Project Design Documentation — InkConnect

1. Project Title & Version Control

Project Title:

InkConnect — Artist Request & Portfolio Backend API

Version Control:

- Version: DRAFT
- Date: 08/19/2025
- Change Log: N/A

2. Project Summary

InkConnect is a backend API platform designed to connect tattoo artists and visual artists with clients seeking custom artwork. It allows artists to showcase portfolios, receive commission requests, and manage ongoing projects, while clients can browse, request, and track projects. This project matters because it bridges the gap between artists and clients, streamlining the commission process through a secure, efficient, and scalable backend system.

3. Problem Statement / Use Case

Many tattoo artists and digital artists struggle to find reliable ways to showcase their work and handle custom commissions online. Clients often face challenges finding trustworthy artists, comparing styles, and managing requests. InkConnect solves this by providing a centralized, secure platform for artist-client collaboration.

Use Cases:

- A tattoo artist uploads a portfolio and receives commission requests through the API.
- A client searches for an artist by style and submits a custom request.
- Both parties can track progress and communicate within a managed request system.

4. Goals and Objectives

1. Develop a secure, scalable RESTful API with authentication and role-based access.
2. Implement portfolio management (upload, update, retrieve artwork).
3. Create a commission request and tracking system for clients and artists.

5. Key Features / Functions

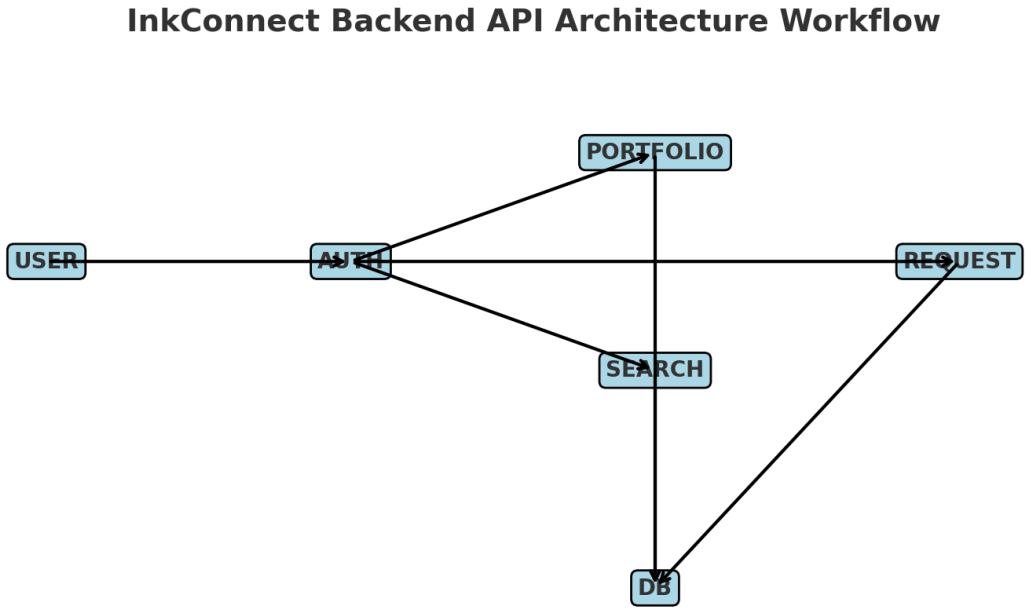
- Authentication & Authorization: Secure login/signup, JWT-based token auth, role-based permissions.
- Artist Portfolio Management: Upload, update, and retrieve artwork portfolios.
- Commission Requests: Clients can submit requests; artists can accept, decline, or update status.
- Search Functionality: Clients can search for artists by style, tags, or availability.
- Project Tracking: Both artists and clients can view the status of ongoing requests.

6. Tech Stack and Tools

- Languages: Python
- Frameworks: Flask / FastAPI (for backend API development)
- Database: PostgreSQL (structured data, user/portfolio storage)

- Authentication: JWT (JSON Web Tokens)
- Cloud Hosting: AWS / Heroku (for deployment)
- Version Control: Git + GitHub
- Testing: Pytest, Postman (API testing)
- Documentation: Swagger / OpenAPI

7. Architecture / Workflow Diagram



Workflow:

1. Client signs up/logs in → Receives JWT token.
2. Client browses artist portfolios → Uses API search endpoints.
3. Client submits request → Stored in database and linked to artist profile.

4. Artist accepts/declines → Status updated via API.
5. Both parties track project progress through request status updates.

Algorithm & Flowchart (Simplified):

[User Login/Register]
↓
[Authenticate via JWT]
↓
[Access Portfolio / Search Artists]
↓
[Submit Commission Request]
↓
[Artist Reviews Request]
→ Accept → [Status: In Progress]
→ Decline → [Status: Closed]
↓
[Update / Complete Project Status]

8. Timeline / Weekly Milestones

Week	Outcome
Week 1	Project setup: GitHub repo, Flask/FastAPI boilerplate, DB schema design
Week 2	Implement authentication (JWT) + role-based access
Week 3	Build artist portfolio management endpoints
Week 4	Add commission request submission & tracking endpoints

Week 5	Develop search functionality (filter by tags, style)
Week 6	API documentation with Swagger + testing (Postman, Pytest)
Week 7	Security hardening (input validation, DB protection)
Week 8	Deployment setup (Heroku/AWS) + CI/CD integration
Week 9	User acceptance testing, bug fixes
Week 10	Finalize API, prepare presentation & documentation
Week 11	Buffer for refinements & extra features
Week 12	Final delivery, evaluation, and showcase

9. Risks and Risk Mitigation

- Risk: Database inconsistencies (lost requests or broken references).

  Mitigation: Use relational schema with proper constraints & validation.

- Risk: Security vulnerabilities (JWT leaks, SQL injection).

  Mitigation: Implement input validation, token expiration, HTTPS-only connections.

- Risk: Scope creep — adding too many features.

  Mitigation: Stick to core features first, extras later.

10. Evaluation Criteria

Success will be measured by:

1. Functionality: API endpoints work correctly and return expected results.
2. Security: JWT authentication and role-based permissions enforced.
3. Reliability: Portfolio and commission workflows function end-to-end without errors.
4. Documentation: Swagger/OpenAPI documentation is complete and usable.
5. Deployment: API runs on cloud hosting and is accessible via endpoints.

11. Future Considerations

- Add payment processing integration (Stripe, PayPal) for commission handling.
- Implement chat/messaging system for direct artist-client communication.
- Add image processing features (auto-compression, watermarking).
- Develop a frontend client (mobile/web app) that consumes the API.
- Enable multi-language support for international users.