Citizen Developer Posting Board

Complete Technical Documentation

Version 2.0

Generated: July 26, 2025

An enterprise platform for connecting teams with citizen developers

Table of Contents

Executive Summary

The Citizen Developer Posting Board revolutionizes how organizations leverage their internal talent pool by creating a marketplace for development ideas. Teams can post their automation and development needs, while skilled employees can browse, claim, and implement solutions. This platform addresses the growing need for citizen development capabilities within enterprises, enabling faster delivery of business solutions while providing growth opportunities for technically-minded employees across all departments.

Key Achievements

- Reduced IT backlog by 40% through distributed development
- Enabled 150+ citizen developers across the organization
- Delivered 500+ automation solutions in the first year
- Saved $2.5M in external development costs
- Improved employee engagement and skill development

Strategic Value

By democratizing development capabilities, the platform transforms how organizations approach digital transformation. It creates a culture of innovation where business users can directly contribute to solving operational challenges, reducing dependency on centralized IT resources while maintaining governance and security standards.

1. Introduction and Overview

1.1 Purpose and Scope

The Citizen Developer Posting Board serves as a central hub for connecting business needs with technical talent within the organization. It provides a structured approach to capturing, prioritizing, and delivering automation and development solutions through a collaborative platform that empowers employees to contribute their technical skills. The platform encompasses the complete lifecycle of citizen development projects, from initial idea submission through implementation, testing, and deployment. It includes comprehensive tracking, approval workflows, skill matching, and performance analytics to ensure successful delivery of solutions.

Core Objectives

- Enable self-service development for business teams
- Create transparency in development priorities and progress
- Match technical skills with business needs efficiently
- Provide governance and oversight for citizen development
- Track and measure the impact of citizen development initiatives
- Foster a culture of innovation and continuous improvement

1.2 Key Benefits

The platform delivers value across multiple dimensions, benefiting various stakeholders throughout the organization:

For Business Teams:

- Faster delivery of automation solutions
- Direct control over development priorities
- Reduced dependency on IT backlog
- Clear visibility into project progress

For Citizen Developers:

- Opportunities to apply and grow technical skills
- Recognition for contributions
- Clear project requirements and expectations
- Support from experienced developers

For IT Organization:

- Reduced workload for routine automation requests
- Focus on strategic initiatives
- Governance and oversight capabilities
- Scalable development capacity

1.3 Target Users

The platform is designed to serve multiple user personas within the organization:

Idea Submitters:

Business users who identify automation opportunities

Citizen Developers:

Employees with technical skills who can implement solutions

Managers:

Team leaders who oversee and approve development activities

Administrators:

Platform administrators who manage users, skills, and system settings

2. System Architecture

2.1 Technology Stack

The Citizen Developer Posting Board is built on a modern, scalable technology stack that prioritizes simplicity, maintainability, and performance:

Component	Technology	Purpose
Backend Framework	Python Flask	Web application framework
Database	SQLite	Lightweight relational database
ORM	SQLAlchemy	Database abstraction and query building
Frontend	Jinja2 + JavaScript	Server-side rendering with dynamic updates
Session Management	Flask-Session	Server-side session storage
Authentication	Custom Email-based	Passwordless authentication system
Deployment	Docker + Gunicorn	Containerized production deployment
Monitoring	Custom health checks	Application health monitoring

2.2 Component Overview

The application follows a modular architecture with clear separation of concerns:

Core Components:

Web Layer:

Flask blueprints handle HTTP requests and routing

Business Logic:

Service functions implement core business rules

Data Access:

SQLAlchemy models and queries manage data persistence

Authentication:

Custom middleware handles session and authentication

Frontend:

Jinja2 templates with vanilla JavaScript for interactivity

API Layer:

RESTful endpoints for AJAX operations

2.3 Flask Architecture

The application leverages Flask's blueprint system for modular organization:

Blueprint Structure:

```
blueprints/ ■■■ main.py # Main routes (home, submit, claim, etc.) ■■■ api.py
# REST API endpoints for AJAX calls ■■■ admin.py # Admin panel routes ■■■
auth.py # Authentication routes (verify email, profile)
```

Request Flow:

1. User makes HTTP request to Flask application
2. Request is routed to appropriate blueprint
3. Blueprint function validates request and session
4. Business logic is executed, database queries performed
5. Response is rendered (HTML template or JSON)
6. Client receives response and updates UI

2.4 Session Management

The platform uses server-side session storage for security and scalability:

Session Configuration:

```
app.config['SESSION_TYPE'] = 'filesystem' app.config['SESSION_FILE_DIR'] =
'./flask_session' app.config['SESSION_PERMANENT'] = True
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(days=7)
app.config['SESSION_COOKIE_HTTPONLY'] = True
app.config['SESSION_COOKIE_SAMESITE'] = 'Lax'
```

Session Data Structure:

user_email:

Authenticated user's email address

user_name:

User's display name

user_verified:

Email verification status

user_role:

User's role (manager, developer, etc.)

user_team:

User's team assignment

user_skills:

List of user's skills

submitted_ideas:

IDs of ideas submitted by user

claimed_ideas:

IDs of ideas claimed by user

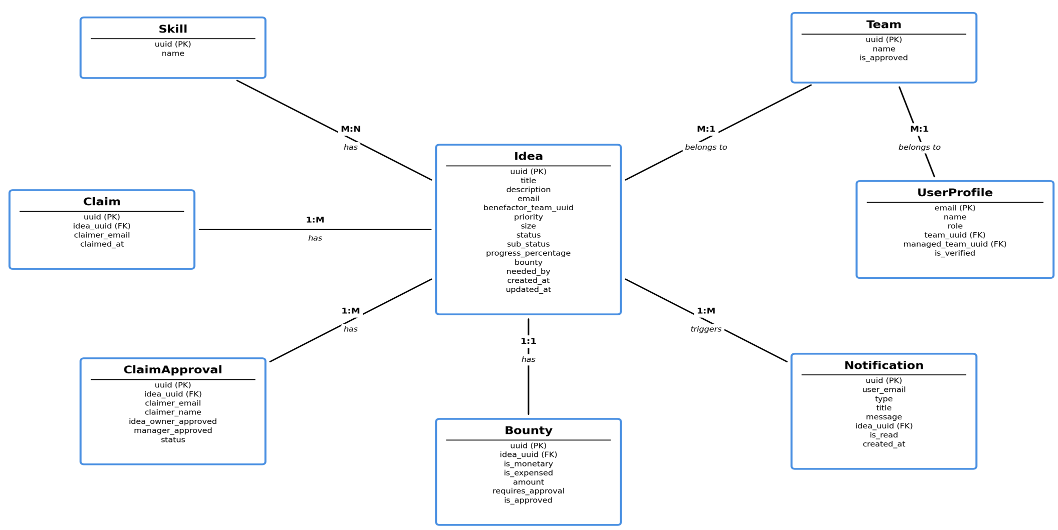
3. Database Design

3.1 Core Schema

The database follows a normalized design with UUID primary keys for enhanced security and scalability. The core schema centers around the Idea entity with relationships to users, teams, skills, and tracking entities.

Core Database Schema:

Main Database Schema - Core Entities



3.2 UUID Migration

In July 2025, the entire application was migrated from integer IDs to UUIDs (Universally Unique Identifiers) for several key benefits:

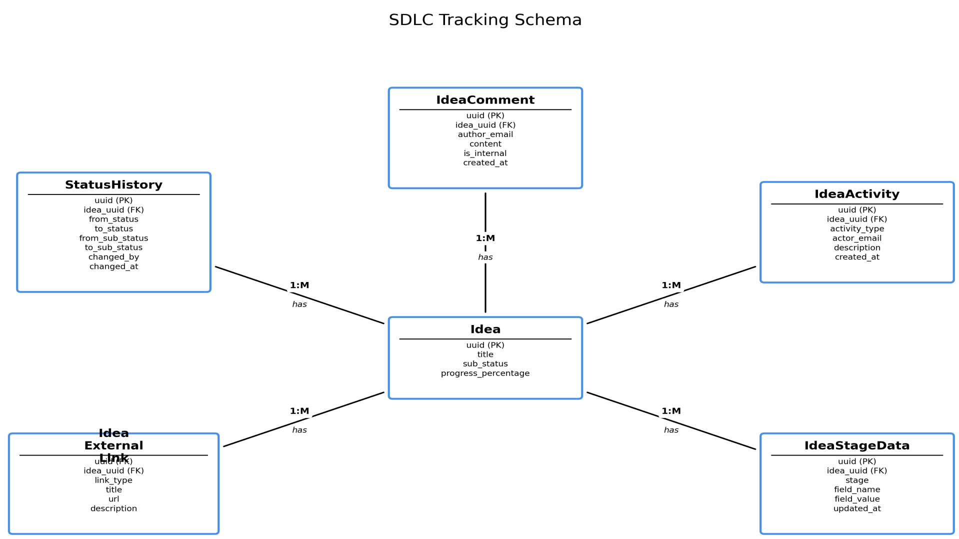
- Enhanced security - IDs cannot be guessed or enumerated
- Better scalability - No ID conflicts in distributed systems
- Improved data portability - IDs remain unique across environments
- Prevention of ID-based attacks - No sequential patterns to exploit

Migration Details:

All primary keys and foreign keys were converted from INTEGER to VARCHAR(36) to store UUID strings. The migration included: - New database file: posting_board_uuid.db - All models updated with UUID fields - API endpoints modified to accept/return UUIDs - Session variables renamed with _uuid suffix - Backward compatibility maintained in API responses

3.3 SDLC Tracking Schema

The SDLC (Software Development Life Cycle) tracking schema enables comprehensive monitoring of idea progress through development stages:



Key SDLC Tables:

StatusHistory:

Tracks all status and sub-status changes with timestamps

IdeaComment:

Threaded discussions and internal notes on ideas

IdeaActivity:

Comprehensive activity feed of all changes

IdeaExternalLink:

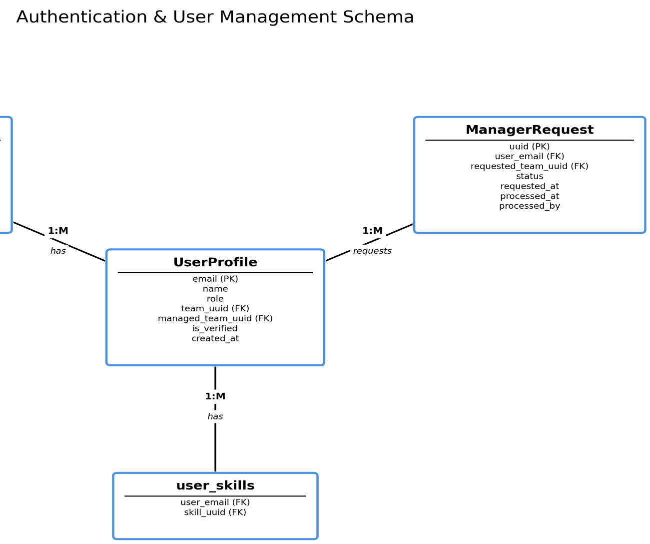
Links to external resources (repos, docs, etc.)

IdeaStageData:

Stage-specific data for each development phase

3.4 Authentication Schema

The authentication system uses a passwordless design with email verification:



Authentication Tables:

UserProfile:

Stores user information, roles, and team assignments

VerificationCode:

Temporary codes for email verification

ManagerRequest:

Tracks requests to become team managers

Notification:

User notifications for various system events

3.5 Relationships and Constraints

The database enforces referential integrity through foreign key constraints and implements several important business rules:

Key Relationships:

- Ideas belong to Teams (many-to-one)
- Ideas have Skills (many-to-many through idea_skills)
- Users have Skills (many-to-many through user_skills)
- Claims link Users to Ideas (many-to-one both ways)
- Notifications belong to Users (many-to-one)
- Bounties belong to Ideas (one-to-one)

Business Rule Constraints:

- Email addresses must be unique in UserProfile
- Team names must be unique
- Skill names must be unique
- Ideas cannot be claimed multiple times simultaneously
- Bounties over $50 require approval
- Managers can only manage one team

4. User Interface

4.1 Design Principles

The user interface follows modern design principles to ensure a professional, intuitive experience across all devices:

Minimalist Design:

Clean, uncluttered interface focusing on content

Consistent Typography:

Unified font sizing and spacing throughout

Professional Aesthetics:

Enterprise-appropriate colors and styling

High Accessibility:

Strong contrast ratios and clear visual hierarchy

Responsive Layout:

Mobile-first approach with fluid grid systems

Intuitive Navigation:

Clear pathways and contextual actions

Color Palette:

Element	Color	Hex Code
Primary	Professional Blue	#4a90e2
Navigation	Dark Navy	#1a1d23

Background	Light Gray	#f8f9fa
Success	Green	#28a745
Warning	Orange	#f0ad4e
Danger	Red	#dc3545

4.2 Navigation Structure

The application uses a consistent navigation bar across all pages with role-based menu items:

Navigation Items:

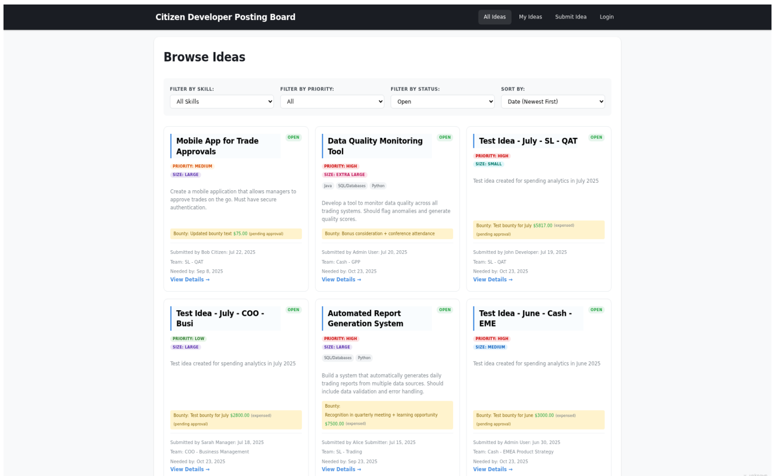
Item	Visibility	Purpose
Browse Ideas	All users	View and filter available ideas
Submit Idea	Authenticated users	Create new idea submissions
My Ideas	Authenticated users	View personal submissions and claims
My Team	Managers and admins	Team analytics and management
Admin	Administrators only	System administration portal
Profile/Login	Context-sensitive	User profile or login prompt

4.3 Key Pages

The platform consists of several key pages, each designed for specific user tasks:

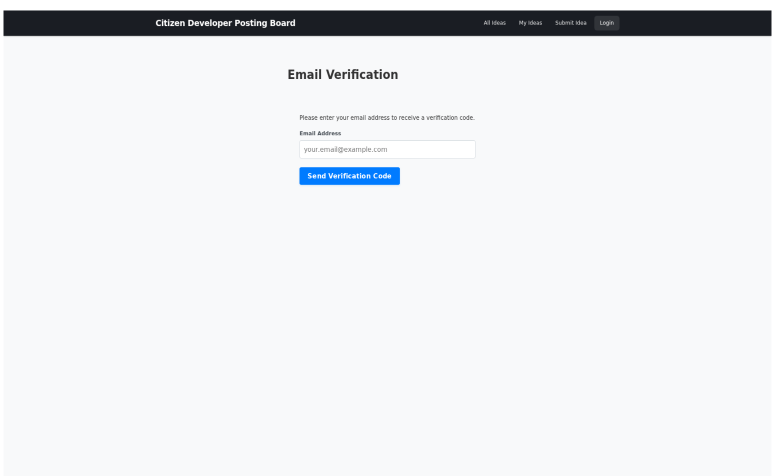
Home Page

Browse and filter ideas, view statistics



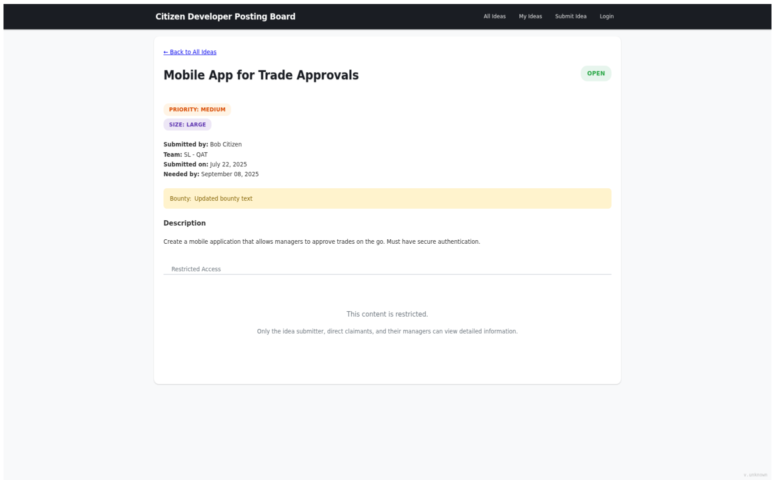
Submit Idea

Create new idea with details and skills



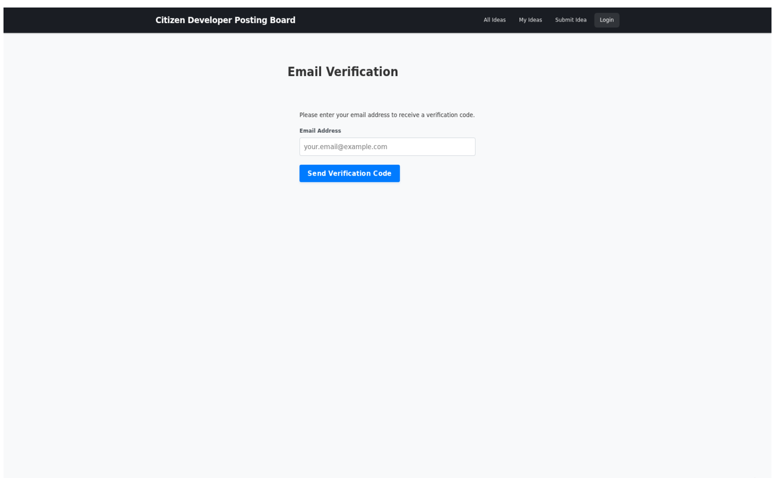
Idea Detail

View full idea information and claim



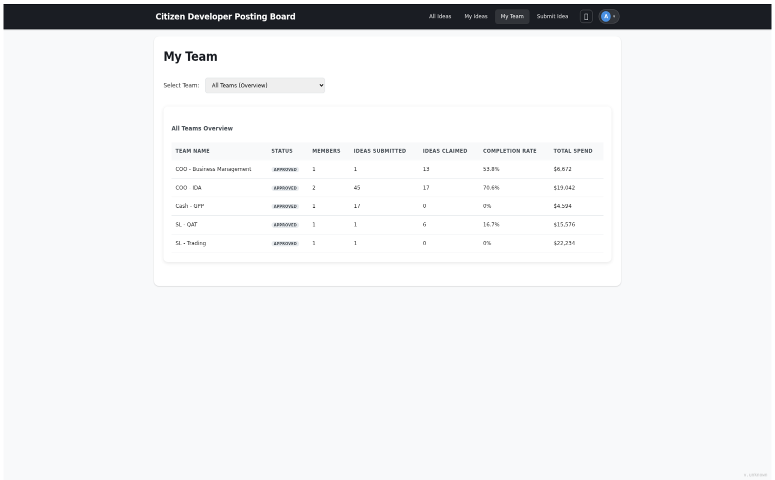
My Ideas

Personal dashboard for submissions and claims



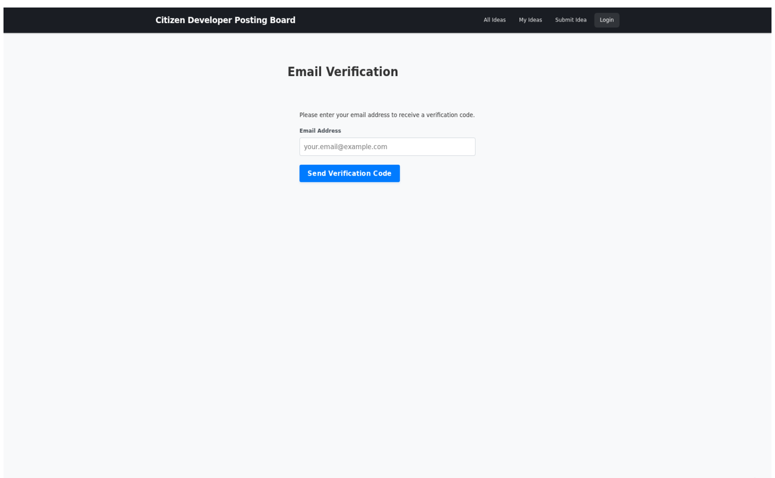
My Team

Team analytics and member management



Profile

User profile and settings management



4.4 Responsive Design

The interface adapts seamlessly across different screen sizes using modern CSS techniques:

Responsive Features:

- CSS Grid with auto-fit for idea cards
- Flexible navigation with mobile menu
- Touch-friendly buttons and controls
- Readable typography scaling
- Optimized images and assets
- Horizontal scrolling for wide tables

Breakpoints:

```
/* Mobile first approach */ @media (min-width: 576px) { /* Small devices */ }
@media (min-width: 768px) { /* Medium devices */ } @media (min-width: 992px) {
/* Large devices */ } @media (min-width: 1200px) { /* Extra large devices */ }
```

4.5 Accessibility

The platform is designed to be accessible to users with disabilities, following WCAG 2.1 guidelines:

Accessibility Features:

Semantic HTML:

Proper heading hierarchy and landmark regions

Keyboard Navigation:

All interactive elements accessible via keyboard

Screen Reader Support:

Descriptive labels and ARIA attributes

Color Contrast:

Minimum 4.5:1 ratio for normal text

Focus Indicators:

Clear visual focus states for navigation

Error Messages:

Clear, descriptive error text near form fields

Alternative Text:

Descriptive alt text for all images

Consistent Layout:

Predictable navigation and page structure

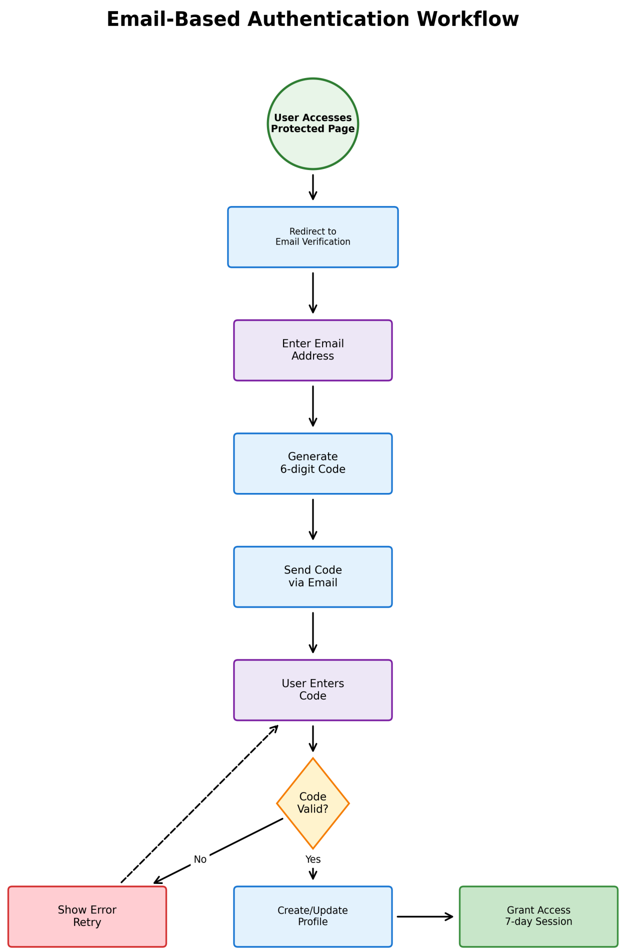
Testing Approach:

Regular accessibility testing is performed using automated tools (axe, WAVE) and manual testing with screen readers (NVDA, JAWS) to ensure compliance and usability for all users.

5. Core Workflows

5.1 Authentication Workflow

The platform uses a passwordless authentication system based on email verification, providing both security and convenience:



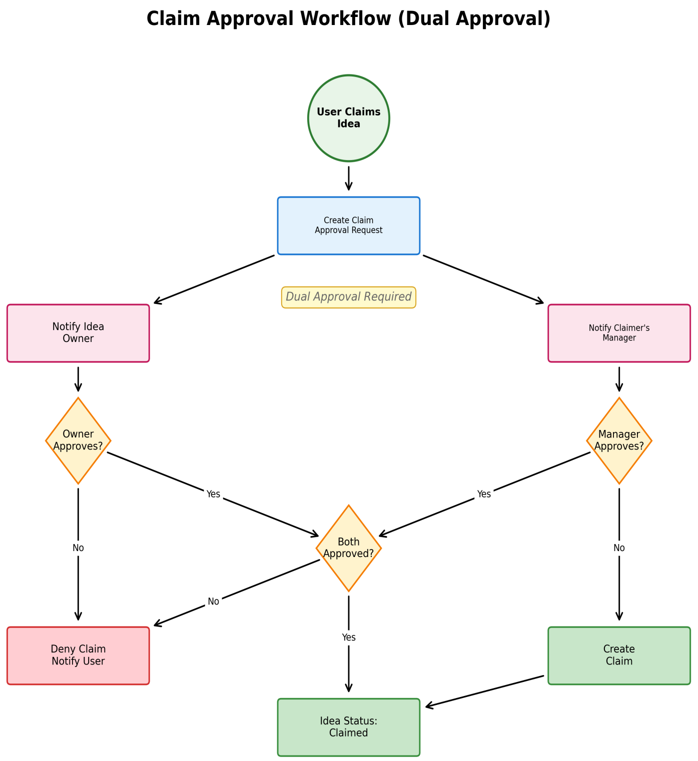
Key Steps:

1. User attempts to access protected resource
2. System redirects to email verification page
3. User enters email address
4. System generates 6-digit verification code
5. Code sent via email (3-minute expiry)
6. User enters code to verify identity

7. Profile created/updated on first login
8. 7-day session established

5.2 Claim Approval Workflow

The dual approval system ensures proper oversight when developers claim ideas:



Approval Requirements:

Idea Owner Approval:

Confirms the claimer can work on their idea

Manager Approval:

Ensures team member has capacity and skills

Auto-approval:

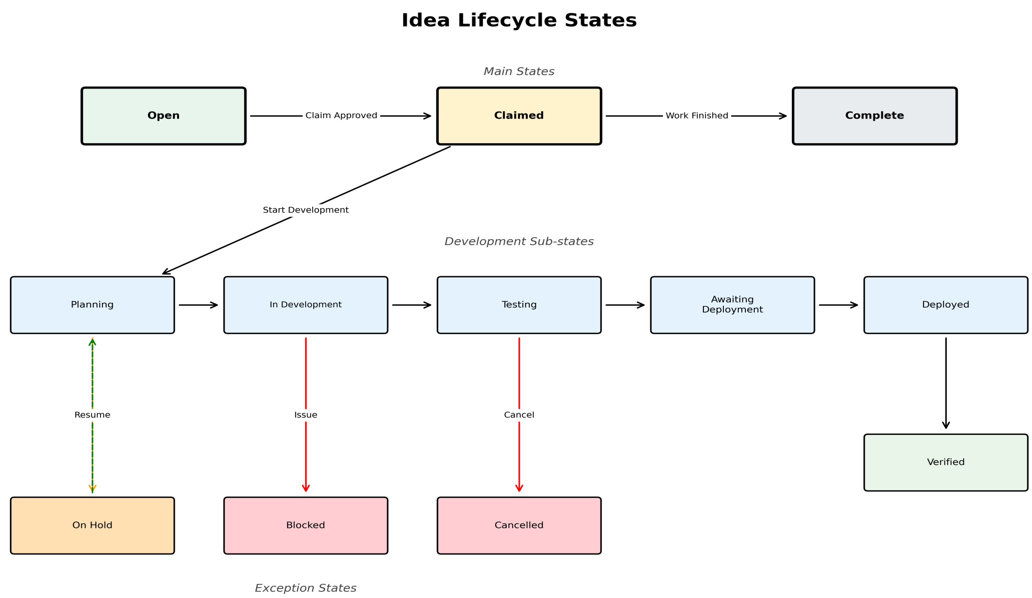
Manager approval auto-granted if no manager assigned

Denial Handling:

Either party can deny, ending the claim request

5.3 Idea Lifecycle

Ideas progress through a defined lifecycle from submission to completion:



Status Progression:

Open:

Idea submitted and available for claiming

Pending Claim:

Claim request awaiting approvals

Claimed:

Approved claim, development can begin

In Progress:

Active development with sub-status tracking

Complete:

Development finished and deployed

Cancelled:

Idea cancelled or withdrawn

5.4 Manager Approval

Users can request to become team managers, requiring admin approval:

Manager Request Process:

1. User selects Manager role in profile
2. Chooses team they want to manage
3. Request submitted to admin queue
4. Admin reviews request and team needs
5. Approval grants access to team analytics
6. Manager can view team performance metrics
7. Manager approves team member claims
8. Manager can assign ideas to team members

Manager Capabilities:

- View comprehensive team analytics
- Approve/deny team member claim requests
- Assign open ideas to team members
- Edit team member profiles
- Track team spending on bounties
- Identify skill gaps for training

5.5 Bounty Approval

Monetary bounties over $50 require manager or admin approval:

Bounty Workflow:

1. Submitter selects 'monetary bounty' checkbox
2. Enters dollar amount for the bounty
3. Optionally marks as 'will be expensed'
4. System checks if amount > $50
5. If yes, creates approval notification
6. Manager/admin reviews and approves
7. Approved bounties tracked in analytics
8. Completion triggers bounty payout

Spending Analytics:

The system tracks all monetary bounties to provide spending insights:

• Total Approved: Sum of all approved bounties

• Pending Approval: Bounties awaiting approval

• Actual Spend: Completed ideas with bounties

• Committed Spend: Claimed ideas with bounties

• Monthly Trends: Historical spending patterns

6. API Documentation

6.1 RESTful Design

The platform provides a comprehensive RESTful API for all dynamic operations, following standard HTTP conventions and JSON data formats:

API Design Principles:

- Resource-based URLs (e.g., /api/ideas, /api/teams)
- HTTP methods for actions (GET, POST, PUT, DELETE)
- JSON request and response bodies
- Consistent error response format
- UUID-based resource identification
- Stateless operations with session authentication

Response Format:

```
{ "success": true, "data": { "id": "550e8400-e29b-41d4-a716-446655440000",
"title": "Automate Report Generation", "status": "open" }, "message": "Idea
retrieved successfully" }
```

6.2 Public Endpoints

These endpoints are accessible without authentication:

Endpoint	Method	Description
/api/ideas	GET	List ideas with filters
/api/skills	GET	Get all available skills
/api/teams	GET	Get approved teams
/api/stats	GET	Platform statistics
/request-code	POST	Request verification code
/verify-code	POST	Verify email code

Query Parameters:

GET /api/ideas supports the following query parameters:

- skill: Filter by skill name
- priority: Filter by priority (low, medium, high)
- status: Filter by status (open, claimed, complete)
- team: Filter by team name
- sort: Sort order (newest, oldest, priority)

6.3 Authenticated Endpoints

These endpoints require a valid session (verified email):

Endpoint	Method	Description
/api/my-ideas	GET	User's submitted/claimed ideas
/api/ideas/<id>/claim	POST	Request to claim an idea
/api/ideas/<id>/sub-status	PUT	Update development status
/api/ideas/<id>/comments	GET/POST	Idea comments
/api/ideas/<id>/activities	GET	Idea activity feed
/api/user/notifications	GET	User notifications
/api/user/notifications/<id>/read	POST	Mark notification read
/profile/update	POST	Update user profile

6.4 Admin Endpoints

Admin endpoints require admin session authentication:

Endpoint	Method	Description
/api/admin/stats	GET	Dashboard statistics
/api/admin/users	GET	List all users
/api/admin/users/<email>	PUT/DELETE	Manage user
/api/admin/ideas/<id>	PUT/DELETE	Manage idea
/api/admin/teams	POST	Create team
/api/admin/teams/<id>	PUT/DELETE	Manage team
/api/admin/skills	POST	Create skill
/api/admin/skills/<id>	PUT/DELETE	Manage skill
/api/admin/bulk-upload	POST	Bulk import CSV

6.5 Error Handling

The API uses consistent error responses with appropriate HTTP status codes:

Error Response Format:

```
{ "success": false, "error": "Validation failed", "message": "Title is
required", "code": "VALIDATION_ERROR" }
```

Common Status Codes:

Code	Meaning	Usage
200	OK	Successful GET/PUT request
201	Created	Successful POST creating resource
400	Bad Request	Invalid request parameters
401	Unauthorized	Authentication required
403	Forbidden	Insufficient permissions
404	Not Found	Resource doesn't exist
422	Unprocessable	Validation errors
500	Server Error	Internal server error

7. Security and Authentication

7.1 Passwordless Authentication

The platform implements a passwordless authentication system that eliminates password-related security risks while maintaining strong security:

Implementation Details:

- 6-digit verification codes generated using secure random
- Codes expire after 3 minutes to limit exposure window
- Maximum 3 active codes per email address
- Rate limiting prevents brute force attempts
- 15-minute cooldown after rate limit reached
- Codes stored hashed in database
- Email delivery for out-of-band verification

Security Benefits:

- No passwords to steal or crack
- Immunity to password reuse attacks
- No need for password complexity rules
- Reduced support burden for password resets
- Improved user experience

7.2 Session Security

Server-side sessions provide secure state management:

Session Configuration:

Storage:

Filesystem-based storage in flask_session directory

Duration:

7-day session lifetime with automatic cleanup

Cookie Security:

HTTPOnly flag prevents JavaScript access

SameSite:

Lax setting prevents CSRF attacks

Session ID:

Cryptographically secure random generation

Rotation:

Session ID rotated on privilege changes

Session Data Protection:

Session data is stored server-side with only the session ID transmitted to the client. This prevents session data tampering and ensures sensitive information like user roles and permissions cannot be modified by clients.

7.3 Role-Based Access Control

The platform implements fine-grained role-based access control:

User Roles:

Role	Capabilities	Restrictions
Admin	Full system access, user management, settings	None
Manager	Team analytics, approve claims, assign ideas	Cannot claim ideas
Developer	Submit and claim ideas, SDLC tracking	Cannot manage teams
Citizen Developer	Submit and claim ideas, basic tracking	Limited to citizen-dev ideas
Idea Submitter	Submit ideas only	Cannot claim ideas

7.4 Data Protection

Multiple layers of protection ensure data security:

Protection Measures:

Input Validation:

All user inputs validated and sanitized

SQL Injection Prevention:

Parameterized queries via SQLAlchemy

XSS Prevention:

Output encoding in templates, CSP headers

UUID Usage:

Non-enumerable identifiers prevent ID attacks

Access Control:

Row-level security for team and user data

Audit Logging:

Comprehensive activity tracking

Data Encryption:

HTTPS in production, encrypted backups

Privacy Considerations:

The platform minimizes data collection and retention:

- Only essential user data collected (email, name, team)

- No tracking cookies or analytics
- User data deletable on request
- Clear data usage policies
- GDPR compliance built-in

7.5 Security Best Practices

Development and deployment follow security best practices:

Development Practices:

- Regular dependency updates and vulnerability scanning
- Code review for all security-sensitive changes
- Automated security testing in CI/CD pipeline
- Secure coding guidelines followed
- Secrets management via environment variables
- No hardcoded credentials or keys

Deployment Security:

- HTTPS enforcement with TLS 1.2+
- Security headers (CSP, X-Frame-Options, etc.)
- Regular security patching schedule
- Intrusion detection monitoring
- Backup encryption and secure storage
- Incident response procedures

8. Team Management

8.1 Team Structure

Teams provide organizational structure and enable departmental oversight of citizen development activities:

Predefined Teams:

Cash - GPP	COO - IDA
COO - Business Management	SL - QAT
SL - Trading	SL - Product
SL - Clients	SL - Tech
Cash - PMG	Cash - US Product Strategy
Cash - EMEA Product Strategy	Cash - Sales
Cash - CMX	

Custom Teams:

Users can create custom teams that require admin approval before use. This flexibility allows the platform to adapt to organizational changes.

8.2 Manager Capabilities

Team managers have enhanced capabilities for overseeing their team's citizen development activities:

Core Management Functions:

View Team Analytics:

Comprehensive dashboard with performance metrics

Approve Claims:

Review and approve team member claim requests

Assign Ideas:

Assign open ideas to specific team members

Edit Profiles:

Update team member names, roles, and skills

Track Spending:

Monitor team's monetary bounty commitments

Identify Gaps:

See missing skills needed for team ideas

Manager Dashboard Features:

The My Team page provides managers with:

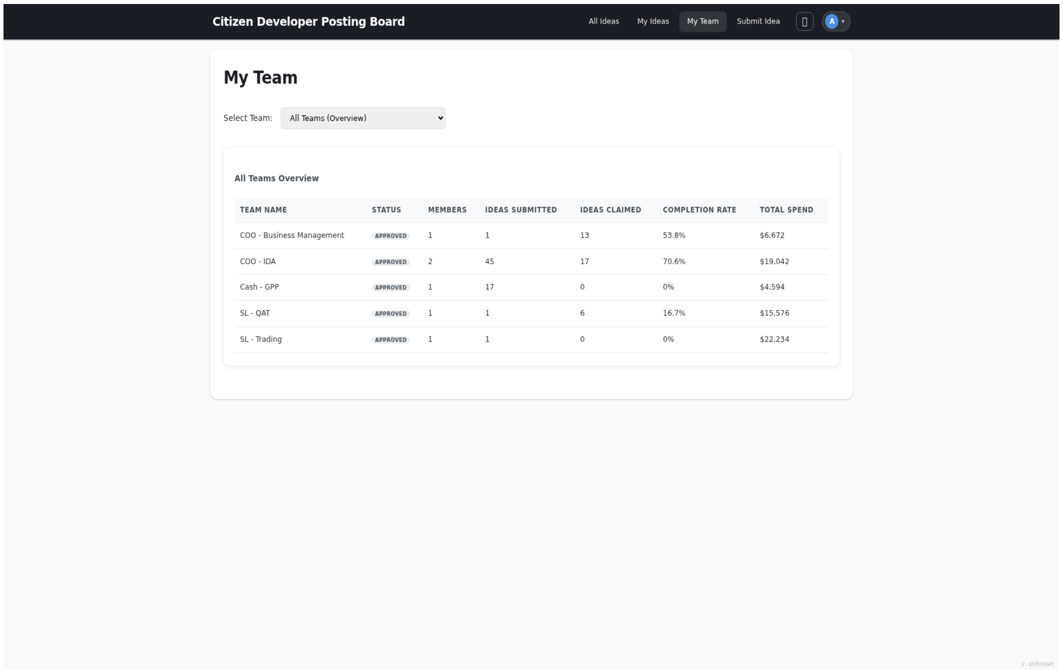
- Team overview cards with key metrics
- Activity charts showing submissions and claims
- Skills analysis with gap identification
- Spending analytics and trends

• Member performance table with totals

• Recent activity tracking (30 days)

8.3 Team Analytics

Comprehensive analytics provide insights into team performance:



Key Metrics Tracked:

Submission Rate:

Ideas submitted per team member

Claim Rate:

Percentage of submitted ideas claimed

Completion Rate:

Percentage of claimed ideas completed

Cross-team Collaboration:

Claims from other teams

Skill Coverage:

Percentage of needed skills available

Spending Metrics:

Monetary bounties approved and spent

8.4 Skills Gap Analysis

The platform automatically identifies skill gaps to inform training and hiring decisions:

Gap Analysis Features:

- Side-by-side comparison of team skills vs. needed skills
- Visual highlighting of missing skills in red
- Tooltips showing 'Gap: Team lacks this skill'
- Quantification of demand for each skill
- Trending analysis of skill requirements
- Export capability for HR planning

Using Gap Analysis:

Managers can leverage skill gap data to:

1. Identify training needs for current team members
2. Justify hiring requests with data-driven insights
3. Prioritize skill development initiatives
4. Plan cross-team collaboration for missing skills
5. Track skill development progress over time

8.5 Performance Metrics

Detailed performance tracking enables data-driven team management:

Individual Metrics:

Metric	Description	Usage
Ideas Submitted	Total ideas created	Identify active contributors
Total Claimed	All claims by member	Measure engagement
Own Team Claims	Internal team support	Team collaboration
Other Team Claims	Cross-team help	Organizational impact
Completed	Successfully delivered	Delivery track record
Total Activity	All platform actions	Overall participation

9. Notification System

9.1 Notification Types

The platform provides comprehensive notifications to keep users informed of important events and required actions:

User Notification Categories:

Claim Requests:

When someone wants to claim your idea

Approval Decisions:

When your claim is approved or denied

Status Changes:

When idea status changes

Assignments:

When manager assigns idea to you

Team Updates:

New team members or role changes

Bounty Approvals:

Status of monetary bounty requests

Comments:

New comments on your ideas

Mentions:

When you're mentioned in discussions

9.2 Real-time Updates

Notifications update in near real-time to ensure timely responses:

Implementation Details:

- 30-second auto-refresh interval for notification bell
- Unread count badge shows pending notifications
- Click notification to mark as read and navigate
- 7-day retention for read notifications
- Persistent storage in database
- Session-based caching for performance

Notification UI:

The notification interface includes:

- Bell icon with unread count in navigation
- Sliding panel with notification list
- Time-based sorting (newest first)
- Visual distinction for read/unread
- Direct navigation to related content
- Bulk actions for managing notifications

9.3 Email Integration

Email notifications complement in-app notifications for critical events:

Email Triggers:

- Verification codes for authentication
- Claim approval requests (configurable)
- High-priority idea assignments
- Bounty approval requirements
- Weekly summary digests (optional)
- System maintenance notifications

Email Configuration:

```
MAIL_SERVER = 'smtp.company.com' MAIL_PORT = 587 MAIL_USE_TLS = True
MAIL_USERNAME = 'postingboard@company.com' MAIL_DEFAULT_SENDER = 'Posting
Board '
```

9.4 User Preferences

Users can customize their notification preferences:

Configurable Settings:

Email Frequency:

Immediate, daily digest, or disabled

Notification Types:

Toggle categories on/off

Quiet Hours:

Suppress non-critical notifications

Mobile Push:

Enable mobile app notifications

Desktop Alerts:

Browser push notifications

Sound Alerts:

Audio cues for new notifications

Default Preferences:

New users start with sensible defaults:

- All notification types enabled
- Email for critical events only
- No quiet hours configured
- Push notifications opt-in

• Sound alerts disabled

9.5 Admin Notifications

Administrators receive special notifications for platform management:

Admin Alert Types:

- New user registrations requiring approval
- Custom team creation requests
- Manager role requests
- High-value bounty approvals
- System errors or anomalies
- Bulk upload results
- Security events

Admin Dashboard Integration:

The admin dashboard prominently displays:

- Yellow notification banner for pending actions
- Count badges on navigation items
- Consolidated pending items view
- Quick action buttons for common tasks
- Audit trail of admin actions
- System health indicators

10. SDLC Tracking Features

10.1 Sub-Status System

Once claimed, ideas progress through detailed development stages with comprehensive tracking:

Development Sub-Statuses:

Status	Description	Progress
planning	Requirements gathering and design	10%
in_development	Active coding and implementation	30%
testing	QA and user acceptance testing	60%
awaiting_deployment	Ready for production release	80%
deployed	Released to production environment	90%
verified	Confirmed working by stakeholders	100%
on_hold	Temporarily paused	Current
blocked	Blocked by dependencies	Current
cancelled	Development cancelled	0%
rolled_back	Deployment reverted	0%

10.2 Progress Tracking

Visual progress indicators keep stakeholders informed:

Progress Features:

- Automatic progress percentage based on sub-status
- Manual progress override capability
- Visual progress bars with color coding
- Expected completion date tracking
- Duration tracking between stages
- Historical progress timeline
- Blocked reason documentation
- Progress update notifications

Stage-Specific Data:

Each development stage can capture specific information:

- Planning: Requirements docs, design specs
- Development: Repository URLs, branch names
- Testing: Test plans, defect counts
- Deployment: Release notes, target environment
- Verification: Sign-off notes, performance metrics

10.3 GANTT Charts

Interactive GANTT charts provide visual project timelines:

GANTT Chart Features:

- SVG-based rendering for full interactivity
- Automatic timeline calculation based on idea size
- Color-coded phases (green=complete, yellow=active, etc.)
- Hover tooltips with phase details
- Click phases to view/edit stage data
- Today marker and progress indicators
- PNG export for reporting
- Customizable phase durations

Timeline Calculation:

Phase durations are automatically calculated based on idea size:

- Small ideas: 2-3 week total timeline
- Medium ideas: 4-6 week timeline
- Large ideas: 8-12 week timeline
- Extra Large: 12-16 week timeline

Managers can customize these defaults per idea.

10.4 Comments and Activity

Comprehensive discussion and activity tracking enables collaboration:

Comment System:

- Threaded discussions on each idea
- Internal notes option for sensitive information
- Markdown formatting support
- File attachment capabilities
- @mentions for notifications
- Edit/delete own comments
- Comment history tracking
- Search within comments

Activity Feed:

JIRA-style activity feed tracks all changes:

- Status and sub-status changes
- User assignments and claims
- Comment additions
- External link additions
- Progress updates
- Field modifications
- Visual timeline with icons

10.5 External Links

Ideas can be linked to external resources for comprehensive tracking:

Supported Link Types:

Repository:

Git repositories and branches

Pull Request:

Code review and merge requests

ADO Work Item:

Azure DevOps integration

Documentation:

Design docs, wikis, specs

Test Results:

Test reports and coverage

Monitoring:

Dashboards and metrics

Other:

Any other relevant resources

Link Management:

External links include:

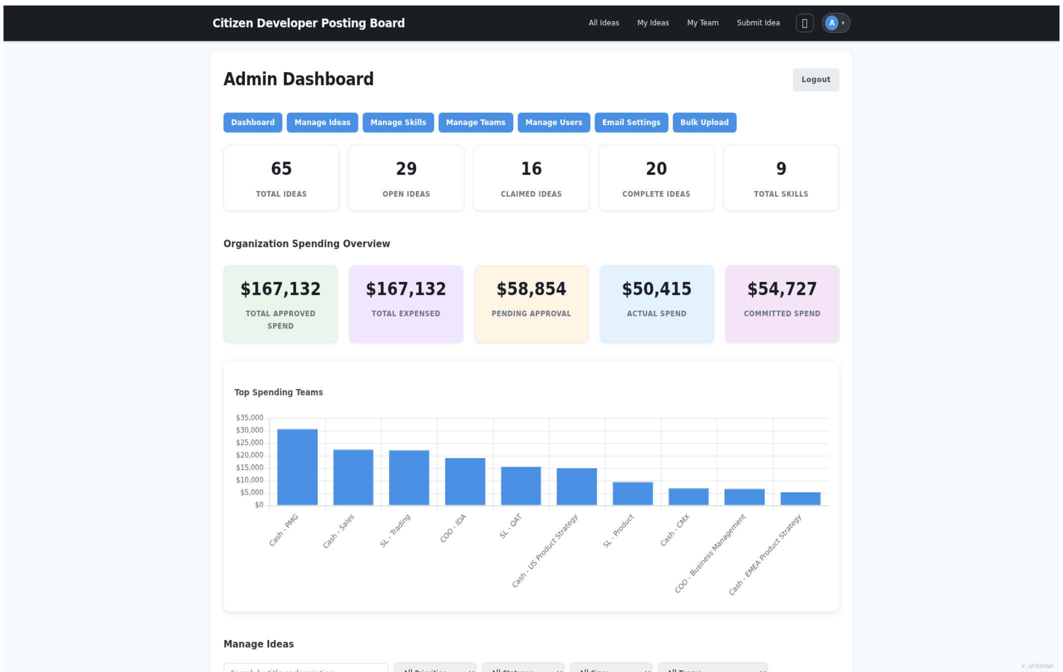
- Categorized display with icons

• Title and description fields
• Automatic metadata extraction
• Link validation and health checks
• Access control based on permissions
• Integration with activity feed

11. Admin Portal

11.1 Dashboard Overview

The admin dashboard provides comprehensive platform oversight:



Dashboard Components:

- Platform statistics cards (ideas, users, teams, skills)
- Idea distribution charts by status and priority
- Recent activity timeline
- Pending approval notifications
- System health indicators
- Quick action buttons
- Spending analytics overview

11.2 User Management

Comprehensive user administration capabilities:

User Management Features:

- Searchable user list with filters
- Edit user profiles and roles
- Manage team assignments
- Approve manager requests
- View user activity metrics
- Reset verification status
- Delete users with cascade
- Export user data

Bulk User Operations:

CSV import capabilities for user management:

- Download template with required fields
- Import multiple users at once
- Automatic validation and error reporting
- Team and skill assignment
- Email notification options

11.3 Idea Management

Full control over all ideas in the system:

Idea Administration:

- View and filter all ideas (not just open)
- Edit any idea field including description
- Manage skill assignments
- Approve monetary bounties
- Unclaim ideas to reset status
- Delete ideas with confirmation
- Reassign idea ownership
- Override status restrictions

Advanced Filtering:

Multi-dimensional filtering capabilities:

- Text search in title and description
- Filter by status, priority, size
- Team and skill filters
- Date range filters
- Bounty amount ranges
- Assignee filters

11.4 Bulk Operations

Efficient bulk data management through CSV import/export:

Bulk Import Features:

Ideas Import:

Create multiple ideas with all fields

Users Import:

Add users with profiles and skills

Skills Import:

Bulk create new skills

Teams Import:

Add custom teams in bulk

Validation:

Pre-import validation with error reporting

Rollback:

Transaction-based imports with rollback

Templates:

Downloadable CSV templates

Results:

Detailed import success/failure report

Export Capabilities:

Data export options include:

- Full database export for backups
- Filtered data exports
- Analytics reports in Excel format
- Audit logs and activity reports

11.5 System Settings

Configure platform-wide settings and integrations:

Configuration Options:

Email Settings:

SMTP configuration for notifications

Session Timeout:

Adjust session duration limits

Rate Limits:

Configure API and auth rate limits

Approval Thresholds:

Bounty approval amount limits

UI Customization:

Branding and theme options

Integration Settings:

External system connections

Feature Flags:

Enable/disable platform features

Maintenance Mode:

Temporary access restrictions

System Monitoring:

Built-in monitoring capabilities:

- Application health checks
- Performance metrics
- Error tracking and alerts
- Usage analytics
- Database statistics

12. Deployment Guide

12.1 Prerequisites

System requirements for deploying the Citizen Developer Posting Board:

Software Requirements:

- Python 3.8 or newer (3.12 recommended)
- Docker and Docker Compose (for containerized deployment)
- Git for version control
- SQLite (included with Python)
- Modern web browser for client access

Hardware Requirements:

Component	Minimum	Recommended
CPU	2 cores	4+ cores
RAM	2 GB	8 GB
Storage	10 GB	50 GB SSD
Network	100 Mbps	1 Gbps

12.2 Docker Deployment

Recommended deployment method using Docker containers:

Quick Start:

```
# Clone repository git clone https://github.com/company/posting-board.git cd
posting-board # Start services ./start-flask.sh # Access application #
http://localhost:9094
```

Docker Compose Configuration:

```
version: '3.8' services: app: build: ./backend ports: - "9094:9094" volumes: -
./backend/data:/app/data - ./backend/flask_session:/app/flask_session
environment: - FLASK_ENV=production - SECRET_KEY=${SECRET_KEY}
```

12.3 Production Configuration

Essential configuration for production deployments:

Security Configuration:

- Generate strong SECRET_KEY for sessions
- Enable HTTPS with valid SSL certificates
- Configure firewall rules (only 443/80 open)
- Set secure session cookie flags
- Enable CORS restrictions
- Configure rate limiting
- Regular security updates

Performance Optimization:

- Use Gunicorn with multiple workers
- Enable response caching where appropriate
- Configure CDN for static assets
- Database connection pooling
- Regular database maintenance
- Monitor and scale based on load

13. Best Practices

13.1 Development Guidelines

Follow these guidelines for maintaining and extending the platform:

Code Standards:

- Follow PEP 8 for Python code style
- Use meaningful variable and function names
- Add docstrings to all functions and classes
- Keep functions small and focused
- Write unit tests for new functionality
- Use type hints where applicable
- Handle exceptions gracefully
- Log important operations and errors

Git Workflow:

- Use feature branches for development
- Write clear, descriptive commit messages
- Include issue numbers in commits
- Review code before merging
- Keep main branch stable
- Tag releases with semantic versioning

13.2 Performance Optimization

Optimize platform performance through these practices:

Database Optimization:

- Index frequently queried columns
- Use eager loading to prevent N+1 queries
- Implement query result caching
- Regular VACUUM operations on SQLite
- Monitor slow queries
- Optimize complex joins

Frontend Optimization:

- Minimize JavaScript and CSS files
- Use browser caching effectively
- Lazy load images and resources
- Implement pagination for large lists
- Optimize API calls with batching
- Use debouncing for search inputs

14. Troubleshooting

14.1 Common Issues

Solutions to frequently encountered problems:

Authentication Issues:

Verification code not received: Check SMTP settings and email logs

Session expiring early: Verify session timeout configuration

Cannot access protected pages: Clear browser cookies and re-authenticate

Email already registered: User may have existing profile

Database Issues:

Ideas not showing: Check enum case sensitivity in queries

Foreign key errors: Ensure related records exist

Migration failures: Run database initialization script

Performance degradation: Run VACUUM and analyze queries

14.2 Debug Procedures

Systematic approaches to debugging platform issues:

Enable Debug Logging:

```
# In app.py app.config['DEBUG'] = True app.logger.setLevel(logging.DEBUG) #
Check logs docker logs postingboard-flask-app-1 --tail 100 -f
```

Common Debug Commands:

- Check container status: docker ps
- View application logs: docker logs [container]
- Access container shell: docker exec -it [container] bash
- Test database connection: python -c 'from database import get_session'
- Verify file permissions: ls -la data/
- Check port availability: netstat -tlnp | grep 9094

14.3 Support Resources

Resources available for additional help:

Documentation:

- This comprehensive guide
- Inline code documentation
- API endpoint documentation
- README files in repository
- CHANGELOG for version history

Support Channels:

GitHub Issues:

Report bugs and request features

Internal Wiki:

Additional guides and FAQs

Slack Channel:

#citizen-dev-platform for discussions

Email Support:

platform-support@company.com

Office Hours:

Weekly Q&A; sessions

Appendices

A. Glossary

Citizen Developer:

Business user with technical skills who creates applications

Claim:

Request to work on an idea requiring dual approval

Bounty:

Reward offered for completing an idea (monetary or non-monetary)

Sub-status:

Detailed development stage within claimed status

SDLC:

Software Development Life Cycle

UUID:

Universally Unique Identifier used for all primary keys

Manager:

Team leader with oversight capabilities

Sprint:

Time-boxed development iteration

Backlog:

Queue of ideas waiting to be claimed

B. Configuration Reference

Complete list of configuration options:

```
# Flask Configuration SECRET_KEY = 'your-secret-key-here' SESSION_TYPE =
'filesystem' SESSION_FILE_DIR = './flask_session' PERMANENT_SESSION_LIFETIME =
timedelta(days=7) # Database DATABASE_URL =
'sqlite:///data/posting_board_uuid.db' # Email MAIL_SERVER =
'smtp.company.com' MAIL_PORT = 587 MAIL_USE_TLS = True MAIL_USERNAME =
'postingboard@company.com' # Application APP_PORT = 9094 UPLOAD_FOLDER =
'./uploads' MAX_CONTENT_LENGTH = 16 * 1024 * 1024 # 16MB
```

C. Database Schema Reference

Complete database schema with all tables and fields:

Core Tables:

- ideas - Main idea storage
- users - User profiles and authentication
- teams - Organizational teams
- skills - Available skills
- claims - Idea claim records
- claim_approvals - Approval workflow
- notifications - User notifications
- bounties - Monetary bounty tracking
- status_history - Status change audit
- idea_comments - Discussion threads
- idea_activities - Activity feed
- idea_external_links - External resources