Rikugan - Architecture and Design Document v1

Table of Contents

1. Introduction and Goals

Rikugan is a gamified project management web application combining Kanban functionality with bounty-based task rewards for software development teams.

1.1. Requirements Overview

Core Features:

- Role-based management: Goons (task workers), Hashira (task creators), Oyakatasama (admins)
- Bounty system with monetary task rewards and deadline penalties
- Kanban board interface with drag-and-drop
- Real-time notifications and license-based access control

1.2. Quality Goals

Priority	Goal	Target
1	Usability	Intuitive interface, <10min learning curve
2	Security	JWT auth, RBAC, data protection
3	Performance	<500ms API response, 50 concurrent users
4	Maintainability	Modular architecture, 70% test coverage
5	Scalability	Support 200 users, 1000 tasks

---

2. Architecture Constraints

Technical: React 18+, Node.js, MySQL 8.0+, Docker, web-based access
Organizational: 3-4 student developers, one semester timeline, Git version control, 70% test coverage
Conventions: RESTful API, arc42 docs, ESLint standards, snake_case (DB), camelCase (code), PascalCase (components)

---

3. System Scope and Context

Users:

- **Goons**: Browse/complete tasks, earn bounties
- **Hashira**: Create tasks, manage teams, all Goon functions
- **Oyakatasama**: Full system admin, user/license management

Technical Components:

- **Frontend**: React 18+ with HeroUI (browser-based UI)
- **Backend**: Node.js/Express.js (RESTful API, auth, business logic)
- **Database**: MySQL 8.0+ (users, tasks, bounties, notifications, licenses)
- **Deployment**: Docker containerization

---

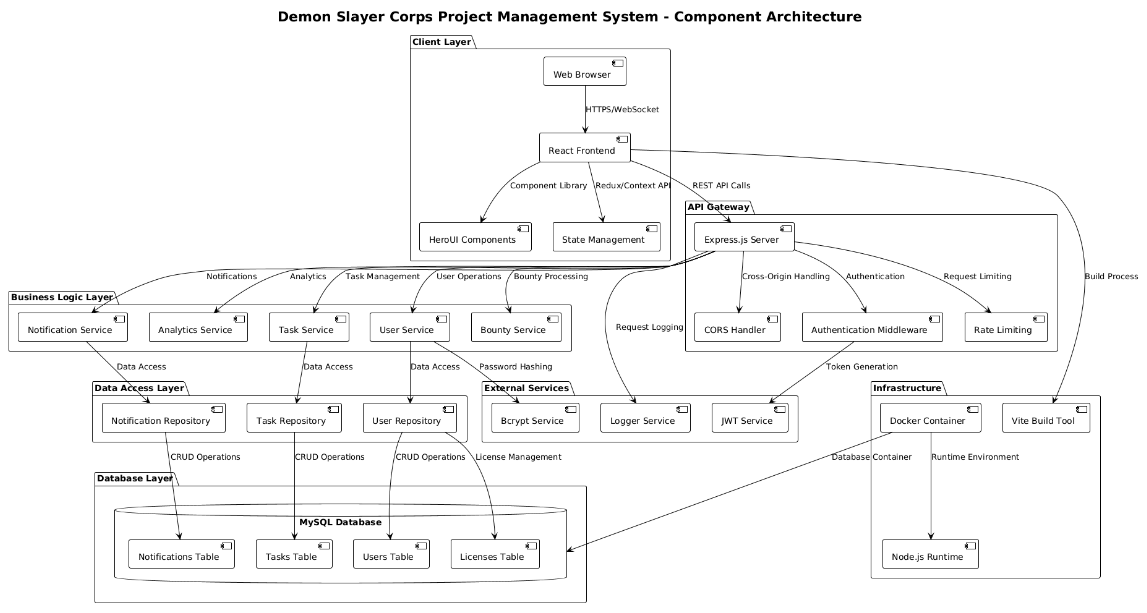
4. Solution Strategy

Architecture: Role-based hierarchy (Goons → Hashira → Oyakatasama) + bounty incentive system
Stack: React 18+ (HeroUI), Node.js/Express.js (MVC), MySQL (normalized schema), Docker
Security: JWT authentication, RBAC at API/component levels
Key Decisions: Bounty-first design, license-controlled access, API-first development, audit trails

---

5. Building Block View

5.1. System Architecture

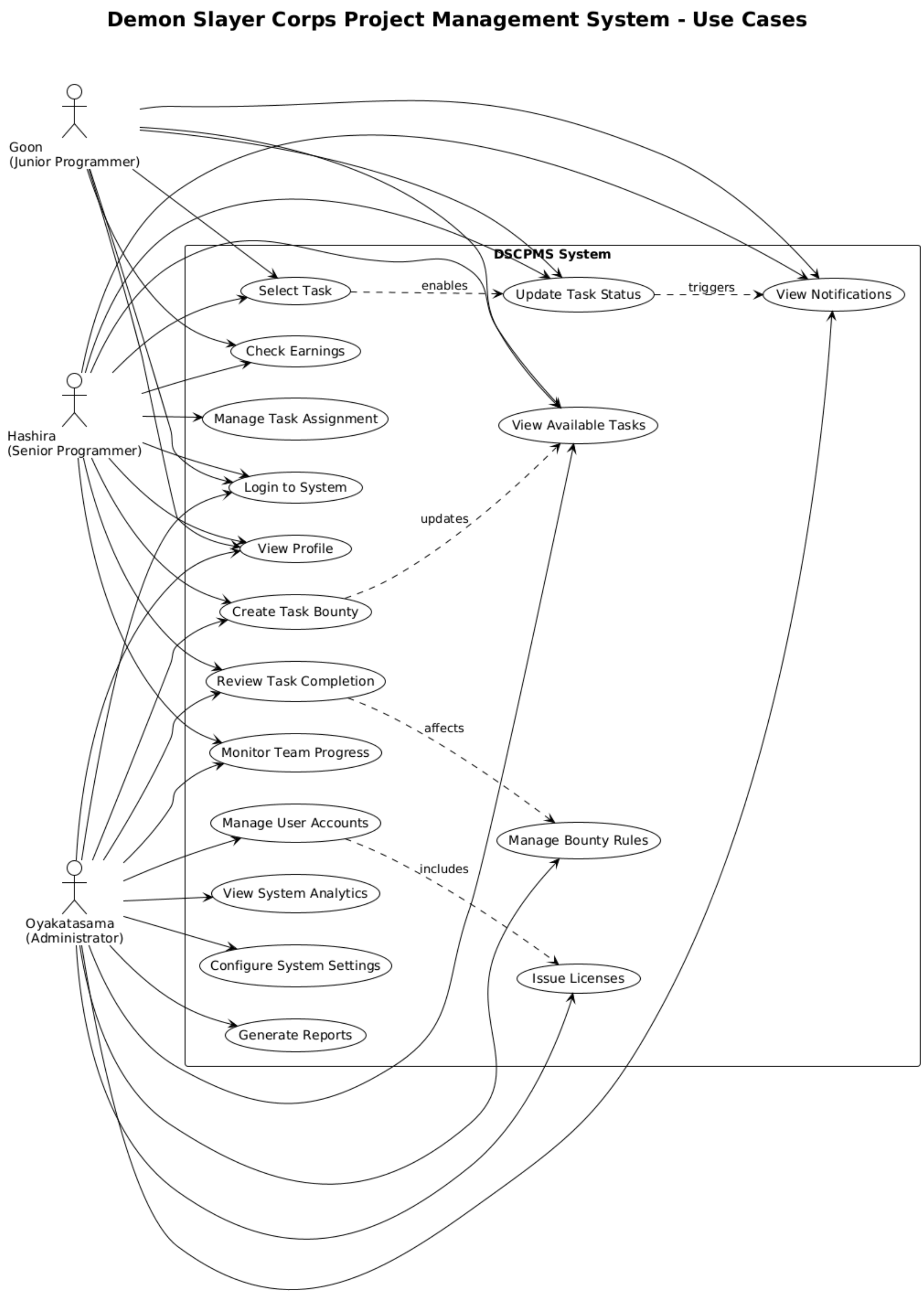


Client Layer: React UI with HeroUI components, state management

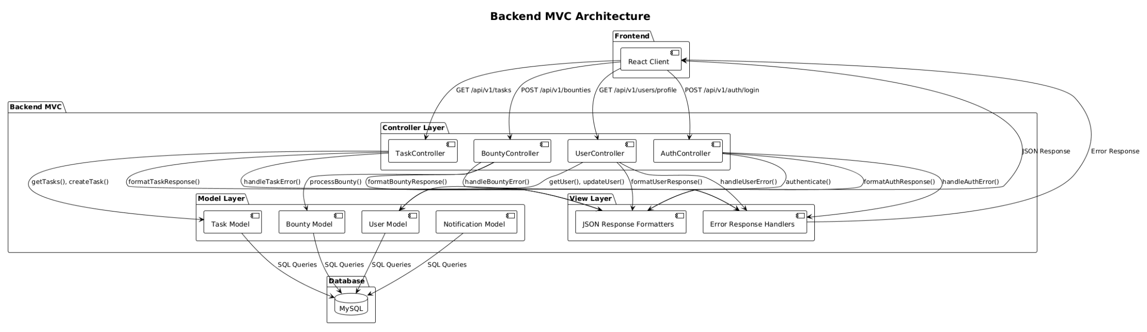
API Gateway: Express.js server, JWT auth, routing, validation

Database: MySQL with normalized schema, connection pooling

5.2. Use Case View



5.3. Backend MVC Architecture

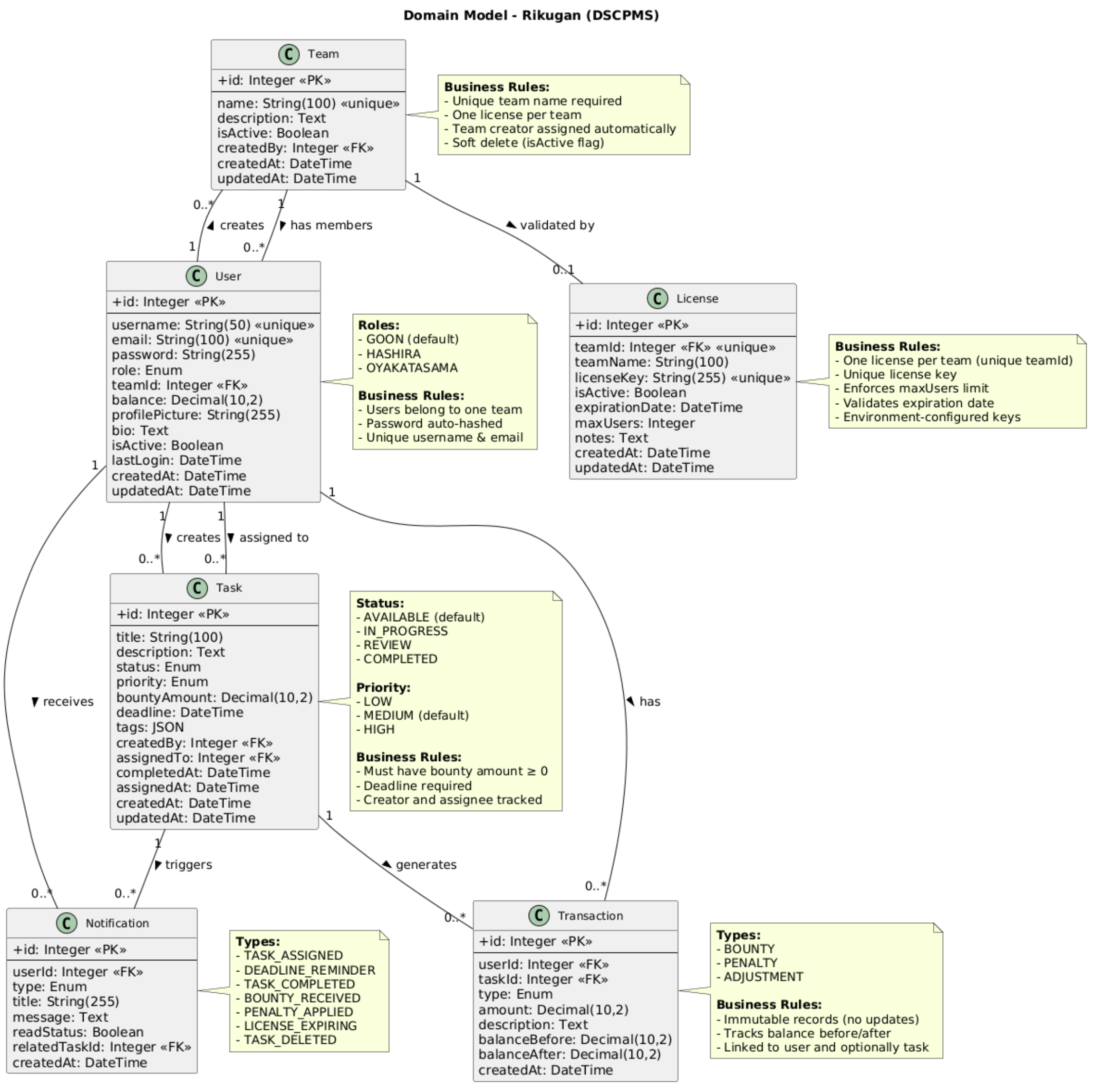


Model: Data entities, business logic, database interactions

Controller: HTTP request handling, route management

View: JSON response formatting

5.4. Domain Model



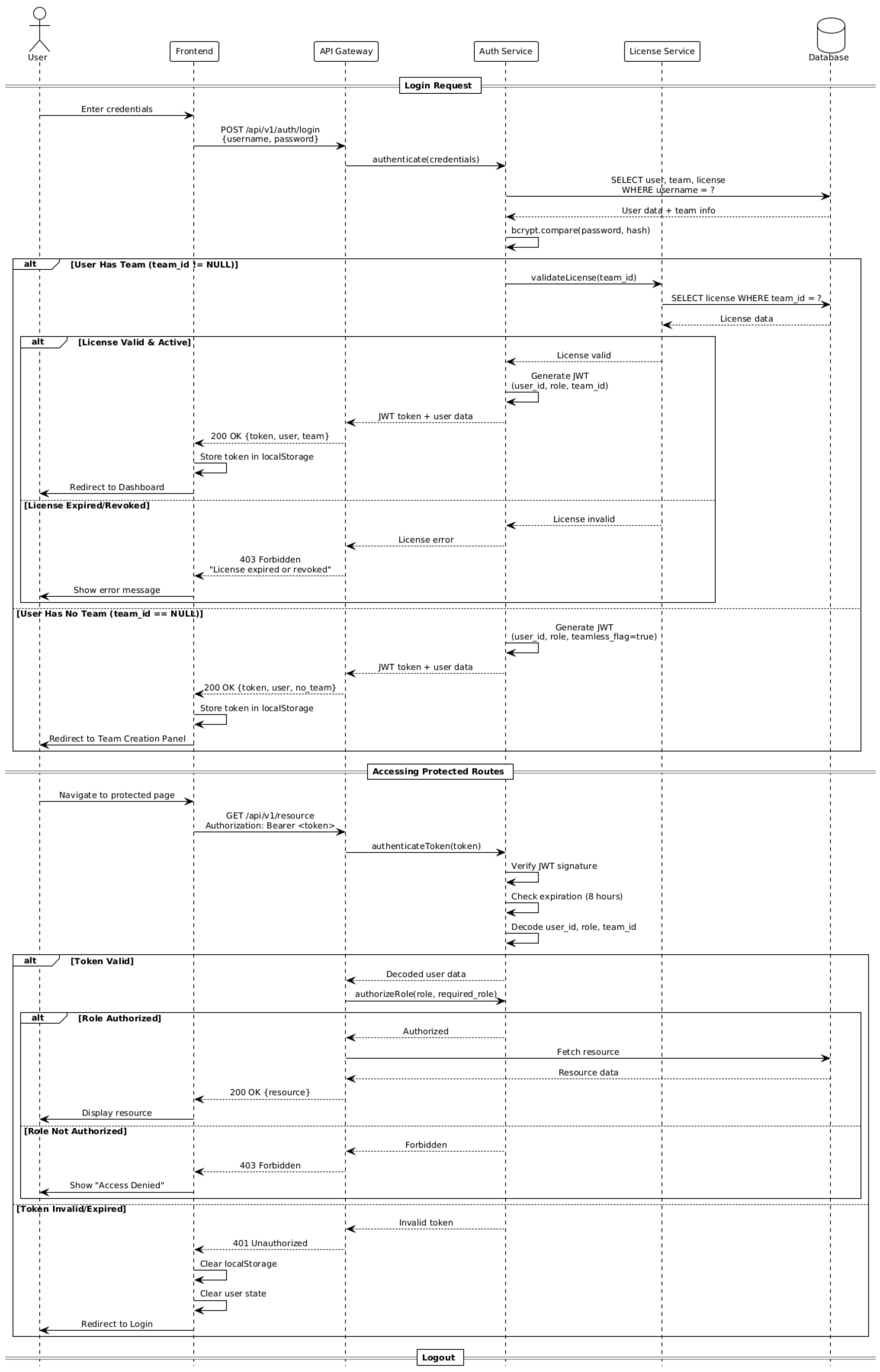
5.5. Backend Modules

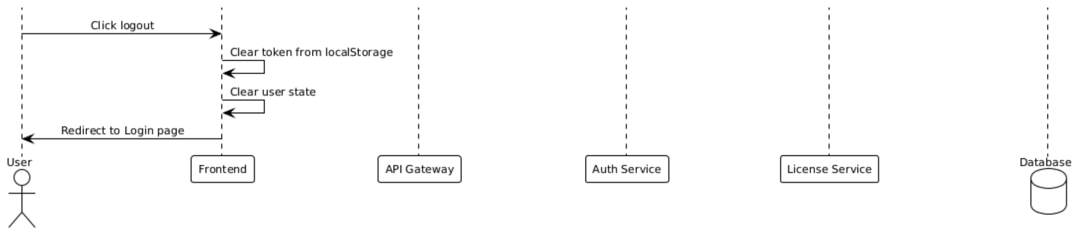
Module	Responsibilities
auth	JWT authentication, login/logout, token management
users	User CRUD, profiles, team membership, earnings
tasks	Task lifecycle, assignment, status tracking, Kanban
bounties	Payment processing, balance management, penalties
notifications	Event notifications, delivery, preferences
licenses	License validation, team access control
teams	Team management, member ops, statistics

---

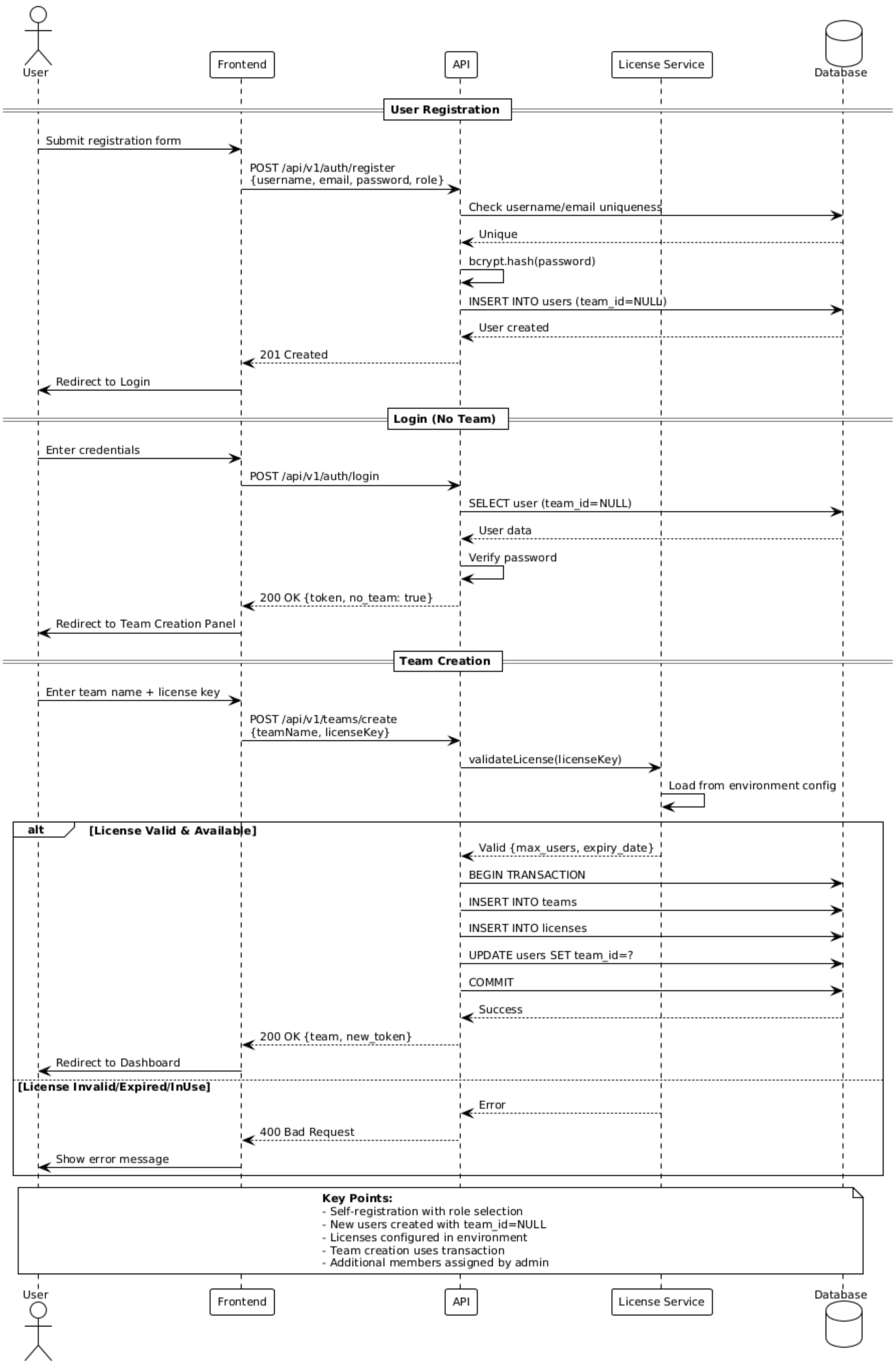
6. Runtime View

6.1. User Authentication Flow

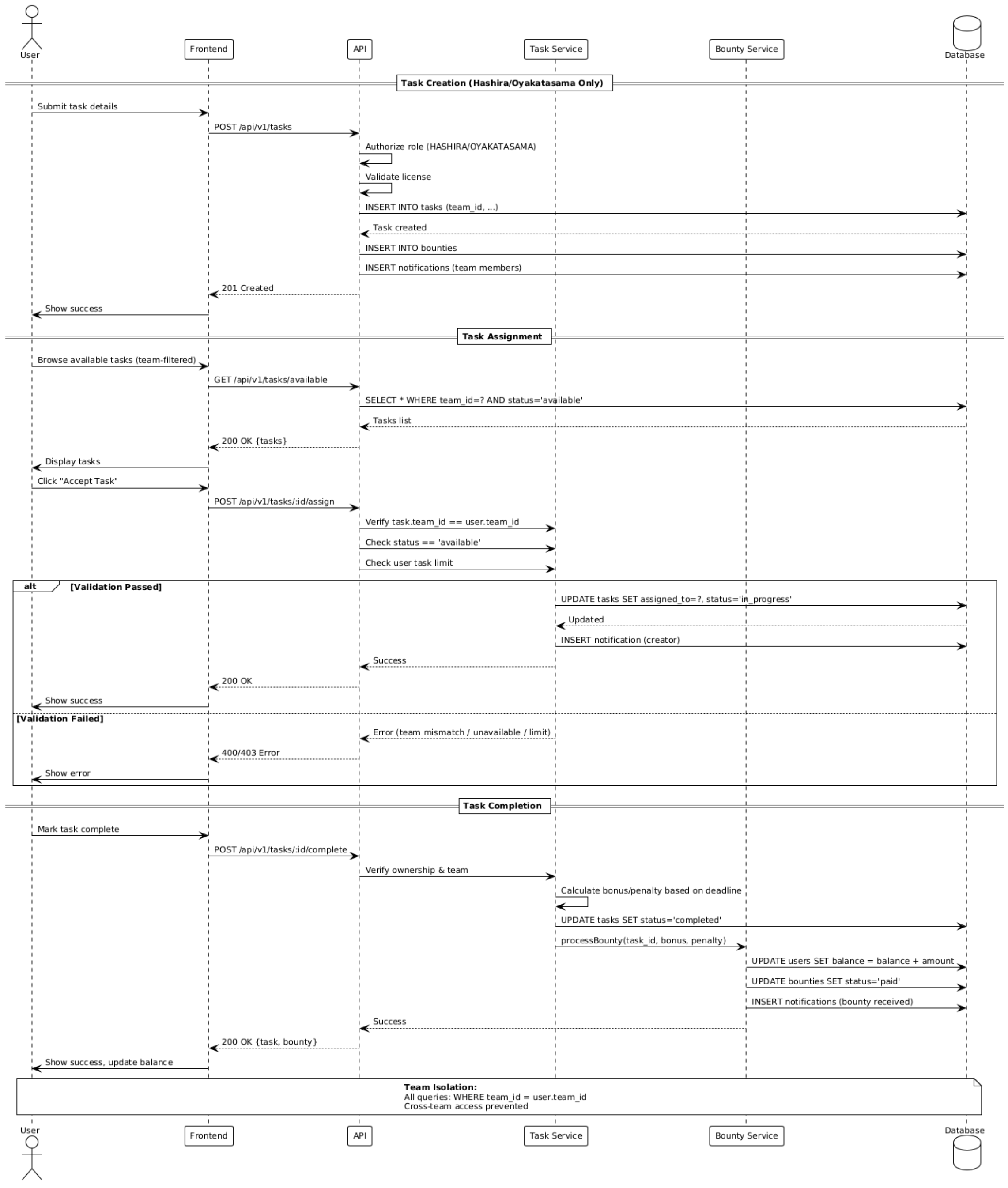

User
Frontend
API Gateway
Auth Service
License Service
Database
Login Request
Enter credentials
POST /api/v1/auth/login
{username, password}
authenticate(credentials)
SELECT user, team, license
WHERE username = ?
User data + team info
bcrypt.compare(password, hash)
alt
[User Has Team (team_id != NULL)]
validateLicense(team_id)
SELECT license WHERE team_id = ?
License data
alt
[License Valid & Active]
License valid
Generate JWT
(user_id, role, team_id)
JWT token + user data
200 OK {token, user, team}
Store token in localStorage
Redirect to Dashboard
[License Expired/Revoked]
License invalid
License error
403 Forbidden
"License expired or revoked"
Show error message
[User Has No Team (team_id == NULL)]
Generate JWT
(user_id, role, teamless_flag=true)
JWT token + user data
200 OK {token, user, no_team}
Store token in localStorage
Redirect to Team Creation Panel
Accessing Protected Routes
Navigate to protected page
GET /api/v1/resource
Authorization: Bearer <token>
authenticateToken(token)
Verify JWT signature
Check expiration (8 hours)
Decode user_id, role, team_id
alt
[Token Valid]
Decoded user data
authorizeRole(role, required_role)
alt
[Role Authorized]
Authorized
Fetch resource
Resource data
200 OK {resource}
Display resource
[Role Not Authorized]
Forbidden
403 Forbidden
Show "Access Denied"
[Token Invalid/Expired]
Invalid token
401 Unauthorized
Clear localStorage
Clear user state
Redirect to Login
Logout



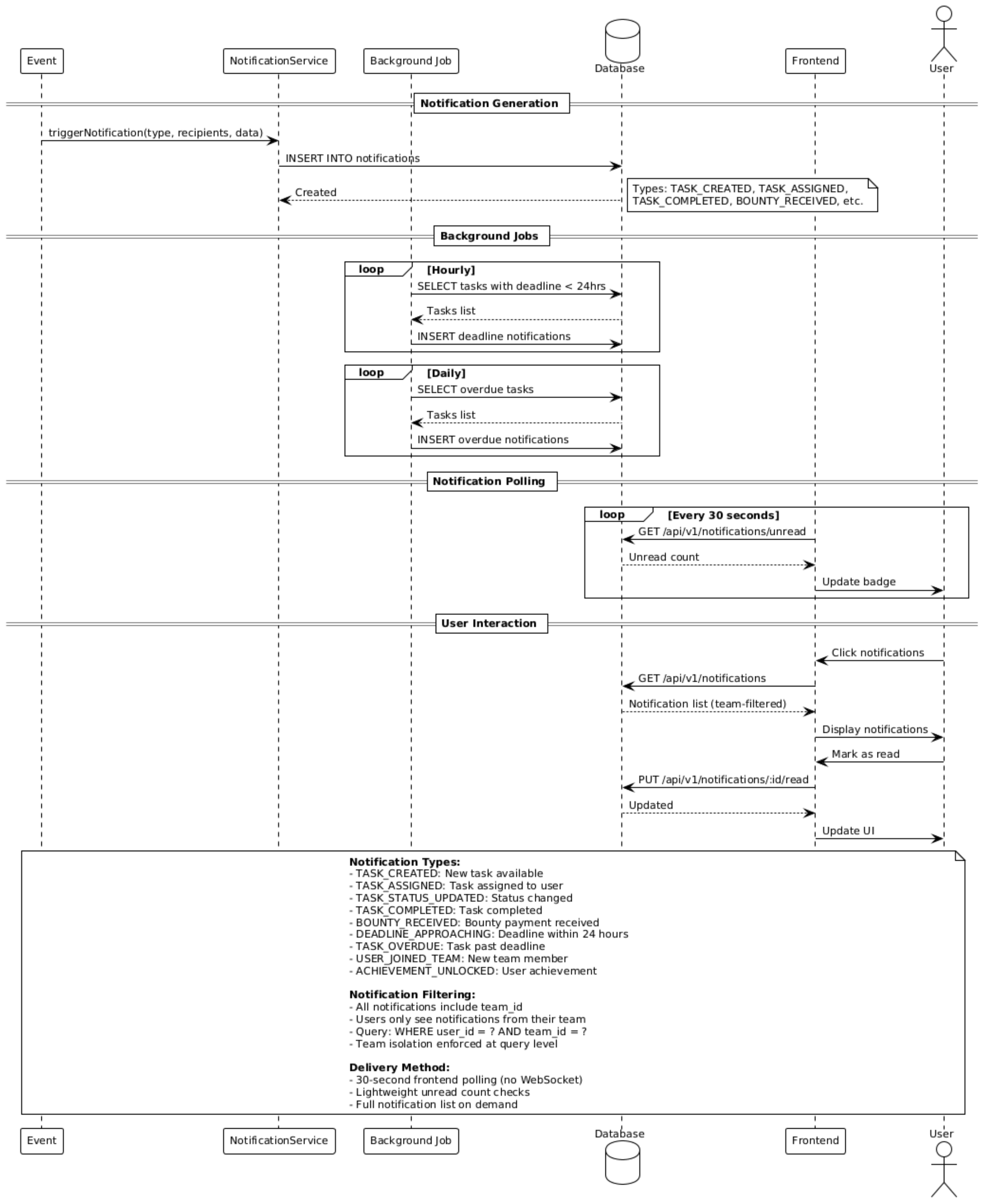
6.2. User Registration and Team Creation Flow


User
Frontend
API
License Service
Database
User Registration
Submit registration form
POST /api/v1/auth/register {username, email, password, role}
Check username/email uniqueness
Unique
bcrypt.hash(password)
INSERT INTO users (team_id=NULL)
User created
201 Created
Redirect to Login
Login (No Team)
Enter credentials
POST /api/v1/auth/login
SELECT user (team_id=NULL)
User data
Verify password
200 OK {token, no_team: true}
Redirect to Team Creation Panel
Team Creation
Enter team name + license key
POST /api/v1/teams/create {teamName, licenseKey}
validateLicense(licenseKey)
Load from environment config
alt
[License Valid & Available]
Valid {max_users, expiry_date}
BEGIN TRANSACTION
INSERT INTO teams
INSERT INTO licenses
UPDATE users SET team_id=?
COMMIT
Success
200 OK {team, new_token}
Redirect to Dashboard
[License Invalid/Expired/InUse]
Error
400 Bad Request
Show error message
Key Points:
- Self-registration with role selection
- New users created with team_id=NULL
- Licenses configured in environment
- Team creation uses transaction
- Additional members assigned by admin

6.3. Task Assignment and Completion Flow


User
Frontend
API
Task Service
Bounty Service
Database
Task Creation (Hashira/Oyakatasama Only)
Submit task details
POST /api/v1/tasks
Authorize role (HASHIRA/OYAKATASAMA)
Validate license
INSERT INTO tasks (team_id, ...)
Task created
INSERT INTO bounties
INSERT notifications (team members)
201 Created
Show success
Task Assignment
Browse available tasks (team-filtered)
GET /api/v1/tasks/available
SELECT * WHERE team_id=? AND status='available'
Tasks list
200 OK {tasks}
Display tasks
Click "Accept Task"
POST /api/v1/tasks/:id/assign
Verify task.team_id == user.team_id
Check status == 'available'
Check user task limit
alt
[Validation Passed]
UPDATE tasks SET assigned_to=?, status='in_progress'
Updated
INSERT notification (creator)
Success
200 OK
Show success
[Validation Failed]
Error (team mismatch / unavailable / limit)
400/403 Error
Show error
Task Completion
Mark task complete
POST /api/v1/tasks/:id/complete
Verify ownership & team
Calculate bonus/penalty based on deadline
UPDATE tasks SET status='completed'
processBounty(task_id, bonus, penalty)
UPDATE users SET balance = balance + amount
UPDATE bounties SET status='paid'
INSERT notifications (bounty received)
Success
200 OK {task, bounty}
Show success, update balance
Team Isolation:
All queries: WHERE team_id = user.team_id
Cross-team access prevented

6.4. Notification System Flow

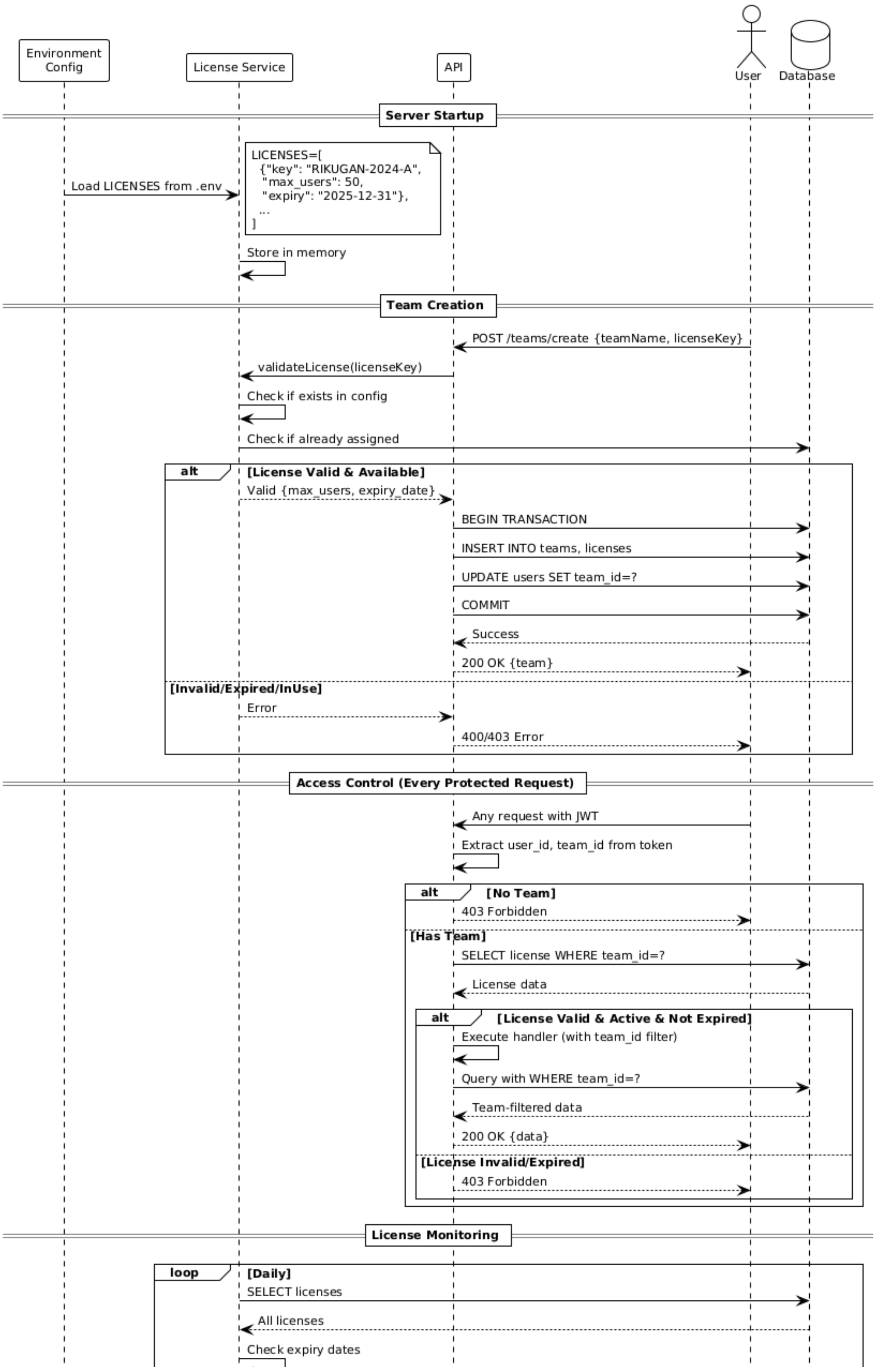

Event
NotificationService
Background Job
Database
Frontend
User
Notification Generation
triggerNotification(type, recipients, data)
INSERT INTO notifications
Created
Types: TASK_CREATED, TASK_ASSIGNED, TASK_COMPLETED, BOUNTY_RECEIVED, etc.
Background Jobs
loop [Hourly]
SELECT tasks with deadline < 24hrs
Tasks list
INSERT deadline notifications
loop [Daily]
SELECT overdue tasks
Tasks list
INSERT overdue notifications
Notification Polling
loop [Every 30 seconds]
GET /api/v1/notifications/unread
Unread count
Update badge
User Interaction
Click notifications
GET /api/v1/notifications
Notification list (team-filtered)
Display notifications
Mark as read
PUT /api/v1/notifications/:id/read
Updated
Update UI

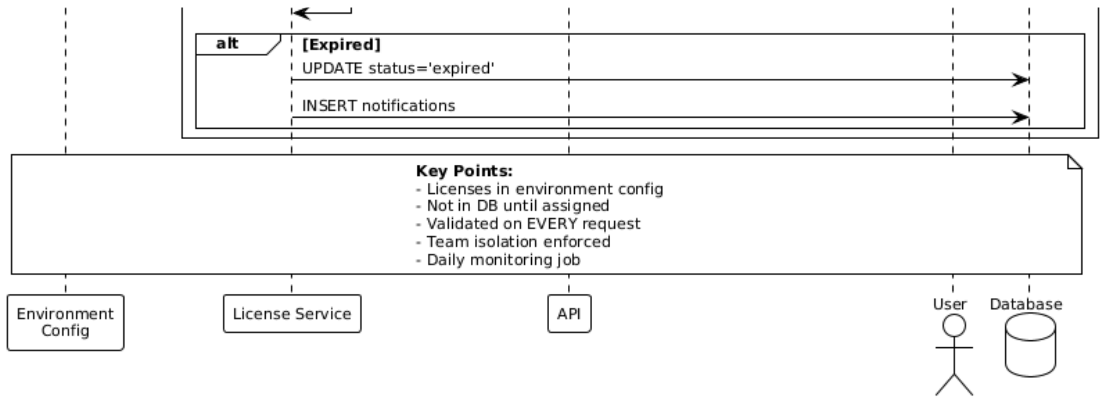
Notification Types:
- TASK_CREATED: New task available
- TASK_ASSIGNED: Task assigned to user
- TASK_STATUS_UPDATED: Status changed
- TASK_COMPLETED: Task completed
- BOUNTY_RECEIVED: Bounty payment received
- DEADLINE_APPROACHING: Deadline within 24 hours
- TASK_OVERDUE: Task past deadline
- USER_JOINED_TEAM: New team member
- ACHIEVEMENT_UNLOCKED: User achievement

Notification Filtering:
- All notifications include team_id
- Users only see notifications from their team
- Query: WHERE user_id = ? AND team_id = ?
- Team isolation enforced at query level

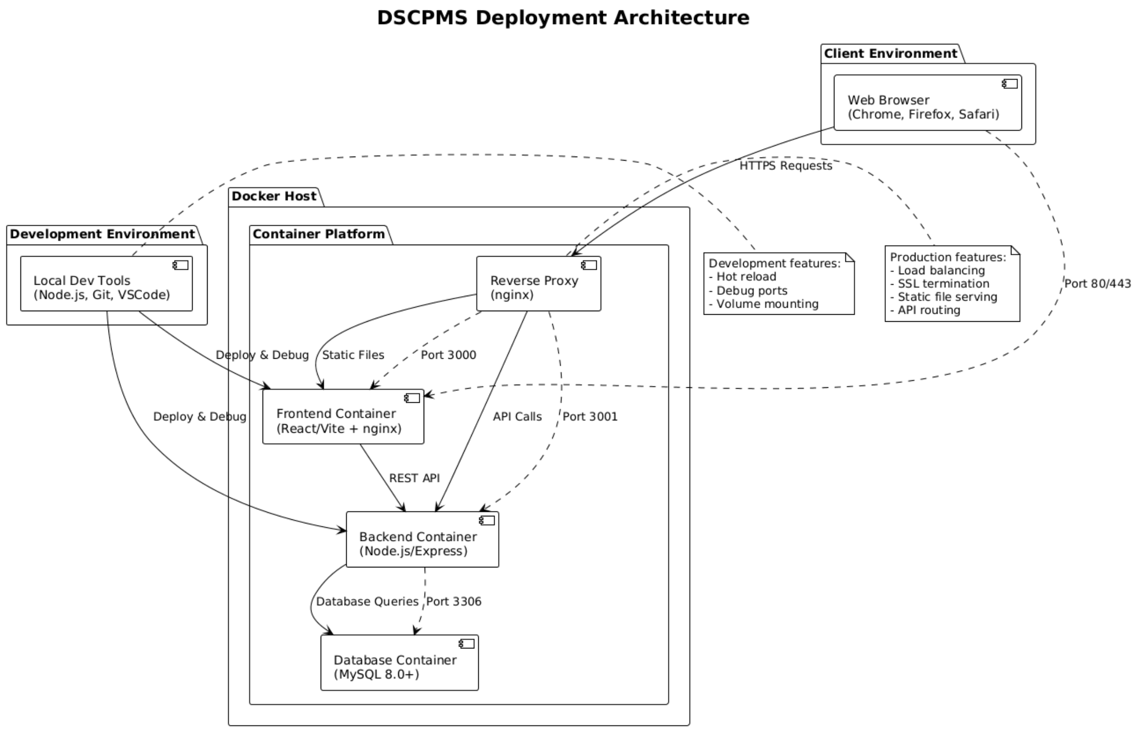
Delivery Method:
- 30-second frontend polling (no WebSocket)
- Lightweight unread count checks
- Full notification list on demand

6.5. License Validation Flow


Environment Config
License Service
API
User
Database
Server Startup
Load LICENSES from .env
LICENSES=[
  {"key": "RIKUGAN-2024-A",
   "max_users": 50,
   "expiry": "2025-12-31"},
  ...
]
Store in memory
Team Creation
POST /teams/create {teamName, licenseKey}
validateLicense(licenseKey)
Check if exists in config
Check if already assigned
alt
[License Valid & Available]
Valid {max_users, expiry_date}
BEGIN TRANSACTION
INSERT INTO teams, licenses
UPDATE users SET team_id=?
COMMIT
Success
200 OK {team}
[Invalid/Expired/InUse]
Error
400/403 Error
Access Control (Every Protected Request)
Any request with JWT
Extract user_id, team_id from token
alt
[No Team]
403 Forbidden
[Has Team]
SELECT license WHERE team_id=?
License data
alt
[License Valid & Active & Not Expired]
Execute handler (with team_id filter)
Query with WHERE team_id=?
Team-filtered data
200 OK {data}
[License Invalid/Expired]
403 Forbidden
License Monitoring
loop
[Daily]
SELECT licenses
All licenses
Check expiry dates



7. Deployment View



Components:

- **Frontend Container**: React 18+/Vite/nginx (production build)
- **Backend Container**: Node.js 18+/Express.js (API server)
- **Database Container**: MySQL 8.0+ (persistent storage)
- **Reverse Proxy**: nginx (load balancing, SSL)

Requirements: Docker 20.10+, Docker Compose 2.0+, 4GB RAM, 20GB disk

8. Concepts

8.1. Domain Models

Core Entities:

- **Team**: id, name, isActive, createdBy | One-to-many with Users, one-to-one with License
- **User**: id, username, email, password, role (GOON/HASHIRA/OYAKATASAMA), teamId, balance | Many-to-one with Team
- **Task**: id, title, description, status, priority, bountyAmount, deadline, createdBy, assignedTo | Status: AVAILABLE → IN_PROGRESS → REVIEW → COMPLETED
- **License**: id, teamId (unique), licenseKey (unique), isActive, expirationDate, maxUsers | One-to-one with Team
- **Notification**: id, userId, type, message, readStatus, relatedTaskId | One-to-many with User
- **Transaction**: id, userId, taskId, type (BOUNTY/PENALTY/ADJUSTMENT), amount, balanceBefore, balanceAfter | Immutable

Key Constraints:

- Unique usernames/emails, one license per team, bounty ≥ 0, one user per task, passwords bcrypt-hashed

8.2. Persistency

MySQL 8.0+ with normalized schema. Core tables: `teams` , `users` (teamId FK), `tasks` (teamId, createdBy, assignedTo FKs), `licenses` (teamId unique FK), `notifications` , `transactions` . Connection pooling, prepared statements, indexed on user_id/task_status/deadlines, automated backups.

8.3. User Interface

React 18+ with HeroUI components, responsive (mobile-first), role-based UI rendering, WCAG 2.1 Level A, lazy loading, dark/light mode.

8.4. Security

JWT auth (8hr expiration), bcrypt passwords, RBAC (API + component level), input validation/sanitization, SQL injection prevention, XSS protection, HTTPS enforcement, Helmet security headers.

8.5. Session Handling

Stateless JWT tokens with user ID, role, permissions. 8hr expiration, refresh token mechanism, auto-logout.

8.6. Error Handling

React error boundaries, global API error handler, structured error responses with codes (VALIDATION_ERROR, INTERNAL_ERROR), user-friendly messages.

8.7. Logging and Monitoring

Structured JSON logs (timestamp, level, service, userId, action), different log levels per environment, request/response logging, security audit trails.

8.8. Configuration

Environment-based config (server port/host, database connection, JWT secrets, bounty rules), separate dev/prod settings via environment variables.

9. Design Decisions

Technology Stack: React/Node.js/MySQL chosen for full JavaScript stack, excellent documentation, learning value, and rapid development.

Database: 3NF with strategic denormalization for performance. Foreign key constraints ensure integrity, separate bounties/transactions table for audit trail.

Authentication: JWT-based (stateless, scalable, mobile-ready, role/permissions in token). 8hr expiration, refresh tokens for UX.

---

10. Quality Scenarios

10.1. Quality Tree

10. Quality Scenarios

10.1. Quality Tree

```
Rikugan Quality Goals
├── Usability
│   ├── Learning Curve (High Priority)
│   │   └── New users can complete basic tasks within 10 minutes
│   └── Interface Consistency (Medium Priority)
│       └── All pages follow consistent navigation patterns
├── Performance
│   ├── Response Time (High Priority)
│   │   └── API calls respond within 500ms for 95% of requests
│   └── Concurrent Users (Medium Priority)
│       └── Support 50 simultaneous users without degradation
├── Security
│   ├── Authentication (High Priority)
│   │   └── Secure login with role-based access control
│   └── Data Protection (High Priority)
│       └── All user data encrypted and validated
└── Maintainability
    ├── Code Quality (Medium Priority)
    │   └── 70% test coverage with clean architecture
    └── Documentation (Medium Priority)
        └── Complete API and component documentation
```

10.2. Evaluation Scenarios

Usability - New User Onboarding *Scenario*: A new Goon user logs in for the first time and wants to select and complete their first task. *Measurement*: Time from login to task selection should be under 5 minutes without training. *Architecture Support*: Clear dashboard design with visual task cards and intuitive status progression.

Performance - Concurrent Task Updates *Scenario*: 20 users simultaneously update task statuses during peak usage. *Measurement*: All updates complete within 2 seconds with no data conflicts. *Architecture Support*: Database connection pooling and optimistic locking prevent performance bottlenecks.

Security - Role Privilege Escalation *Scenario*: A Goon user attempts to access Hashira-only functions through direct API calls. *Measurement*: All unauthorized attempts are blocked and logged. *Architecture Support*: Multi-layer authorization checks at API middleware and service levels.

Maintainability - Feature Addition *Scenario*: Adding a new task filter feature requires changes across frontend and backend. *Measurement*: Implementation completed in under 4 hours by a new team member. *Architecture Support*: Modular component structure and clear API patterns enable quick feature addition.

---

11. Technical Risks

Risk	Probability	Impact	Mitigation Strategy
Database Performance Degradation	Medium	High	Implement query optimization, indexing, and connection pooling. Monitor query performance in development.
JWT Token Security Vulnerabilities	Low	High	Use strong secrets, implement proper token expiration, and regular security audits.
React Component State Management Complexity	High	Medium	Use established patterns (Context API, custom hooks) and maintain clear data flow.
API Rate Limiting Bypass	Medium	Medium	Implement multiple layers of rate limiting and input validation.
Database Schema Changes Breaking Compatibility	Medium	High	Use database migration scripts and maintain backward compatibility during transitions.
Third-party Library Vulnerabilities	Medium	Medium	Regular dependency updates, security scanning, and minimal external dependencies.

Risk Monitoring:

```
- Weekly security scans of dependencies
- Performance monitoring in development environment
- Code review process for all changes
- Automated testing to catch regression issues
│   │   └── API calls respond within 500ms for 95% of requests
│   └── Concurrent Users (Medium Priority)
│       └── Support 50 simultaneous users without degradation
├── Security
│   ├── Authentication (High Priority)
│   │   └── Secure login with role-based access control
│   └── Data Protection (High Priority)
│       └── All user data encrypted and validated
└── Maintainability
    ├── Code Quality (Medium Priority)
    │   └── 70% test coverage with clean architecture
    └── Documentation (Medium Priority)
        └── Complete API and component documentation
```

10.2. Evaluation Scenarios

Usability - New User Onboarding *Scenario*: A new Goon user logs in for the first time and wants to select and complete their first task. *Measurement*: Time from login to task selection should be under 5 minutes without training. *Architecture Support*: Clear dashboard design with visual task cards and intuitive status progression.

Performance - Concurrent Task Updates *Scenario*: 20 users simultaneously update task statuses during peak usage. *Measurement*: All updates complete within 2 seconds with no data conflicts. *Architecture Support*: Database connection pooling and optimistic locking prevent performance bottlenecks.

Security - Role Privilege Escalation *Scenario*: A Goon user attempts to access Hashira-only functions through direct API calls. *Measurement*: All unauthorized attempts are blocked and logged. *Architecture Support*: Multi-layer authorization checks at API middleware and service levels.

Maintainability - Feature Addition *Scenario*: Adding a new task filter feature requires changes across frontend and backend. *Measurement*: Implementation completed in under 4 hours by a new team member. *Architecture Support*: Modular component structure and clear API patterns enable quick feature addition.

---

11. Technical Risks

Risk	Probability	Impact	Mitigation Strategy
Database Performance Degradation	Medium	High	Implement query optimization, indexing, and connection pooling. Monitor query performance in development.
JWT Token Security Vulnerabilities	Low	High	Use strong secrets, implement proper token expiration, and regular security audits.
React Component State Management Complexity	High	Medium	Use established patterns (Context API, custom hooks) and maintain clear data flow.
API Rate Limiting Bypass	Medium	Medium	Implement multiple layers of rate limiting and input validation.
Database Schema Changes Breaking Compatibility	Medium	High	Use database migration scripts and maintain backward compatibility during transitions.
Third-party Library Vulnerabilities	Medium	Medium	Regular dependency updates, security scanning, and minimal external dependencies.

Risk Monitoring:

- Weekly security scans of dependencies
- Performance monitoring in development environment
- Code review process for all changes
- Automated testing to catch regression issues

---

12. Glossary

Bounty: Monetary task reward | **Goon**: Junior programmer role | **Hashira**: Senior programmer role | **Oyakatasama**: Administrator role | **JWT**: JSON Web Token authentication | **Kanban**: Visual project management | **RBAC**: Role-Based Access Control | **REST**: Representational State Transfer API | **Docker**: Containerization platform | **Express.js**: Node.js web framework | **HeroUI**: React component library | **MySQL**: Relational database | **Vite**: Build tool | **WCAG**: Web accessibility standards | **XSS**: Cross-Site Scripting attack

---

Document: Architecture design per arc42 template | **Version:** 1.0 | **Updated:** December 2024