Citizen Developer Posting Board

Technical Documentation

System Launch Edition

Version 1.0 - Launch Release
Generated: July 26, 2025

Table of Contents

Section	Title	Page
1.	Executive Summary	3
	1.1 System Overview	3
	1.2 Launch Objectives	3
	1.3 Expected Benefits	4
	1.4 Success Metrics	4
2.	Technical Architecture	5
	2.1 Technology Stack	5
	2.2 Component Overview	7
	2.3 Flask Architecture	8
	2.4 Session Management	9
3.	Database Design	10
	3.1 Core Schema	10
	3.2 Entity Relationships	11
	3.3 UUID Implementation	12
	3.4 SDLC Extensions	13
	3.5 Data Integrity	14
4.	User Interface	15
	4.1 Browse Ideas	15
	4.2 Submit Ideas	16
	4.3 My Ideas Dashboard	17
	4.4 Team Analytics	18
	4.5 Mobile Responsiveness	19
5.	Core Workflows	20
	5.1 Authentication Flow	20
	5.2 Idea Lifecycle	21

	5.3 Claim Process	22
	5.4 Manager Approval	23
	5.5 Bounty Approval	24
6.	API Documentation	25
	6.1 RESTful Design	25
	6.2 Public Endpoints	26
	6.3 Protected Endpoints	27
	6.4 Admin Endpoints	28
	6.5 Error Handling	29
7.	Security and Authentication	30
	7.1 Passwordless Authentication	30
	7.2 Session Security	31
	7.3 Role-Based Access Control	32
	7.4 Data Protection	33
	7.5 Security Best Practices	34
8.	SDLC Features	35
	8.1 Sub-Status Tracking	35
	8.2 Development Progress	36
	8.3 Interactive GANTT Charts	37
	8.4 Comments System	38
	8.5 Activity Tracking	39
9.	Admin Portal	40
	9.1 Dashboard Overview	40
	9.2 Idea Management	41
	9.3 User Management	42
	9.4 Team Management	43
	9.5 Analytics & Reporting	44
10.	Deployment Guide	45

	10.1 Requirements	45
	10.2 Native Deployment	46
	10.3 Docker Deployment	47
	10.4 Production Configuration	48
	10.5 Monitoring Setup	49
11.	Performance Optimization	50
	11.1 Database Optimization	50
	11.2 Caching Strategies	51
	11.3 Frontend Performance	52
	11.4 API Performance	53
	11.5 Monitoring & Metrics	54
12.	Troubleshooting	55
	12.1 Common Issues	55
	12.2 Authentication Problems	56
	12.3 Database Issues	57
	12.4 Performance Issues	58
	12.5 Debug Procedures	59
13.	Future Roadmap	60
	13.1 Planned Features	60
	13.2 Integration Plans	61
	13.3 Scalability Roadmap	62
14.	Appendices	63
	14.1 Configuration Reference	63
	14.2 API Response Examples	64
	14.3 Additional Resources	65

1. Executive Summary

1.1 System Overview

The Citizen Developer Posting Board represents a strategic initiative to democratize software development within our organization. This platform creates a marketplace where business teams can post their automation and development needs, while technically-skilled employees from any department can discover and implement these solutions. By leveraging the growing citizen developer movement, this system addresses the critical gap between IT capacity and business demand for digital solutions. The platform facilitates collaboration, skill development, and rapid solution delivery while maintaining appropriate governance and oversight.

1.2 Launch Objectives

The system launch aims to achieve the following key objectives:

- Establish a centralized platform for capturing and prioritizing business automation needs across all departments
- Enable skilled employees to contribute to digital transformation efforts regardless of their formal IT role
- Reduce the backlog of small to medium-sized development requests that don't warrant full IT project resources
- Create visibility into the organization's collective technical capabilities and development capacity
- Foster a culture of innovation and continuous improvement through democratized problem-solving
- Implement appropriate governance and approval workflows for citizen development initiatives

1.3 Expected Benefits

Based on industry benchmarks and pilot program results, we anticipate the following benefits within the first year of operation:

- 30-40% reduction in IT backlog for small automation and development requests
- Identification and enablement of 50+ citizen developers across the organization
- Average 2-3 week reduction in time-to-solution for qualifying projects
- Improved employee engagement through skill development and contribution opportunities
- Enhanced visibility into departmental technology needs and priorities
- Creation of a knowledge base of reusable solutions and best practices

1.4 Success Metrics

The platform will track the following key performance indicators to measure success and guide continuous improvement:

Metric	Target (Year 1)	Measurement Method
Active Citizen Developers	50+ users	Unique users claiming ideas
Ideas Submitted	200+ ideas	Total submissions in system
Ideas Completed	100+ solutions	Ideas marked as complete
Average Time to Claim	< 5 days	Time from submission to claim
Average Completion Time	< 30 days	Time from claim to completion
User Satisfaction	> 80%	Quarterly survey results
Cost Avoidance	$500K+	Compared to external development

The Citizen Developer Posting Board positions our organization at the forefront of the digital transformation movement. By empowering our workforce to participate directly in solution development, we create a sustainable competitive advantage through increased agility, innovation, and employee engagement. This platform serves as the foundation for a broader citizen development program that will evolve based on user feedback and organizational needs.

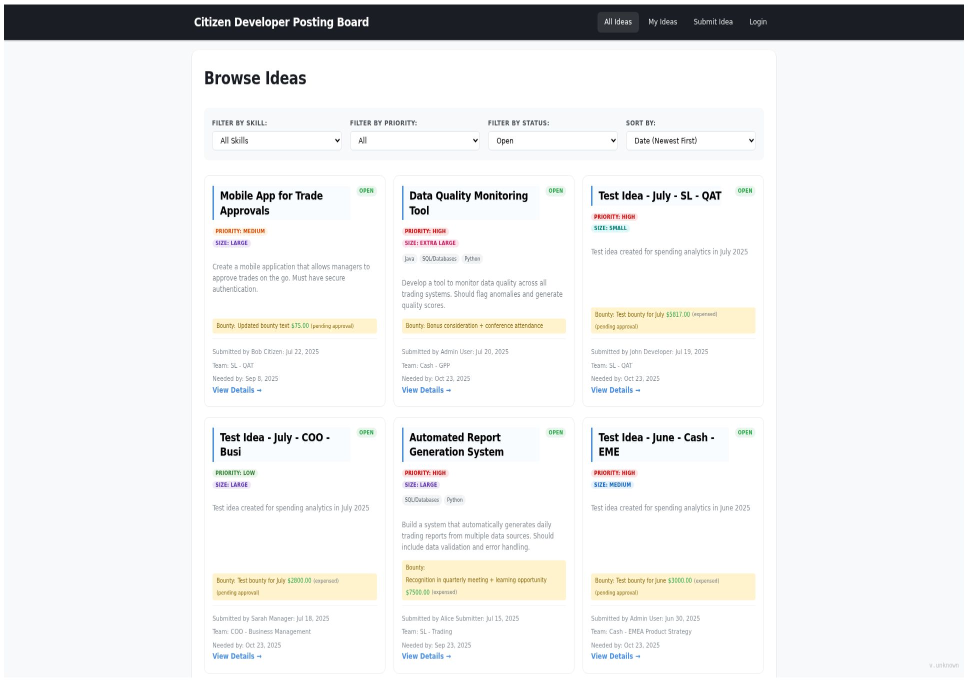
2. Technical Architecture

2.1 Technology Stack

The platform is built on a modern, maintainable technology stack that prioritizes developer productivity, system reliability, and operational simplicity:

Layer	Technology	Purpose
Backend Framework	Flask 2.3+	Lightweight Python web framework for rapid development
Database	SQLite/PostgreSQL	Embedded database for development, PostgreSQL for production
ORM	SQLAlchemy	Database abstraction layer with migration support
Frontend	Jinja2 + JavaScript	Server-side templating with dynamic client interactions
Session Management	Flask-Session	Secure server-side session storage with Redis support
Authentication	Custom Email-based	Passwordless authentication for enhanced security
Deployment	Docker + Gunicorn	Containerized deployment with production WSGI server
Version Control	Git	Source code management and collaboration

System Homepage



Browse Ideas

FILTER BY SKILL: All Skills
FILTER BY PRIORITY: All
FILTER BY STATUS: Open
SORT BY: Date (Newest First)

Mobile App for Trade Approvals OPEN
PRIORITY: MEDIUM
SIZE: LARGE
Create a mobile application that allows managers to approve trades on the go. Must have secure authentication.
Bounty: Updated bounty text $75.00 (pending approval)
Submitted by Bob Citizen: Jul 22, 2025
Team: SL - QAT
Needed by: Sep 8, 2025
View Details →

Data Quality Monitoring Tool OPEN
PRIORITY: HIGH
SIZE: EXTRA LARGE
Java SQL/Databases Python
Develop a tool to monitor data quality across all trading systems. Should flag anomalies and generate quality scores.
Bounty: Bonus consideration + conference attendance
Submitted by Admin User: Jul 20, 2025
Team: Cash - GPP
Needed by: Oct 23, 2025
View Details →

Test Idea - July - SL - QAT OPEN
PRIORITY: HIGH
SIZE: SMALL
Test idea created for spending analytics in July 2025
Bounty: Test bounty for July $5817.00 (expensed) (pending approval)
Submitted by John Developer: Jul 19, 2025
Team: SL - QAT
Needed by: Oct 23, 2025
View Details →

Test Idea - July - COO - Busi OPEN
PRIORITY: LOW
SIZE: LARGE
Test idea created for spending analytics in July 2025
Bounty: Test bounty for July $2800.00 (expensed) (pending approval)
Submitted by Sarah Manager: Jul 18, 2025
Team: COO - Business Management
Needed by: Oct 23, 2025
View Details →

Automated Report Generation System OPEN
PRIORITY: HIGH
SIZE: LARGE
SQL/Databases Python
Build a system that automatically generates daily trading reports from multiple data sources. Should include data validation and error handling.
Bounty: Recognition in quarterly meeting + learning opportunity $7500.00 (expensed)
Submitted by Alice Submitter: Jul 15, 2025
Team: SL - Trading
Needed by: Sep 23, 2025
View Details →

Test Idea - June - Cash - EME OPEN
PRIORITY: HIGH
SIZE: MEDIUM
Test idea created for spending analytics in June 2025
Bounty: Test bounty for June $3000.00 (expensed) (pending approval)
Submitted by Admin User: Jun 30, 2025
Team: Cash - EMEA Product Strategy
Needed by: Oct 23, 2025
View Details →

v.unknown

2.4 Session Management

The platform implements a robust session management system that maintains user state across requests while ensuring security and performance. Flask-Session provides server-side session storage, preventing client-side tampering and enabling complex session data structures.

Session Configuration

```
app.config['SESSION_TYPE'] = 'filesystem' app.config['SESSION_PERMANENT'] =
False app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(days=7)
app.config['SESSION_FILE_DIR'] = './flask_session/'
app.config['SESSION_COOKIE_SECURE'] = True # HTTPS only in production
app.config['SESSION_COOKIE_HTTPONLY'] = True # Prevent XSS attacks
app.config['SESSION_COOKIE_SAMESITE'] = 'Lax' # CSRF protection
```

Session Variables Reference

The following session variables are used throughout the application to maintain user state and provide personalized experiences:

user_email (String (required))

The authenticated user's email address, serving as the primary identifier throughout the system. This value is set during the email verification process and persists for the duration of the session. Used for database lookups, audit trails, and access control. Format: Standard email format (e.g., user@company.com). Maximum length: 120 characters.

```
Example: john.doe@company.com
```

user_name (String (optional))

The user's display name as entered in their profile. This human-readable name appears throughout the UI instead of the email address for better user experience. If not set, the system falls back to displaying the email address. Updated when users complete their profile or modify their name. Maximum length: 100 characters.

```
Example: John Doe
```

user_verified (Boolean (required))

Indicates whether the user has successfully completed email verification. Set to True after entering a valid verification code. This flag gates access to protected features like submitting ideas, claiming ideas, and accessing personal dashboards. Unverified users can only browse public content. Reset to False if verification expires.

```
Example: True
```

user_role (String Enum (required))

The user's assigned role within the system, determining their permissions and available features. Valid values: 'manager' (can approve claims and view team data), 'developer' (can claim and implement ideas), 'citizen_developer' (same as developer), 'idea_submitter' (can only submit ideas). Set during profile creation and modifiable by administrators only.

```
Example: developer
```

user_team (String (required))

The user's organizational team assignment. Used for filtering ideas, tracking team performance, and routing approvals. Must match a pre-approved team name or be a custom team approved by administrators. This field drives team-based analytics and access controls. Maximum length: 100 characters.

```
Example: SL - Tech
```

user_team_uuid (UUID String (required))

The unique identifier for the user's team. This 36-character UUID is used for all database relationships and API calls. More secure than using team names directly and prevents issues with team name changes. Format: Standard UUID v4.

```
Example: 123e4567-e89b-12d3-a456-426614174000
```

user_skills (Array of Strings (conditional))

List of technical skills possessed by the user. Required for users with 'developer' or 'citizen_developer' roles. Used for matching users with appropriate ideas and tracking skill gaps within teams. Each skill must match a pre-defined skill in the system. Stored as JSON array in session. Maximum 20 skills per user.

```
Example: ["Python", "JavaScript", "SQL", "APIs"]
```

submitted_ideas (Array of UUIDs (dynamic))

Tracks all idea UUIDs submitted by the user during their session. Provides quick access to user's submissions without database queries. Updated in real-time as users submit new ideas. Persists across session for consistent user experience. Used in My Ideas dashboard for filtering.

```
Example: ["uuid1", "uuid2", "uuid3"]
```

claimed_ideas (Array of UUIDs (dynamic))

Maintains list of idea UUIDs claimed by the user. Updated when claims are approved through the dual-approval process. Enables quick filtering in My Ideas dashboard and prevents duplicate claims. Synchronized with database on session start to handle claims made in other sessions.

```
Example: ["uuid4", "uuid5"]
```

user_managed_team_uuid (UUID String (conditional))

For users with manager role, stores the UUID of the team they manage. Enables access to team analytics, member management, and claim approvals. Set after admin approval of manager request. Null for non-managers. Used to filter team-specific data throughout the application.

```
Example: 456e7890-e89b-12d3-a456-426614174000
```

is_admin (Boolean (required))

Administrative access flag set after successful admin authentication. Grants access to admin portal, user management, system configuration, and all teams' data. Requires separate password authentication beyond email verification. Expires after 2 hours of inactivity for security.

```
Example: False
```

pending_manager_request (Boolean (dynamic))

Indicates if the user has a pending request to become a team manager. Set when users with manager role request to manage a specific team. Cleared when admin approves or denies the request. Displays appropriate notifications in UI while request is pending.

```
Example: True
```

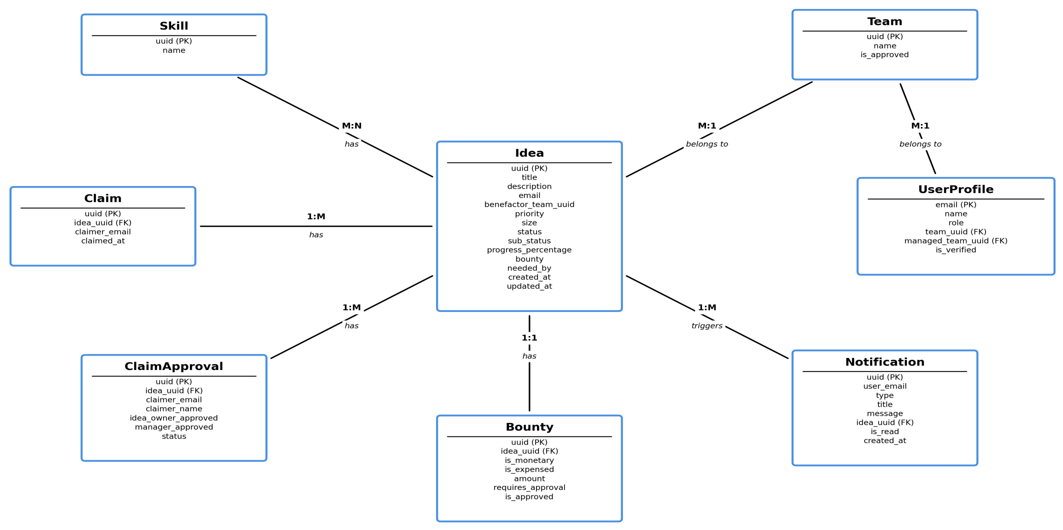
3. Database Design

3.1 Core Schema

The database follows a normalized design with UUID primary keys for enhanced security and scalability. All tables use 36-character UUID strings as primary keys, preventing enumeration attacks and supporting distributed systems. The schema is designed for flexibility and performance with appropriate indexes on foreign keys and commonly queried fields.

Entity Relationship Diagram - Core Tables

Main Database Schema - Core Entities



Core Tables Overview

Table: ideas

Central table storing all submitted ideas and their metadata

Key Fields:

- uuid (PRIMARY KEY): Unique identifier for the idea
- title: Brief, descriptive title (max 200 chars)
- description: Detailed explanation of the need or request
- email: Submitter's email address for ownership tracking
- benefactor_team_uuid: Team that will benefit from this idea

- priority: Enum (low, medium, high) indicating urgency
- size: Enum (small, medium, large, extra_large) estimating effort
- status: Enum (open, claimed, complete) tracking progress
- sub_status: Detailed development stage when claimed
- bounty: Text description of recognition or reward
- created_at: Timestamp of submission
- needed_by: Optional deadline for completion

Relationships: Links to skills, teams, claims, comments, activities

Table: user_profiles

Stores verified user information and preferences

Key Fields:

- email (PRIMARY KEY): User's email address as unique identifier
- name: Display name for UI presentation
- role: User's system role (manager, developer, etc.)
- team_uuid: User's organizational team assignment
- managed_team_uuid: Team managed (for managers only)
- is_verified: Email verification status
- created_at: Account creation timestamp
- last_login: Most recent activity timestamp

Relationships: Links to teams, skills via junction table

Table: teams

Organizational units for grouping users and ideas

Key Fields:

- uuid (PRIMARY KEY): Unique team identifier
- name: Team display name (must be unique)
- is_approved: Admin approval status for custom teams
- created_at: Team creation timestamp
- created_by: Email of user who created team

Relationships: Referenced by users and ideas

Table: skills

Technical capabilities that can be assigned to users and required by ideas

Key Fields:

- uuid (PRIMARY KEY): Unique skill identifier
- name: Skill name (e.g., Python, SQL, APIs)
- category: Optional grouping (e.g., Programming, Database)
- is_active: Whether skill is available for selection

Relationships: Many-to-many with users and ideas

Table: claims

Records approved assignments of ideas to developers

Key Fields:

- uuid (PRIMARY KEY): Unique claim identifier
- idea_uuid: Reference to claimed idea
- claimer_email: Developer who will implement
- claimed_at: Timestamp of approved claim
- completed_at: Timestamp of completion

Relationships: Links ideas to implementing users

Table: bounties

Tracks monetary rewards and expense approvals for ideas

Key Fields:

- uuid (PRIMARY KEY): Unique bounty identifier
- idea_uuid: Reference to associated idea
- is_monetary: Boolean indicating cash value
- amount: Dollar amount if monetary
- is_expensed: Whether amount will be reimbursed
- requires_approval: True if amount > $50
- is_approved: Approval status
- approved_by: Manager/admin who approved

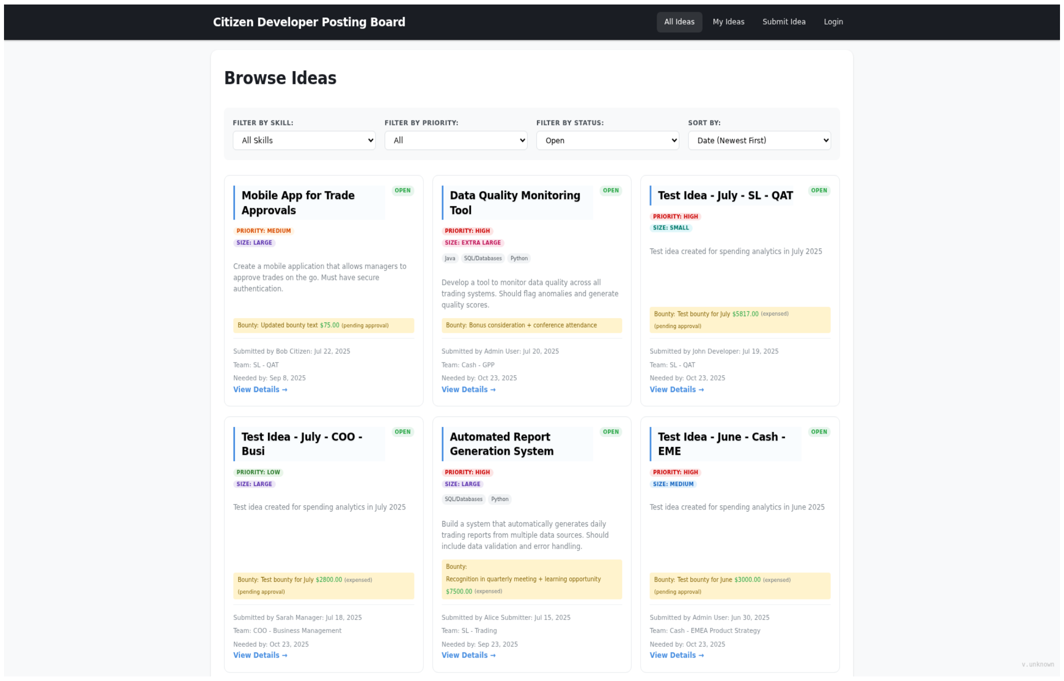
- approved_at: Approval timestamp

Relationships: One-to-one with ideas table

4. User Interface

4.1 Browse Ideas

The home page provides a comprehensive view of all available ideas with advanced filtering and sorting capabilities. Users can quickly identify opportunities that match their skills and interests.



Browse Ideas

FILTER BY SKILL: All Skills
FILTER BY PRIORITY: All
FILTER BY STATUS: Open
SORT BY: Date (Newest First)

Mobile App for Trade Approvals OPEN
PRIORITY: MEDIUM
SIZE: LARGE
Create a mobile application that allows managers to approve trades on the go. Must have secure authentication.
Bounty: Updated bounty text $75.00 (pending approval)
Submitted by Bob Citizen: Jul 22, 2025
Team: SL - QAT
Needed by: Sep 8, 2025
View Details →

Data Quality Monitoring Tool OPEN
PRIORITY: HIGH
SIZE: EXTRA LARGE
Java SQL/Databases Python
Develop a tool to monitor data quality across all trading systems. Should flag anomalies and generate quality scores.
Bounty: Bonus consideration + conference attendance
Submitted by Admin User: Jul 20, 2025
Team: Cash - GPP
Needed by: Oct 23, 2025
View Details →

Test Idea - July - SL - QAT OPEN
PRIORITY: HIGH
SIZE: SMALL
Test idea created for spending analytics in July 2025
Bounty: Test bounty for July $5817.00 (expensed) (pending approval)
Submitted by John Developer: Jul 19, 2025
Team: SL - QAT
Needed by: Oct 23, 2025
View Details →

Test Idea - July - COO - Busi OPEN
PRIORITY: LOW
SIZE: LARGE
Test idea created for spending analytics in July 2025
Bounty: Test bounty for July $2800.00 (expensed) (pending approval)
Submitted by Sarah Manager: Jul 18, 2025
Team: COO - Business Management
Needed by: Oct 23, 2025
View Details →

Automated Report Generation System OPEN
PRIORITY: HIGH
SIZE: LARGE
SQL/Databases Python
Build a system that automatically generates daily trading reports from multiple data sources. Should include data validation and error handling.
Bounty: Recognition in quarterly meeting + learning opportunity $7500.00 (expensed)
Submitted by Alice Submitter: Jul 15, 2025
Team: SL - Trading
Needed by: Sep 23, 2025
View Details →

Test Idea - June - Cash - EME OPEN
PRIORITY: HIGH
SIZE: MEDIUM
Test idea created for spending analytics in June 2025
Bounty: Test bounty for June $3000.00 (expensed) (pending approval)
Submitted by Admin User: Jun 30, 2025
Team: Cash - EMEA Product Strategy
Needed by: Oct 23, 2025
View Details →

v.unknown

Key Features:

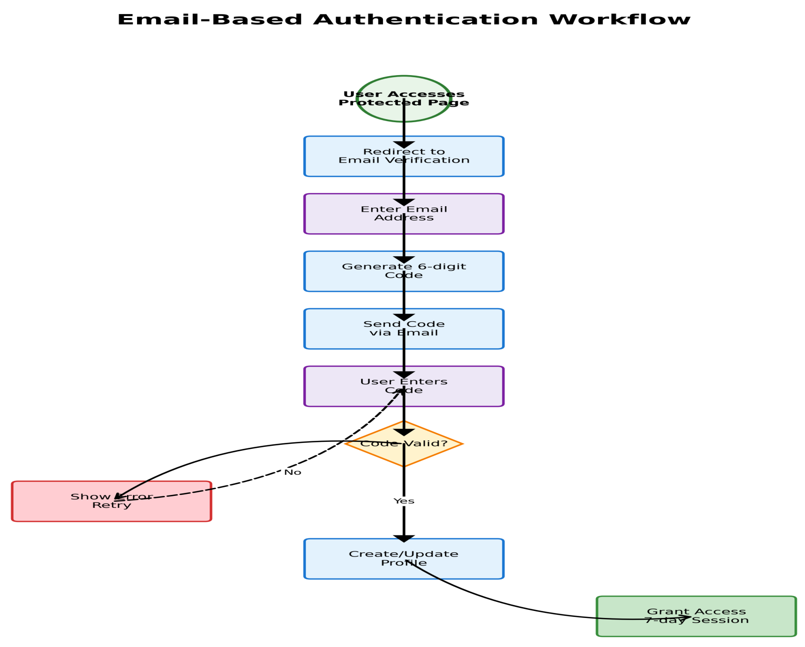
- Real-time filtering by skill, priority, status, and team
- Visual indicators for idea priority and size
- Bounty information prominently displayed
- One-click access to detailed idea information
- Responsive grid layout adapting to screen size
- Auto-refresh to show latest submissions

5. Core Workflows

5.1 Authentication Flow

The platform implements a passwordless authentication system that balances security with user convenience. This approach eliminates password-related vulnerabilities while providing a smooth user experience.

Authentication Workflow Diagram



Authentication Steps:

1. User enters email address on verification page
2. System generates 6-digit verification code
3. Code sent via email (or displayed in console for development)
4. User enters code within 3-minute expiration window
5. System validates code and creates authenticated session
6. User redirected to complete profile if first-time login
7. Session persists for 7 days with sliding expiration

6. API Documentation

6.1 RESTful Design

The platform provides a comprehensive RESTful API following industry best practices for naming conventions, HTTP methods, status codes, and response formats. All endpoints return JSON responses with consistent structure and error handling.

API Design Principles:

- Resource-based URLs using nouns, not verbs
- HTTP methods indicate actions (GET, POST, PUT, DELETE)
- Consistent JSON response format with success indicators
- Detailed error messages with actionable information
- UUID-based resource identification for security
- Pagination support for list endpoints
- Filtering and sorting via query parameters

6.2 Public Endpoints

These endpoints are accessible without authentication:

GET /api/ideas

Retrieves a filtered list of ideas based on query parameters. Returns all ideas by default, or filtered by status, priority, skill, or team. Supports sorting by date, priority, or alphabetical order.

Parameters:

- skill (optional): Filter by required skill name
- priority (optional): Filter by priority level (low, medium, high)
- status (optional): Filter by status (open, claimed, complete)
- benefactor_team (optional): Filter by team UUID
- sort (optional): Sort order (date_desc, date_asc, priority, alphabetical)

Example Response:

```
{ "success": true, "ideas": [ { "id": "uuid-string", "title": "Idea title",
"description": "Detailed description", "priority": "medium", "size":
"large", "status": "open", "skills": ["Python", "SQL"], "bounty":
"Recognition in team meeting", "bounty_details": { "is_monetary": true,
"amount": 500.00, "is_expensed": true, "is_approved": false }, "team_name":
"SL - Tech", "submitter_name": "John Doe", "created_at": "2025-01-26",
"needed_by": "2025-02-15" } ] }
```

GET /api/skills

Returns a list of all available skills in the system. Used for populating filter dropdowns and skill selection interfaces.

Example Response:

```
[ { "id": "uuid-string", "name": "Python", "category": "Programming" }, {
"id": "uuid-string", "name": "SQL", "category": "Database" } ]
```

GET /api/teams

Returns teams based on user role. Admins see all teams with approval status, while regular users see only approved teams.

Example Response:

```
[ { "id": "uuid-string", "name": "SL - Tech", "is_approved": true } ]
```

7. Security and Authentication

7.1 Passwordless Authentication

The platform implements a passwordless authentication system that eliminates common password-related vulnerabilities while maintaining strong security. This approach uses time-limited verification codes sent via email.

Security Features:

- No password storage eliminates breach risks
- 6-digit verification codes with 3-minute expiration
- Rate limiting prevents brute force attempts (3 attempts per hour)
- Email validation ensures valid corporate addresses
- Session tokens use cryptographically secure generation
- HTTPOnly cookies prevent XSS attacks
- CSRF protection via SameSite cookie attribute

8. SDLC Features

8.1 Sub-Status Tracking

The platform includes comprehensive Software Development Life Cycle (SDLC) tracking features that enable detailed monitoring of idea progress through development stages. These features provide transparency and accountability throughout the implementation process.

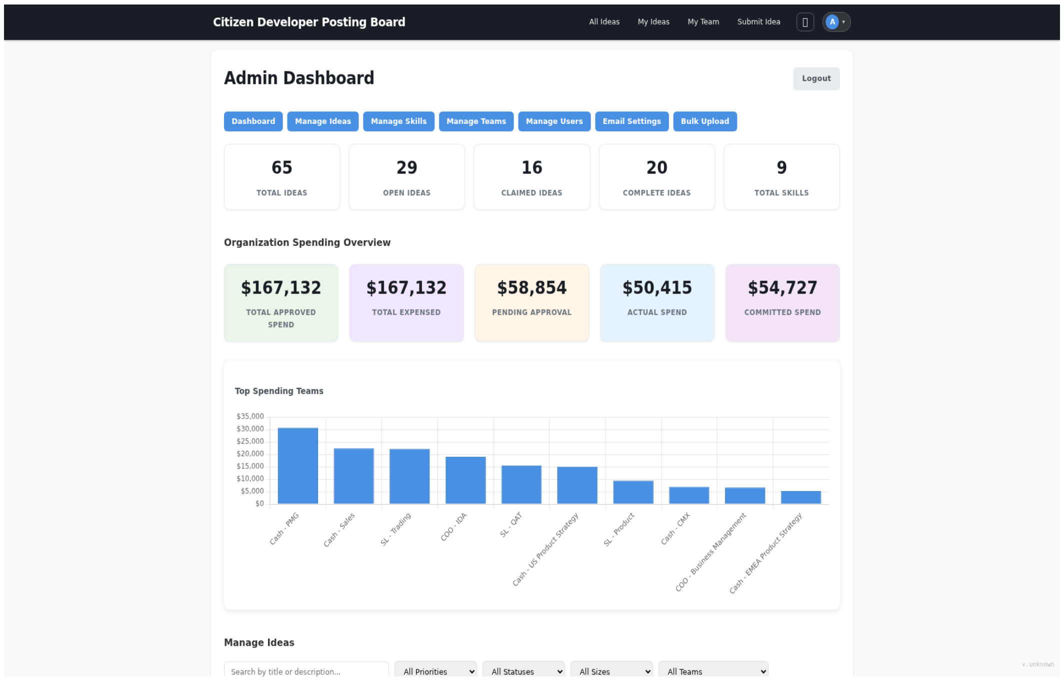
Development Sub-Statuses:

- planning: Initial requirements gathering and design (10% progress)
- in_development: Active development work (30% progress)
- testing: Quality assurance and testing (60% progress)
- awaiting_deployment: Ready for production deployment (80% progress)
- deployed: Deployed to production environment (90% progress)
- verified: Fully tested and verified in production (100% progress)
- on_hold: Temporarily paused (maintains current progress)
- blocked: Blocked by dependencies or issues (maintains progress)
- cancelled: Development cancelled (0% progress)
- rolled_back: Deployment rolled back (0% progress)

9. Admin Portal

9.1 Dashboard Overview

The administrative portal provides comprehensive system management capabilities with real-time analytics, user management, and configuration options. Access is restricted to authorized administrators through additional authentication.



Admin Dashboard

65	29	16	20	9
TOTAL IDEAS	OPEN IDEAS	CLAIMED IDEAS	COMPLETE IDEAS	TOTAL SKILLS

Organization Spending Overview

$167,132	$167,132	$58,854	$50,415	$54,727
TOTAL APPROVED SPEND	TOTAL EXPENSED	PENDING APPROVAL	ACTUAL SPEND	COMMITTED SPEND

Manage Ideas

Key Metrics Displayed:

- Total ideas submitted across all teams
- Open ideas awaiting claims
- Active claims in progress
- Completed ideas with success metrics
- Total registered users and skill distribution
- Spending analytics for monetary bounties
- Team performance comparisons

10. Deployment Guide

10.1 Requirements

The platform is designed for easy deployment in various environments with minimal dependencies. Both development and production configurations are supported.

System Requirements:

- Python 3.8 or higher (3.12 recommended)
- 2GB RAM minimum (4GB recommended)
- 10GB disk space for application and data
- Linux or macOS for production (Windows supported for development)
- PostgreSQL 12+ for production database (SQLite for development)
- SMTP server access for email notifications
- HTTPS certificate for production deployment

11. Performance Optimization

11.1 Database Optimization

The platform implements several optimization strategies to ensure responsive performance even with large datasets and concurrent users.

Optimization Techniques:

- UUID indexes on all foreign key columns
- Composite indexes for common query patterns
- Query result caching for frequently accessed data
- Lazy loading of related entities
- Connection pooling for database connections
- Prepared statements to prevent SQL injection
- Pagination for large result sets

12. Troubleshooting

12.1 Common Issues

This section addresses frequently encountered issues and their solutions to help administrators and developers quickly resolve problems.

Issue: Email verification codes not received

Cause: SMTP configuration incorrect or email server blocking

Solution: Check SMTP settings in config, verify firewall rules, check spam folder

Issue: Session expires unexpectedly

Cause: Session timeout too short or cookie settings incorrect

Solution: Adjust SESSION_LIFETIME in config, verify cookie domain settings

Issue: Cannot claim ideas

Cause: User profile incomplete or wrong role assigned

Solution: Ensure user has developer/citizen_developer role and skills selected

Issue: Admin portal access denied

Cause: Admin session expired or incorrect password

Solution: Re-authenticate with admin password, check session configuration

13. Future Roadmap

13.1 Planned Features

The following features are planned for future releases based on user feedback and organizational needs:

- Microsoft Teams integration for notifications
- AI-powered idea matching and recommendations
- Mobile application for iOS and Android
- Advanced analytics with PowerBI integration
- Automated testing framework for citizen developers
- Template library for common solutions
- Gamification elements to encourage participation
- Integration with corporate SSO systems
- Multi-language support for global teams
- API webhooks for external integrations

14. Appendices

14.1 Configuration Reference

The following configuration options are available for customizing the platform:

DATABASE_URL:
PostgreSQL connection string for production

SECRET_KEY:
Secret key for session encryption (generate unique value)

SMTP_SERVER:
Email server hostname for notifications

SMTP_PORT:
Email server port (typically 587 for TLS)

SMTP_USERNAME:
Email account username

SMTP_PASSWORD:
Email account password

SESSION_LIFETIME:
Session duration in seconds (default: 604800 = 7 days)

VERIFICATION_CODE_EXPIRY:
Code expiration in seconds (default: 180 = 3 minutes)

BOUNTY_APPROVAL_THRESHOLD:
Amount requiring approval (default: 50)

DEBUG:
Enable debug mode (never true in production)