EAP Project - Repository Structure Guide

Version: 1.0
Date: 2026-01-06
Status: Approved
Audience: Staff + Interns

Overview

The EAP project uses a **multi-repository structure** with four separate repositories:

1. **eap-architecture** - Platform decisions, API specs, system diagrams
2. **eap-backend** - Python/FastAPI backend application
3. **eap-frontend** - React frontend application
4. **eap-qa** - Test automation and QA documentation

This document explains the structure, rationale, and conventions for working with multiple repositories.

Repository Details

1. eap-architecture

Purpose: Cross-cutting documentation and platform-level decisions

GitHub URL: `https://github.com/[org]/eap-architecture`

Structure:

```
eap-architecture/
├── decisions/
│   ├── ADR-001-repository-strategy.md
│   ├── ADR-002-tech-stack.md
│   ├── ADR-003-api-authentication.md
│   ├── template.md
│   └── README.md (index of all ADRs)
├── diagrams/
│   ├── context.drawio
│   ├── container.drawio
│   ├── component-backend.drawio
│   └── README.md
├── api-specs/
│   ├── openapi.yaml
│   └── README.md
├── deployment/
│   ├── docker-compose.yml
│   ├── deployment-guide.md
│   └── README.md
├── .github/
```

```
│   └── workflows/
│       └── validate-openapi.yml
└── README.md
```

What belongs here:

- Platform decisions (tech stack, infrastructure, tooling)
- API specifications (OpenAPI/Swagger)
- Architecture diagrams (C4 model)
- Deployment documentation
- Cross-repository coordination documentation

Who owns it:

- Product Owner (primary)
- DevOps lead (deployment docs)
- Whole team (contributes to ADRs and diagrams)

---

2. eap-backend

Purpose: Backend API application

GitHub URL: `https://github.com/[org]/eap-backend`

Structure:

```
eap-backend/
├── app/
│   ├── api/
│   │   ├── routes/
│   │   └── dependencies.py
│   ├── models/
│   ├── services/
│   ├── core/
│   │   ├── config.py
│   │   └── security.py
│   └── main.py
├── tests/
│   ├── unit/
│   ├── integration/
│   └── conftest.py
├── alembic/
│   ├── versions/
│   └── env.py
├── docs/
│   ├── decisions/
│   │   ├── ADR-001-database-schema.md
│   │   ├── ADR-002-orm-patterns.md
│   │   ├── template.md
│   │   └── README.md
```

```
|   └── api/
|       └── README.md
├── .github/
|   └── workflows/
|       ├── ci.yml
|       └── deploy.yml
├── Dockerfile
├── requirements.txt
├── pyproject.toml
├── alembic.ini
└── README.md
```

What belongs here:

- Backend application code
- Backend-specific tests
- Database migrations
- Backend-specific ADRs (schema design, ORM patterns, etc.)
- Backend API documentation

Who owns it:

- Backend developers
- DevOps lead (deployment configuration)

---

3. eap-frontend

Purpose: React frontend application

GitHub URL: `https://github.com/[org]/eap-frontend`

Structure:

```
eap-frontend/
├── src/
|   ├── components/
|   |   ├── common/
|   |   └── features/
|   ├── pages/
|   ├── services/
|   |   ├── api/
|   |   └── auth/
|   ├── store/
|   ├── hooks/
|   ├── utils/
|   ├── App.tsx
|   └── main.tsx
├── tests/
|   ├── unit/
|   └── integration/
├── public/
├── docs/
│   ├── decisions/
│   │   ├── ADR-001-state-management.md
│   │   ├── ADR-002-component-patterns.md
│   │   ├── template.md
│   │   └── README.md
│   └── components/
│       └── README.md
├── .github/
│   └── workflows/
│       ├── ci.yml
│       └── deploy.yml
├── Dockerfile
├── package.json
├── tsconfig.json
├── vite.config.ts
└── README.md
```

What belongs here:

- Frontend application code
- Frontend-specific tests
- Frontend-specific ADRs (state management, component patterns, etc.)
- Component documentation

Who owns it:

- Frontend developers
- DevOps lead (deployment configuration)

---

4. eap-qa

Purpose: Test automation and QA documentation

GitHub URL: `https://github.com/[org]/eap-qa`

Structure:

```
eap-qa/
├── e2e-tests/
│   ├── auth/
│   ├── requests/
│   ├── workflows/
│   └── conftest.py
├── performance/
│   ├── load-tests/
│   └── stress-tests/
├── test-plans/
│   ├── sprint-01-test-plan.md
│   └── regression-test-plan.md
├── test-data/
```

```
│   ├── fixtures/
│   └── mocks/
├── docs/
│   ├── decisions/
│   │   ├── ADR-001-e2e-framework.md
│   │   ├── ADR-002-test-data-strategy.md
│   │   ├── template.md
│   │   └── README.md
│   └── test-strategy.md
├── .github/
│   └── workflows/
│       ├── e2e.yml
│       └── performance.yml
├── requirements.txt
└── README.md
```

What belongs here:

- End-to-end tests
- Performance tests
- Test plans and documentation
- QA-specific ADRs (testing approach, tooling, etc.)
- Test data and fixtures

Who owns it:

- QA/Test automation lead
- Whole team contributes to test coverage

Why Multi-Repository?

This decision is documented in **eap-architecture ADR-001: Repository Strategy**.

Key reasons:

1. **Team experience** - Team is familiar with single-repo workflows
2. **DevOps simplicity** - Separate CI/CD pipelines per component
3. **Clear ownership** - Each repo has a clear primary owner
4. **Deployment independence** - Backend can deploy without triggering frontend build

Trade-offs accepted:

- Feature coordination requires discipline (Jira tickets link multiple PRs)
- API contract synchronization needs explicit process (OpenAPI in eap-architecture)
- ADRs need clear categorization rules

Working Across Repositories

Git Workflow

All repositories use Git Flow:

- `main` - Production-ready code
- `dev` - Integration branch
- `feature/EAP-XXX-description` - Feature branches

Branch protection:

- `main`: Requires 2 PR reviews, all CI checks must pass
- `dev`: Requires 1 PR review, all CI checks must pass

Feature Development Workflow

User Story: EAP-123 "User can submit access request"

1. **Sprint Planning:**
    - Identify which repos are affected: backend + frontend + qa
    - Create feature branches in each repo:
        - `eap-backend: feature/EAP-123-access-request-endpoint`
        - `eap-frontend: feature/EAP-123-access-request-form`
        - `eap-qa: feature/EAP-123-access-request-tests`
2. **Development:**
    - Backend team implements API endpoint
    - Frontend team implements UI (may need to wait for backend or use mocks)
    - QA prepares test cases
3. **Pull Requests:**
    - Each repo gets its own PR
    - All PRs reference Jira ticket: `EAP-123`
    - PRs may reference each other for context
4. **Review & Merge:**
    - PRs reviewed independently
    - Backend merges first (API available)
    - Frontend merges next (consumes API)
    - QA merges last (tests against integrated system)
5. **Jira Integration:**
    - Jira ticket shows all related PRs
    - Status: In Progress → In Review → Done (when all PRs merged)

Commit Message Convention

Format: `type(scope): description (EAP-XXX)`

Examples:

```
# In eap-backend
git commit -m "feat(api): add access request endpoint (EAP-123)"

# In eap-frontend
git commit -m "feat(requests): add request submission form (EAP-123)"

# In eap-qa
git commit -m "test(e2e): add access request flow tests (EAP-123)"

# In eap-architecture
git commit -m "docs(adr): add ADR-005 access request workflow (EAP-123)"
```

Types: feat, fix, docs, test, refactor, chore, ci

API Contract Management

Problem: Backend and frontend must agree on API structure.

Solution: OpenAPI specification in `eap-architecture`

Workflow:

1. **API design discussion** → Update `eap-architecture/api-specs/openapi.yaml`
2. **Backend** implements according to spec
3. **Frontend** generates TypeScript types from spec
4. **QA** validates contracts using spec

Validation:

- Backend: OpenAPI validation in tests
- Frontend: TypeScript types generated from spec
- CI: Automated validation of OpenAPI spec

Cross-Repository ADR References

Always include repository name when referencing ADRs from other repos.

Examples:

In `eap-backend/docs/decisions/ADR-003-endpoint-structure.md`:

```
## Related ADRs

### In this repository (eap-backend):
- ADR-001: Database Schema Approach
- ADR-002: ORM Patterns

### In other repositories:
- [eap-architecture ADR-003: API Authentication](https://github.com/org/eap-
architecture/blob/main/decisions/ADR-003-api-authentication.md)
- [eap-architecture ADR-004: API Versioning](https://github.com/org/eap-
architecture/blob/main/decisions/ADR-004-api-versioning.md)
```

ADR Decision Matrix

Which repository should your ADR go in?

Decision Type	Example	Repository	Who Decides	Rationale
Platform	React version (v18 vs v19)	eap-architecture	**Staff** (after team input)	Affects entire frontend, learning goals, hard to change
Platform	Component library ecosystem	eap-architecture	**Staff** (after team input)	Community size impacts maintenance, learning
Platform	Tech stack choice	eap-architecture	**Staff** (after team input)	Affects all components
Platform	Authentication approach	eap-architecture	**Staff** (after team input)	Security-critical, affects all components
Platform	Deployment strategy	eap-architecture	**Staff** (after team input)	Affects all deployments
Application	Specific component from library	eap-frontend	**Team**	Within approved ecosystem
Application	State management implementation	eap-frontend	**Team**	Within chosen stack (e.g., Zustand vs Context)
Application	Database schema design	eap-backend	**Team**	Within constraints (with GDPR considerations)
Application	ORM usage patterns	eap-backend	**Team**	Backend-specific detail
Application	Component structure patterns	eap-frontend	**Team**	Frontend-specific detail
Application	E2E framework choice	eap-qa	**Team**	QA-specific decision (within constraints)

Decision Type	Example	Repository	Who Decides	Rationale
Application	Test data management	eap-qa	**Team**	QA process detail

Platform vs Application Decision Criteria

Platform Decision Criteria:

- Affects entire system or multiple components
- Hard to change later (high switching cost)
- Requires experience/judgment beyond team level
- **Impacts learning goals or market preparation**
- Team lacks context for long-term implications

Examples:

- React v18 vs v19 (version choice = platform)
- Component library with niche vs mainstream community (ecosystem = platform)
- Database type (PostgreSQL vs MySQL)
- Authentication method (JWT vs Session)

Application Decision Criteria:

- Within established platform constraints
- Team can evaluate trade-offs
- Relatively changeable (lower cost)
- Good learning opportunity
- Team can experiment safely

Examples:

- Which Button component from approved library to use
- How to structure Redux state (if Redux chosen)
- Specific database table design
- Component file organization

Repository Assignment Rule of Thumb

- **Affects multiple repos?** → `eap-architecture` (Platform Decision, staff reviews)
- **Specific to one component?** → That component's repo (Application Decision, team owns)
- **Grey area?** → Start in `eap-architecture`, staff will guide during review

---

Versioning and Releases

Semantic Versioning

All repositories use semantic versioning: `MAJOR.MINOR.PATCH`

Sprint releases:

- Sprint 1: v0.1.0
- Sprint 2: v0.2.0
- Sprint 3: v0.3.0
- Final release: v1.0.0

Synchronized Releases

At the end of each sprint, **all repositories are tagged with the same version**:

```
# End of Sprint 2
eap-architecture: git tag -a v0.2.0 -m "Sprint 2 release"
eap-backend:      git tag -a v0.2.0 -m "Sprint 2 release"
eap-frontend:     git tag -a v0.2.0 -m "Sprint 2 release"
eap-qa:           git tag -a v0.2.0 -m "Sprint 2 release"
```

Deployment: Matching versions are deployed together.

Version Compatibility

Document version compatibility in each repository's README:

```
## Version Compatibility

This version (v0.2.0) is compatible with:
- eap-backend: v0.2.0
- eap-frontend: v0.2.0
- API spec: v0.2.0 (eap-architecture)
```

Repository README Structure

Each repository should have a README.md with:

1. **Project description**
2. **Setup instructions**
3. **Development workflow**
4. **Testing instructions**
5. **Deployment process**
6. **Link to other repositories**
7. **Link to architecture documentation**

8. **Version compatibility matrix**

Template: See `eap-architecture/templates/repository-readme-template.md`

CI/CD Per Repository

Each repository has its own GitHub Actions workflows:

Backend CI (eap-backend/.github/workflows/ci.yml)

- Runs on: Pull requests to `dev` or `main`
- Steps: Lint → Test → Build → Deploy (if main)

Frontend CI (eap-frontend/.github/workflows/ci.yml)

- Runs on: Pull requests to `dev` or `main`
- Steps: Lint → Test → Build → Deploy (if main)

E2E Tests (eap-qa/.github/workflows/e2e.yml)

- Runs on: Schedule (every 4 hours) or manual trigger
- Steps: Run E2E tests against deployed environment

OpenAPI Validation (eap-architecture/.github/workflows/validate.yml)

- Runs on: Pull requests to `main`
- Steps: Validate OpenAPI spec syntax and completeness

Coordination Tools

Jira

Ticket structure:

- Epic: Major feature (e.g., "User Access Management")
- User Story: Cross-repo feature (e.g., "User can submit request")
- Sub-tasks (optional): Per-repo implementation

Linking:

- PRs automatically link to tickets via commit messages containing ticket ID
- Tickets show all related PRs from all repositories

Microsoft Teams

Two separate Teams are used:

1. EAP Project (Trainees + Staff)

- Primary communication for project work
- All trainees have access
- Staff participates actively

Channels:

- `#general` - Project updates, announcements, general questions
- `#backend` - Backend development discussions
- `#frontend` - Frontend development discussions
- `#qa` - Testing and quality assurance
- `#devops` - CI/CD, deployments, infrastructure
- `#architecture` - Architecture discussions and ADR reviews
- `#random` - Social and non-work chat

2. EAP Staff Only (Staff ONLY)

- Staff coordination and private discussions
- Trainees do NOT have access
- Completely separate from trainee team

Channels:

- `#general` - Staff coordination and planning
- `#assessments` - Assessment discussions (private)
- `#incidents` - Incident observations (private)
- `#coaching` - Coaching strategies (private)

Important: Staff-only discussions (assessments, incidents, coaching notes) happen ONLY in the Staff team, never in the trainee team.

Daily Standup

Format:

- What did you do? (mention repo if relevant)
- What will you do? (mention repo if relevant)
- Any blockers? (especially cross-repo dependencies)

Common Pitfalls and Solutions

Pitfall 1: API Mismatch

Problem: Backend implements API differently than frontend expects.

Solution:

- Always update OpenAPI spec first in `eap-architecture`
- Backend validates responses against spec in tests
- Frontend generates types from spec
- QA validates contracts

Pitfall 2: Circular Dependencies

Problem: Backend PR needs frontend PR needs backend PR.

Solution:

- Design APIs before implementation
- Use mocks/stubs during parallel development
- Backend implements endpoint with sample data
- Frontend develops against mocks
- Integration testing validates the connection

Pitfall 3: Forgotten ADRs

Problem: Decision made in Slack, not documented in ADR.

Solution:

- Sprint Retrospective: "Did we make any significant decisions this sprint?"
- Write retroactive ADRs for important decisions
- Make ADR writing part of Definition of Done

Pitfall 4: Version Confusion

Problem: Not sure which backend version works with which frontend version.

Solution:

- Synchronize version tags at end of each sprint
- Document version compatibility in README
- Use matching version numbers in deployment

Getting Started

First Week Checklist

For all team members:

- [ ] Clone all 4 repositories
- [ ] Verify access to all repositories
- [ ] Read each repository's README.md
- [ ] Understand Git Flow workflow
- [ ] Review ADR template in eap-architecture
- [ ] Join Microsoft Teams channels

For DevOps lead:

- [ ] Verify CI/CD pipelines work
- [ ] Test deployment to TransIP VPS
- [ ] Ensure branch protection rules are active
- [ ] Verify Jira-GitHub integration works

For Product Owner:

- [ ] Review eap-architecture structure
- [ ] Understand ADR categorization
- [ ] Prepare first architectural decisions for documentation
- [ ] Receive GDPR requirements from staff/DPO before Sprint Planning

For Staff (before Sprint 1):

- [ ] Consult with DPO/legal on GDPR requirements
- [ ] Define lawful basis for data processing
- [ ] Set retention policies for personal data
- [ ] Provide GDPR requirements to Product Owner
- [ ] Review proposed GDPR compliance approach (ADR)

Further Reading

- eap-architecture ADR-001: Repository Strategy
- Architecture Decision Record Template
- Git Flow Workflow
- Semantic Versioning
- OpenAPI Specification

Document Control

- Created: 2026-01-06
- Owner: Motopp Staff
- Review Frequency: After Sprint 1
- Status: Approved