

APIBridge: Unified API Creation for DIC Projects

Introduction

This proposal presents APIBridge, a transformative initiative to convert isolated Digital Innovation Center (DIC) projects into accessible, standardized APIs that enable widespread adoption and collaboration. Our 6-week development plan will establish a unified API platform serving 2-3 high-value DIC projects through modern microservices architecture, comprehensive documentation, and free-tier deployment infrastructure. The project addresses the critical gap between innovative research outputs and practical accessibility, potentially impacting hundreds of external users while establishing DIC as a leader in open innovation.

1. Problem Statement

Digital Innovation Centers develop numerous high-value projects annually, yet these solutions remain significantly underutilized due to accessibility barriers and lack of standardized interfaces. Current projects exist as isolated systems without unified access methods, limiting their potential for broader academic and industry adoption.

2. Research Gap

While individual DIC projects demonstrate technical excellence, there exists no comprehensive platform for exposing these innovations as consumable services. This isolation prevents synergistic effects between projects and limits opportunities for external collaboration and validation.

3. Proposed Solution

APIBridge implements a unified API platform using FastAPI, Docker containerization, and comprehensive OpenAPI documentation to transform selected DIC projects into publicly accessible microservices. Our approach leverages proven open-source technologies to ensure cost-effectiveness while maintaining enterprise-grade capabilities.

4. Project Significance

This initiative will maximize the societal impact of research investments, establish sustainable infrastructure for future innovations, and position DIC as a leader in academic-industry collaboration through accessible, well-documented APIs.

Details of Students/Team

No of team members: 2

| | |
|----------------------------|---|
| Name of the Student | RAGHAV GUPTA |
| Course Details | BE IT 2 ND YEAR Sem 4 |
| Contact No. | 7973457736 |
| Email | raghavrgupta56@gmail.com |
| GitHub Username | Raghav-56 |
| LinkedIn | https://www.linkedin.com/in/raghav-gupta-035b4a292/ |
| Resume | https://raghav-56.github.io/Resume/ |

| | |
|----------------------------|---|
| Name of the Student | Priyanshu Anand |
| Course Details | BE IT 2 ND YEAR Sem 4 |
| Contact No. | 6205146659 |
| Email | priyanshu82711@gmail.com |
| GitHub Username | priyans11 |
| LinkedIn | https://www.linkedin.com/in/nice-x1 |
| Resume | https://priyans11.github.io/P-resume/ |

Project Objectives and background

1. Primary Objectives

- Accessibility Enhancement: Convert 2-3 high-value DIC projects into public-facing REST APIs with comprehensive documentation
- Standardization Framework: Establish robust API design patterns using OpenAPI/Swagger specifications
- Scalable Infrastructure: Implement containerized deployment architecture using free-tier cloud platforms
- Collaboration Catalyst: Create foundation supporting interdisciplinary development and external partnerships

2. Success Metrics

- 100% of selected APIs deployed with 99.9% uptime
- Interactive documentation with <2-second response times
- Complete test coverage (>90%) for all endpoints
- Zero-cost hosting solution with scalability to 10,000+ requests/month

3. Expected Outcomes

- Functional API platform serving 2-3 DIC projects
- Comprehensive documentation enabling external adoption
- Replicable template for future project integrations
- Enhanced DIC visibility in academic and industry communities

Methodology & Technical Approach

1. Development Methodology

Our approach combines agile development principles with academic research best practices, ensuring rapid iteration while maintaining rigorous quality standards. We employ API-first design, creating contracts before implementation to enable parallel development of documentation and testing.

2. Technical Architecture

The platform follows proven microservices patterns incorporating:

- **API Gateway:** Central routing, authentication, and rate limiting
- **Microservices Layer:** Independent APIs for each DIC project
- **Data Layer:** PostgreSQL with Redis caching for performance
- **Documentation Layer:** Automated Swagger UI generation

3. Implementation Strategy

1. **Week 1-2 (Foundation):** Project selection, API specification, environment setup
2. **Week 3-4 (Development):** Microservices implementation, testing, integration
3. **Week 5 (Deployment):** Containerization, CI/CD pipeline, staging deployment
4. **Week 6 (Launch):** Performance testing, security validation, production deployment

4. Quality Assurance

Comprehensive testing framework including functional validation, performance benchmarking, and security assessment using industry-standard tools and practices.

Technical Stack and Implementation

1. Technology Stack

FastAPI Framework: High-performance async Python framework with automatic OpenAPI generation, native type hint support for robust validation, and seamless integration with scientific computing libraries—superior to Flask/Django for API-first development.

Docker Containerization: Lightweight containerization ensuring consistent deployment environments, efficient resource utilization, and seamless integration with free-tier hosting platforms for cost-effective scaling.

PostgreSQL Database: Open-source relational database with excellent Python integration, JSON support for flexible research data models, and proven compatibility across major hosting platforms.

GitHub Actions CI/CD: Automated testing and deployment pipelines with free-tier support for public repositories, enabling continuous integration and seamless deployment workflows.

2. Architecture Components

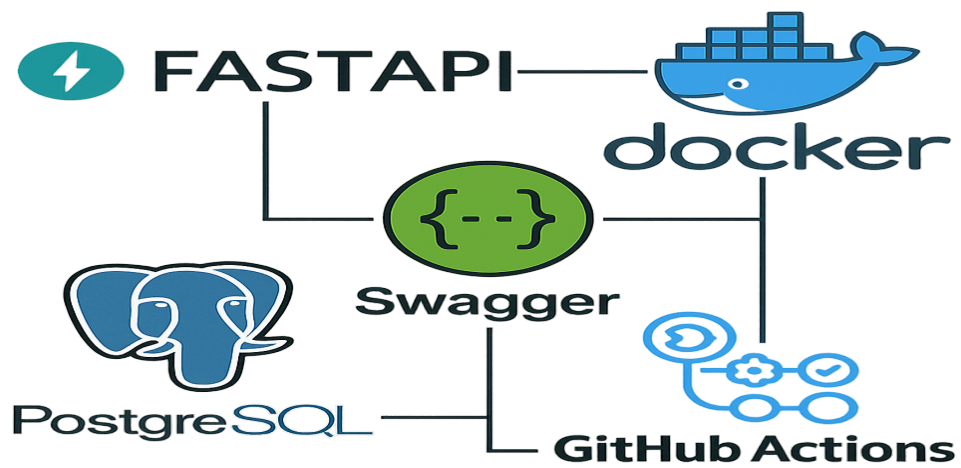
Presentation Layer: Interactive Swagger/OpenAPI 3.0 documentation with automated generation and built-in testing capabilities

API Gateway: JWT/OAuth2 authentication, intelligent rate limiting, and optimized request routing

Microservices: Independent APIs for each DIC project with comprehensive health monitoring and service isolation

Data Layer: PostgreSQL primary storage with Redis caching integration and scalable file storage solutions

Infrastructure: Containerized deployment with automated CI/CD pipelines, real-time monitoring, and free-tier optimization



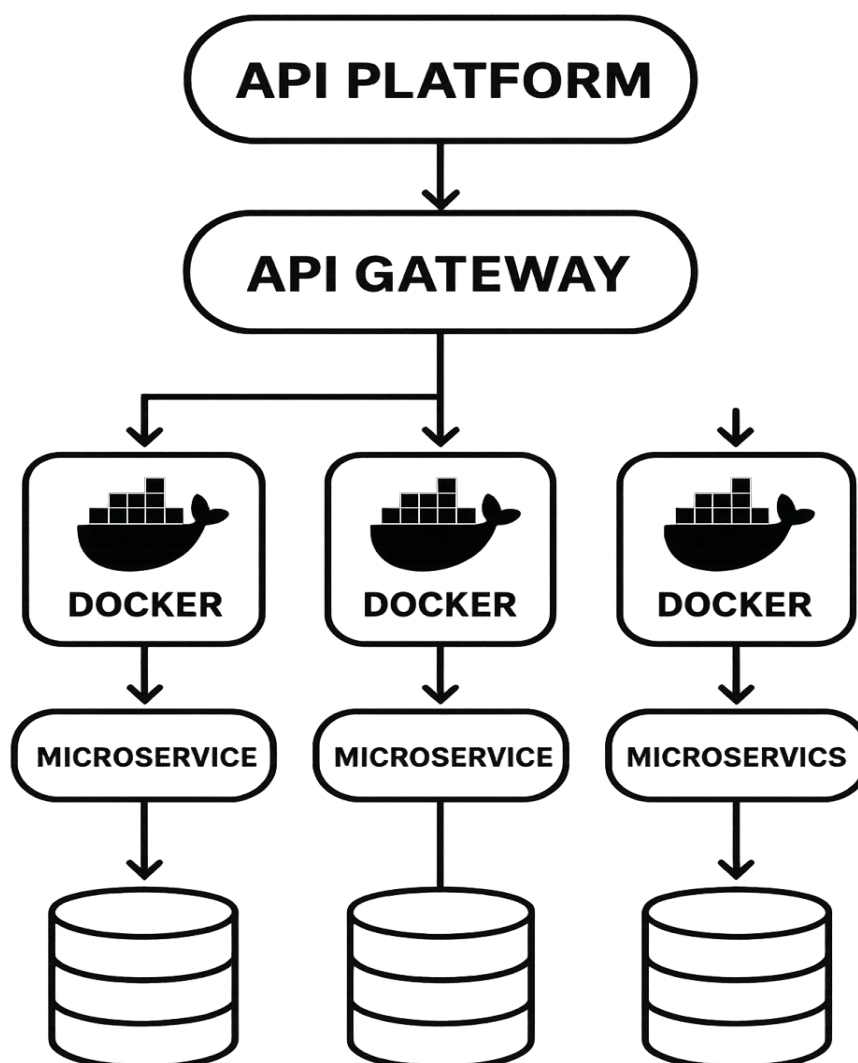
3. Deployment Strategy

Multi-platform deployment leveraging Render as primary host, with Vercel and Railway as strategic alternatives, ensuring 99.9% uptime through intelligent redundancy and cost-effective resource management across free-tier services.

API platform architecture

Our architecture follows proven patterns for scalable API platforms, incorporating essential components for academic and research environments. The microservices architecture enables independent development and deployment of each DIC project while maintaining centralized management through an API gateway that handles routing, authentication, and traffic distribution.

The containerized deployment strategy ensures consistent environments across development, testing, and production phases, while the documentation layer provides automated Swagger UI generation for interactive API exploration.



This design supports both current project requirements and future expansion to accommodate AI models, data analytics tools, and collaborative research platforms.

REST API

Project

GET`/api/projects`

Get all projects

POST`/api/projects`

Create a project

Submission

GET`/api/submissions/{id}`

Get a submission by ID

PUT`/api/submissions/{id}`

Update a submission by ID

DELETE`/api/submissions/{id}`

Delete a submission by ID

User

GET`/api/users/me`

Get current user

Token

POST`/api/tokens`

Create a token

Swagger documentation interface

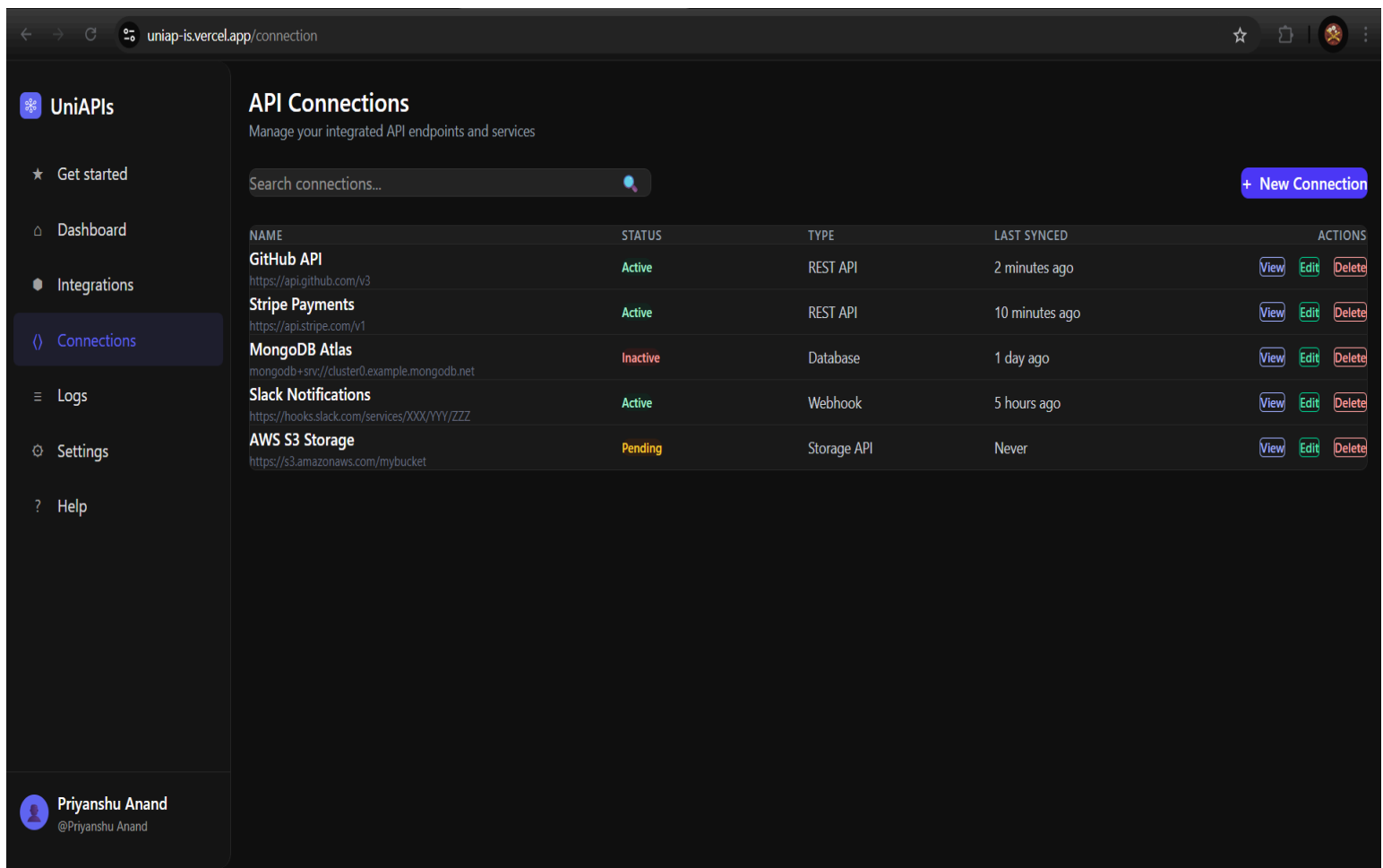
The interactive documentation interface provides comprehensive API exploration capabilities, including real-time endpoint testing with customizable parameters, detailed request/response schemas with validation rules, and code generation examples in multiple programming languages. This approach ensures that both technical and non-technical users can effectively discover and integrate DIC services into their research workflows.

Implementation: (poc)

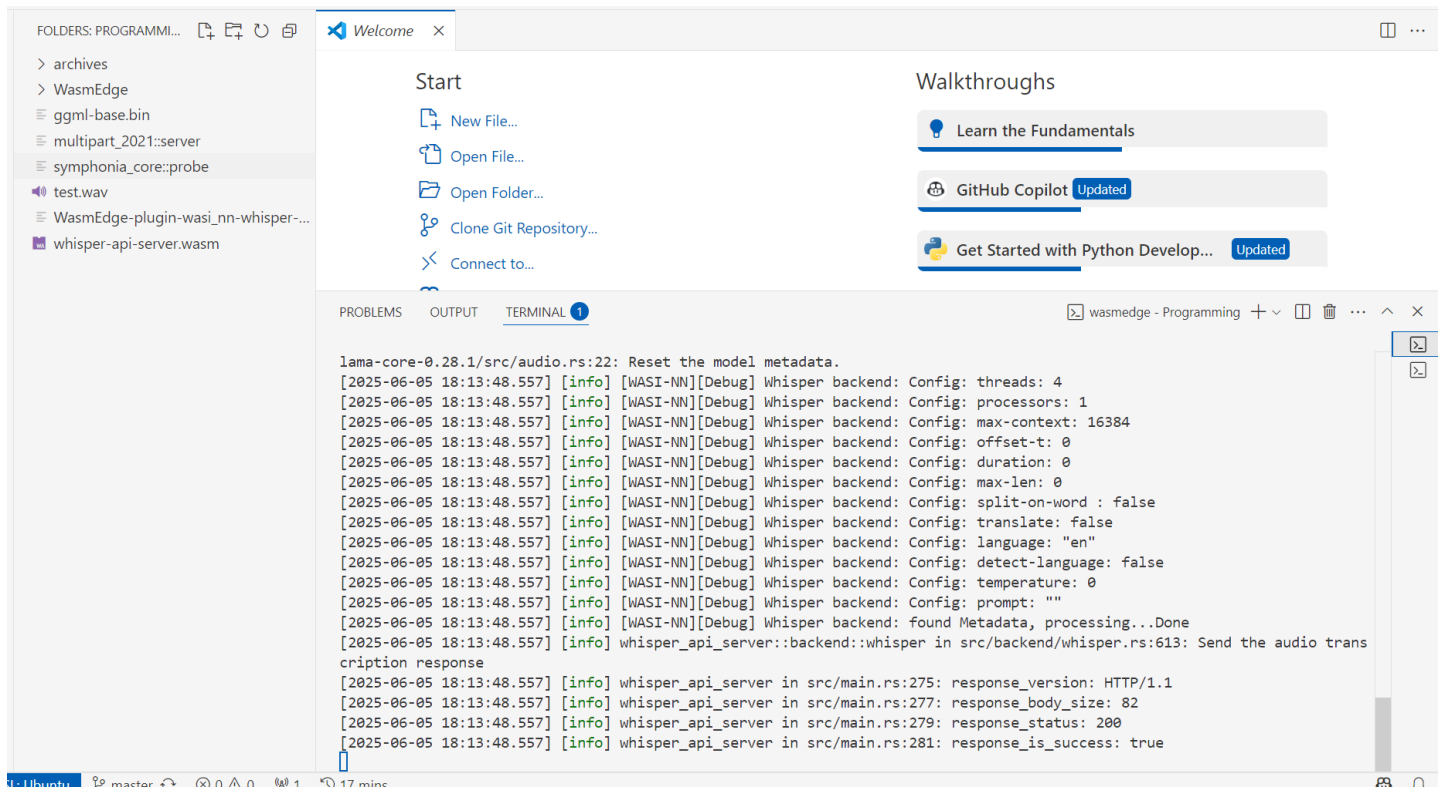
UI:

<http://uniap-is.vercel.app>

<https://github.com/priyans11/backend.git>



Bankend:



```
/home/raghav/.zshrc:90: command not found: EDITOR
raghav@MSI-Raghav ~/Programming$ sudo ufw allow 8080
[sudo] password for raghav:
Skipping adding existing rule
Skipping adding existing rule (v6)
raghav@MSI-Raghav ~/Programming$ curl --location 'http://localhost:8080/v1/audio/transcriptions' \
--header 'Content-Type: multipart/form-data' \
--header 'Authorization: Bearer itiswhatitis56' \
--form 'file=@"test.wav"'
{"text":"[00:00:00.000 --> 00:00:04.000] This is a test record for Whisper.CPP."}%
raghav@MSI-Raghav ~/Programming$
```

<https://github.com/Raghav-56/audio-transcription-backend>

Project Scope & Deliverables

Phase 1: Foundation Development (Weeks 1-2)

Project Selection & Analysis

- Identify and prioritize 2-3 high-impact DIC projects based on:
 - Technical feasibility assessment
 - Research value and potential external adoption
 - Resource requirements and complexity analysis
 - Stakeholder interviews and approval documentation
- Target candidates: handwriting generator, chatbot backend, data analysis tools
- Complete project evaluation matrix with scoring criteria

Technical Infrastructure Setup

- Development environment standardization using Docker Compose
- FastAPI project templates with pre-configured dependencies
- Authentication framework implementation (OAuth 2.0 + JWT)
- Database selection and setup (PostgreSQL for structured data, MongoDB flexibility options)
- Git repository structure with branching strategy

API Specifications & Design

- Complete OpenAPI 3.0 specifications for each selected project
- RESTful API design patterns and endpoint mapping
- Request/response schema definitions with validation rules
- Error handling standards and HTTP status code conventions
- Rate limiting and throttling specifications

Phase 2: Core Development (Weeks 3-4)

API Implementation

- Functional REST APIs using FastAPI framework
- Pydantic models for automatic data validation
- Async/await implementation for I/O-bound operations
- Database integration with connection pooling
- Middleware implementation for logging, CORS, and security headers

Security Implementation

- JWT-based authentication with refresh token support
- Role-based access control (RBAC) with project-specific scopes

- API key management system for external integrations
- Input sanitization and SQL injection prevention
- HTTPS enforcement and security header configuration

Testing & Quality Assurance

- Unit test suite achieving >90% code coverage
- Integration tests for API endpoints
- Load testing specifications and benchmarks
- Automated security scanning integration
- Code quality tools (linting, formatting, type checking)

Phase 3: Deployment Infrastructure (Week 5)

Containerization & Orchestration

- Multi-stage Docker builds for optimized production images
- Docker Compose configurations for local development
- Environment-specific configuration management
- Health checks and container monitoring setup
- Volume management for persistent data storage

CI/CD Pipeline Development

- GitHub Actions workflows for automated testing
- Automated security scanning and dependency checks
- Multi-environment deployment (development, staging, production)
- Automated documentation generation and deployment
- Rollback mechanisms and deployment monitoring

Hosting & Infrastructure

- Free-tier cloud deployment on Render/Heroku/Vercel
- Database hosting setup (PostgreSQL on Railway/Supabase free tier)
- CDN configuration for static assets
- Domain configuration and SSL certificate management
- Backup and disaster recovery procedures

Phase 4: Analytics, Testing & Launch (Week 6)

Performance Optimization

- API response time optimization and caching strategies
- Database query optimization and indexing
- Load balancing configuration for high availability
- Performance monitoring dashboard setup
- Scalability testing and bottleneck identification

Analytics & Monitoring Implementation

- Usage analytics dashboard (open-source tools like Grafana)
- API usage metrics (requests/minute, error rates, response times)
- User adoption tracking and engagement metrics
- System health monitoring with alerting
- Custom analytics for research impact measurement

Launch Preparation

- Comprehensive API documentation with interactive testing
- User onboarding guides and tutorials
- Production deployment with monitoring
- Stakeholder training sessions
- Community feedback collection mechanisms

Future Extensions

The platform's modular design ensures easy scalability and integration for evolving research needs.

1.AI/ML Model Integration

Supports seamless AI/ML deployment via `/ai-model` endpoints:

- Plug in custom models tailored for DIC research
- Deploy algorithms in production with consistent APIs
- Enable versioning, A/B testing, and flexible model updates

2.Advanced Analytics & Monitoring

Built-in observability using open-source tools:

- Real-time API usage and performance tracking
- Logging, alerting, and resource monitoring
- Custom dashboards and historical trend analysis

3.External API Integration

Supports third-party services with standardized, secure interfaces:

- Auth, rate limiting, and consistent data formats
- Webhook/event support and multi-endpoint config
- Data transformation layers for seamless integration

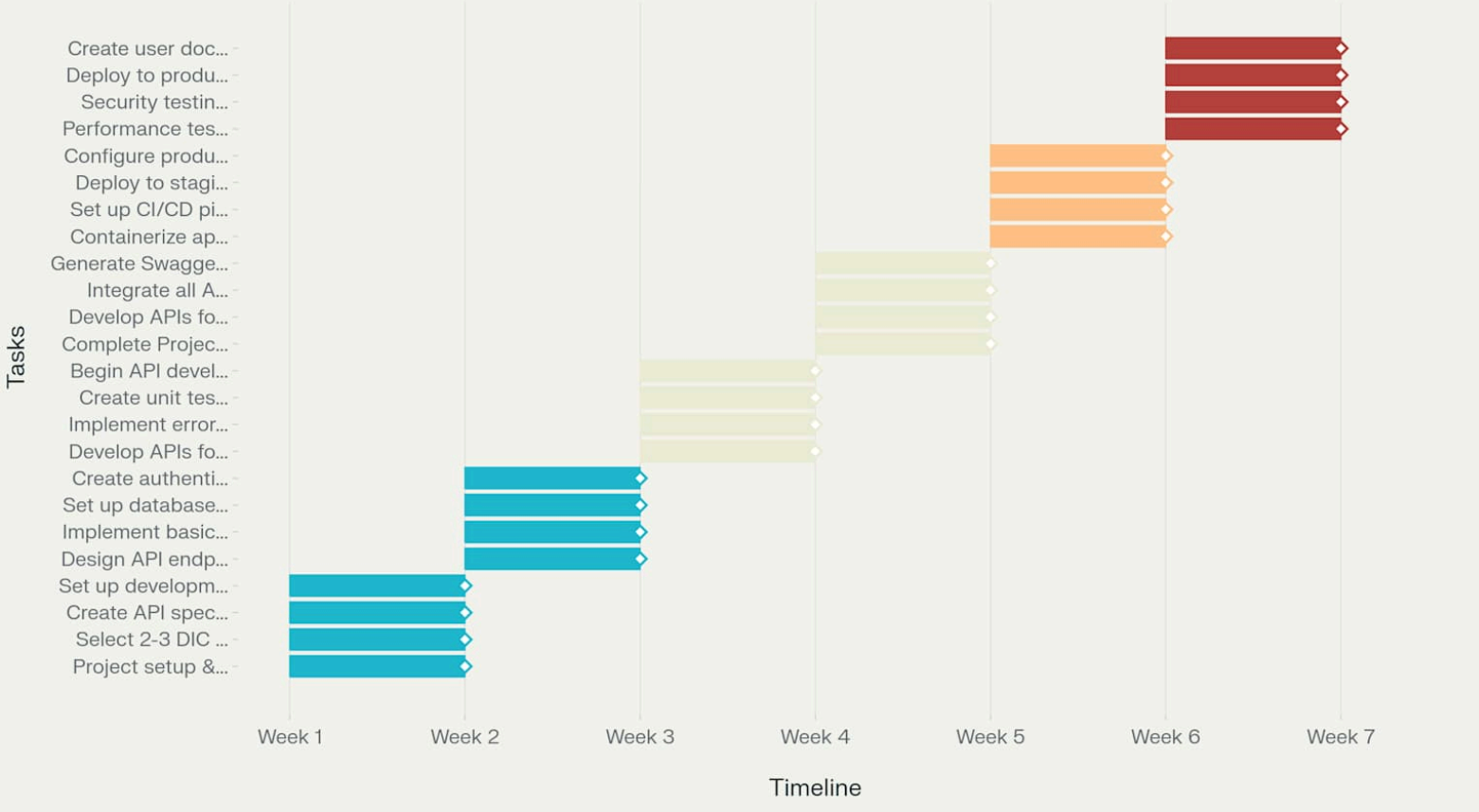
The architecture prioritizes adaptability, reliability, and cost-efficiency—leveraging open-source, self-hostable components.

Project Timeline and Work Plan

| Week | Phase | Key Deliverables | Success Metrics |
|------|---|---|---|
| 1 | Plan API endpoints | Project selection, API specifications, development environment | 3 projects selected, specs approved, environment operational |
| 2-3 | Core Development of APIs | REST API implementation, testing framework, basic documentation | APIs functional, 90%+ test coverage, preliminary docs complete |
| 4 | Documentation & Testing | Swagger UI, comprehensive testing, user feedback integration | Interactive docs live, performance benchmarks met |
| 5 | Containerisation and Deployment | Containerization, CI/CD pipeline, production deployment | Services deployed, automation functional, monitoring active |
| 6 | Validation Testing and final Improvements | Performance optimization, analytics implementation, community rollout | Performance targets achieved, analytics operational, user onboarding complete |

Project Timeline

Foundation Development Deployment Testing & Launch



Outcomes and Unique Value Proposition

Immediate Impact

- Research Acceleration: DIC projects become accessible to external researchers, students, and industry partners, expanding impact and citation opportunities
- Collaboration Enhancement: Standardized APIs enable seamless project integration, fostering interdisciplinary collaboration
- Technical Excellence: Demonstrates DIC's commitment to industry best practices in software architecture and deployment

Strategic Long-Term Benefits

- Innovation Ecosystem: Platform infrastructure supports student entrepreneurship, research commercialization, and technology transfer
- Industry Partnerships: Standardized APIs reduce integration barriers, enabling effective university-industry collaboration and new funding opportunities
- Educational Value: Students gain enterprise-grade experience in API development, containerization, and deployment practices
- Scalable Growth: Modular architecture supports unlimited expansion while maintaining quality and user experience

Competitive Advantages

- Open Innovation Leadership: Positions DIC as a leader in collaborative research, attracting academic and industry partners
- Cost-Effective Sustainability: Free-tier and open-source technologies ensure long-term viability without licensing costs
- Reproducible Model: Provides a template for other institutions, establishing DIC as a thought leader in academic technology

Strategic Vision

This project transforms academic innovation by converting isolated projects into accessible, standardized services through API-first design methodology, amplifying the value of research investments.

The agile development approach combines industry best practices with academic rigor, ensuring rapid iteration and quality standards. The microservices architecture enables independent service operation with granular scaling and fault isolation, translating research excellence into practical applications.

The platform establishes DIC as a pioneer in open research infrastructure, demonstrating how institutions can leverage containerized deployment and automated documentation to enhance collaboration. This foundation provides machine-readable specifications and interactive capabilities that attract academic peers, industry partners, and funding organizations.

Implementation Readiness: The comprehensive technical plan featuring API contract-first design and proven containerization strategy positions this project for immediate implementation with minimal risk and maximum impact potential.

Attachments


Research tables:

- architecture_components.csv
- hosting_platform_comparison.csv
- project_timeline.csv
- technology_stack_comparison.csv
- risk_assessment_matrix.csv

 AISOC

<https://drive.google.com/drive/folders/1m0vKkQWcR4YB6q3gqQ06D9DB2ZCzvsJB>

<https://docs.google.com/document/d/1lo1lsKXt4l15qSgKa7k05m57D8QJVngeGcddWml4TCE/edit?tab=t.t7mfumw30tu6>

 Proposal_API Creation for DIC Projects