json
{
  "team": {...},
  "doraMetrics": {...} | null,
  "repositoryInfo": {...} | null,
  "lastUpdated": "...",
  "myStats": { /* stats for requesting user */ },
  "permissions": "read-only"
}
```

Errors:

- 403 — authorizeTeamAccess restricts access if not allowed
- 404 — team not found
- 500 — server error

Side effects: none

Delete team (creator only)

Method: DELETE
Path: /api/teams/:name
Auth: required
Middleware: authorizeTeamAccess — must set req.team; additional checks performed

Authorization check: current code requires req.user.teamRole === "creator" || req.user.role === "admin" (note: code checks both; the snippet currently checks if (req.user.teamRole !== "creator" || req.user.role !== "admin") which effectively denies deletion unless both true — **recommend**: change to && vs || depending on desired policy. See Notes.)

Success (200):

```json
{ "message": "Team deleted successfully" }
```

Errors:

- 403 — not creator/admin
- 500 — failed to delete team

Side effects:

- Deletes Team doc and RepoMetrics for teamId.

.

List all teams (admin-only)

Method: GET
Path: /api/teams
Auth: required; role === "admin"

Success (200):

```json
{ "teams": [ /* teams with populated creator & members */ ] }
```

Errors:

- 403 — admin required
- 500 — server error

AI Review / AI-generated analysis

Method: GET
Path: /api/ai-review
Auth: required

Query params:

- teamId (required)

Success (200):

```
{
  "aiFeedback": [ /* insights array from metricsEntry.aiAnalysis.insights */ ],
  "analysisMetadata": {
    /* metadata fields from metricsEntry.aiAnalysis.metadata */,
    "lastUpdated": "metricsEntry.aiAnalysis.lastAnalyzed"
  },
  "status": "completed"
}
```

Alternate success (202): if metricsEntry.analysisStatus !== 'completed':

{ "status": "in-progress", "message": "Analysis in progress. Please check back later." }

Errors:

- 400 — missing teamId
- 404 — metrics not found
- 429 — rate limit (e.g., GitHub API limit) — returned when error message contains "rate limit"
- 404 — repository not found (when error contains 404)
- 500 — other internal errors

Side effects: none (reads RepoMetrics).

Notes: error handling returns helpful suggestion strings.

Global middleware and error handlers

- **Logging middleware** logs each request: console.log("Incoming request:", req.method, req.url).
- **Error-handling middleware:** final app.use((err, req, res, next) => {...}) sends 500 with "Something went wrong!".
- **404 handler:** returns { message: "Route not found" }.

Background job

Job: GitHub username sync
Tool: node-cron
Schedule: daily at 01:00 (server local time) — cron "0 1 * * *"
Action: refreshGithubUsernames() — updates cached GitHub usernames in DB.

Notes: ensure server timezone expectation matches deployment (cron uses server local timezone).

Example request/responses

Create team example
Request:

```
POST /api/teams
Cookie: token=...
Content-Type: application/json

{
  "name": "TeamA",
  "password": "secret123",
  "repoUrl": "https://github.com/org/repo"
}
```

Success (201):

{
  "message": "Team created successfully",
  "team": { "id": "...", "name": "TeamA", "repoUrl":
"https://github.com/org/repo" },

"repositoryInfo": { "url": "https://github.com/org/repo", "description": "...", ... }
}

Join team example
Request:

```
POST /api/teams/join
Cookie: token=...
Content-Type: application/json

{ "name": "TeamA", "password": "secret123" }
```

Success:

```
{ "message": "Joined team", "teamId": "..." }
```