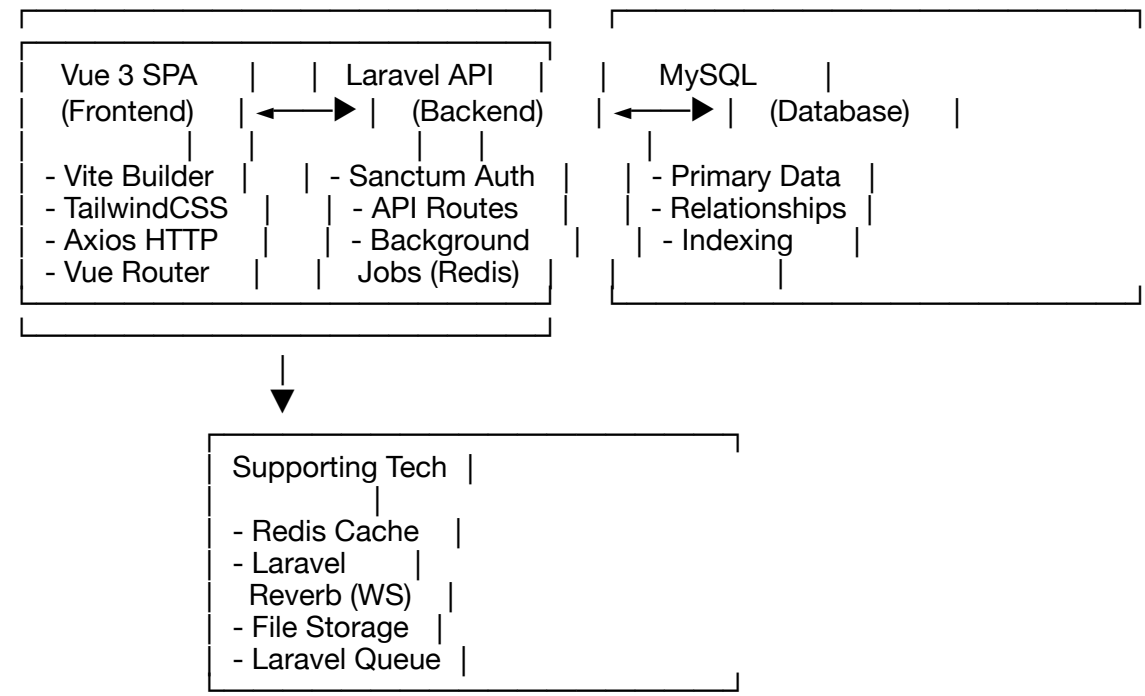
Eco Track System design

EcoTrack: System Design Document

1. System Architecture Overview

1.1 High-Level Architecture



1.2 Component Breakdown

Frontend Layer (Vue 3 SPA)
- **Authentication Module**: Login/Register/Password Reset
- **Dashboard Module**: Role-specific dashboards (User/Moderator/Admin)
- **Emissions Module**: Carbon footprint logging and analytics
- **Social Module**: Following/Followers, User profiles
- **Blog Module**: Post creation, feed, reactions
- **Notifications Module**: Real-time notification system
- **Administration Module**: User management, content moderation

Backend Layer (Laravel API)
- **Authentication Service**: Sanctum-based API authentication
- **User Management Service**: User roles, permissions, profiles
- **Emissions Service**: Carbon footprint calculations and storage
- **Social Service**: Following relationships, user interactions
- **Content Service**: Blog posts, reactions, content moderation
- **Notification Service**: Real-time notifications via Reverb
- **Analytics Service**: Data aggregation and insights
- **File Service**: Image uploads and processing

Data Layer (MySQL)
- **Core Tables**: Users, Posts, Emissions, Reactions
- **Relationship Tables**: Follows, Reports, Notifications
- **System Tables**: Announcements, Settings, Logs

1.3 Technology Stack Integration

Frontend Stack
- **Vue 3**: Component-based UI framework with Composition API
- **Vue Router**: Client-side routing for SPA navigation
- **Pinia**: State management for application data
- **Vite**: Build tool and development server
- **TailwindCSS 4**: Utility-first CSS framework
- **Axios**: HTTP client for API communication
- **Chart.js/ApexCharts**: Data visualization libraries

Backend Stack
- **Laravel 12**: PHP framework for API development
- **Sanctum**: API authentication system
- **Reverb**: WebSocket server for real-time features
- **Intervention Image**: Image processing and manipulation
- **Laravel Queue**: Background job processing
- **Redis**: Caching and session storage

Infrastructure
- **Docker**: Containerization for development and deployment
- **MySQL 8.0**: Primary database with optimized indexing
- **Redis**: Cache layer and queue backend
- **File Storage**: Local/S3-compatible storage for uploads

2. Data Flow Architecture

2.1 Authentication Flow
```
User Input → Vue Form → Form Validation → Axios Request → Laravel Auth Controller
    ↓
Sanctum Token Generation → Redis Token Storage → JSON Response → Vue Store Update
    ↓
Automatic Header Injection → Protected Route Access → User Dashboard Render
```

2.2 Real-time Notification Flow
```
Event Trigger (Follow/React/Report) → Laravel Event System → Queue Job Dispatch
    ↓
Background Processing → Notification Creation → Reverb WebSocket Broadcast
    ↓
Vue WebSocket Listener → Store Update → UI Component Refresh → User Notification
```

2.3 Content Creation Flow
```
Vue Form Submission → File Upload (if any) → Laravel Validation Layer
    ↓
Image Processing Job → Database Transaction → Cache Invalidation
    ↓
Real-time Broadcast → Feed Update → Analytics Update
```

2.4 Social Interaction Flow
```
User Action (Follow/React) → API Request → Permission Check → Database Update
    ↓
Event Broadcasting → Notification Generation → Real-time UI Update
    ↓
```

Analytics Tracking → Eco Score Calculation → Achievement Check
```

## 3. System Components Architecture

### 3.1 Frontend Architecture

#### Component Hierarchy
```
App.vue
├── Router View
│   ├── Auth Layout
│   │   ├── Login Component
│   │   ├── Register Component
│   │   └── Password Reset
│   ├── Dashboard Layout
│   │   ├── Navigation Bar
│   │   ├── Sidebar Menu
│   │   ├── Main Content Area
│   │   └── Notification Center
│   └── Admin Layout
│   ├── Admin Navigation
│   ├── Management Panels
│   └── System Analytics
```

#### State Management Structure
```
Pinia Stores:
├── auth.js (User authentication state)
├── posts.js (Blog posts and feed data)
├── notifications.js (Real-time notifications)
├── emissions.js (Carbon footprint data)
├── social.js (Following/followers data)
└── admin.js (Administrative data)
```

#### Routing Structure
```
Routes:
├── / (Landing page)
├── /auth/* (Authentication routes)
├── /dashboard (User dashboard)
├── /profile/:id (User profiles)
├── /posts (Blog feed)
├── /emissions (Emission tracking)
├── /moderator/* (Moderator panel)
└── /admin/* (Admin panel)
```
```

3.2 Backend Architecture

Service Layer Organization
```
Services:
├── AuthService (User authentication logic)
├── EmissionCalculatorService (Carbon footprint calculations)
├── NotificationService (Notification management)
├── SocialService (Following/follower logic)
├── ModerationService (Content moderation)
├── AnalyticsService (Data analysis and reporting)
├── FileUploadService (Image processing and storage)
└── PermissionService (Role-based access control)
```

Event-Driven Architecture
```
Events & Listeners:
├── UserRegistered → SendWelcomeNotification
├── UserFollowed → SendFollowNotification
├── PostCreated → BroadcastPostUpdate
├── PostReacted → SendReactionNotification
├── UserReported → NotifyModerators
├── EmissionLogged → UpdateEcoScore
└── AchievementUnlocked → SendAchievementNotification
```

4. Security Architecture

4.1 Authentication & Authorization

API Security Layers
1. **Rate Limiting**: API throttling per user/IP
2. **Sanctum Tokens**: Stateless API authentication
3. **Role-Based Access**: User/Moderator/Admin permissions
4. **Request Validation**: Input sanitization and validation
5. **CORS Protection**: Cross-origin request filtering

Permission Matrix
```
Actions                | User | Moderator | Admin
-----------------------|------|-----------|-------
View Posts             | ✓    | ✓         | ✓
Create Posts           | ✓    | ✓         | ✓
Delete Own Posts       | ✓    | ✓         | ✓
Delete Any Posts       | ✗    | ✓         | ✓
Ban Users              | ✗    | ✓         | ✓
Delete Users           | ✗    | ✗         | ✓
Manage Announcements   | ✗    | ✓*        | ✓
System Settings        | ✗    | ✗         | ✓
```
*Moderators can create, but need admin approval

4.2 Data Protection

Input Validation
- Server-side validation for all API endpoints
- Client-side validation for immediate feedback
- File upload restrictions and scanning
- SQL injection prevention through Eloquent ORM
- XSS protection through output escaping

Privacy Controls
- User data encryption for sensitive information
- GDPR compliance features (data export/deletion)
- Profile visibility controls
- Content reporting and moderation

5. Performance Architecture

5.1 Caching Strategy

Redis Caching Layers
```
Cache Levels:
├── Session Storage (User sessions)
├── API Response Caching (Frequently accessed data)
├── Database Query Caching (Complex aggregations)
├── File Caching (Processed images)
└── Feed Caching (Personalized user feeds)
```

Cache Invalidation Strategy
- Time-based expiration for static content
- Event-based invalidation for dynamic content
- Cache tagging for selective clearing
- Background cache warming for heavy queries

5.2 Database Optimization

Indexing Strategy
```sql
-- Performance Critical Indexes
INDEX idx_users_role_status (role, status)
INDEX idx_posts_user_created (user_id, created_at)
INDEX idx_emissions_user_date (user_id, activity_date)
INDEX idx_follows_follower_following (follower_id, following_id)
INDEX idx_notifications_user_unread (user_id, read_at)
INDEX idx_reactions_reactable (reactable_id, reactable_type)
```

Query Optimization
- Eager loading for N+1 query prevention
- Database query monitoring and analysis
- Pagination for large datasets
- Aggregation pre-calculation for analytics

5.3 Real-time Performance

WebSocket Optimization

- Connection pooling and management
- Message queuing for offline users
- Selective channel subscriptions
- Automatic reconnection handling

Background Processing
- Queue-based job processing
- Job prioritization and batching
- Failed job handling and retry logic
- Performance monitoring and alerting

6. Scalability Architecture

6.1 Horizontal Scaling Considerations

Load Balancing Strategy
- API server load balancing
- Database read replicas
- CDN for static asset delivery
- Session storage externalization

Microservice Preparation
- Service boundary identification
- API contract definitions
- Data consistency strategies
- Inter-service communication patterns

6.2 Growth Planning

Data Growth Management
- Database partitioning strategies
- Archive data management
- Storage tier optimization
- Backup and recovery planning

Feature Scalability
- Plugin architecture for new features
- API versioning strategy
- Feature flag implementation
- A/B testing framework

7. Monitoring & Analytics

7.1 System Monitoring

Performance Metrics
- API response times
- Database query performance
- WebSocket connection health
- Queue job processing times
- Error rates and patterns

Business Metrics
- User engagement analytics
- Content creation rates
- Social interaction metrics
- Carbon footprint trends
- Feature usage statistics

7.2 Error Handling & Logging

Logging Strategy
```
Log Levels:
├── Error (System failures, exceptions)
├── Warning (Performance issues, deprecated features)
├── Info (User actions, system events)
└── Debug (Detailed application flow)
```

Error Recovery
- Graceful degradation strategies
- Automatic retry mechanisms
- Fallback content delivery
- User notification systems

8. Deployment Architecture

8.1 Container Strategy

Docker Configuration
```
Services:
├── app (Laravel API server)
├── web (Nginx reverse proxy)
├── db (MySQL database)
├── redis (Cache and queue backend)
├── queue (Background job processor)
└── reverb (WebSocket server)
```

Environment Management
- Development, staging, production environments
- Configuration management
- Secret management
- Database migration strategies

8.2 CI/CD Pipeline

Deployment Flow
```
Code Push → Automated Testing → Build Docker Images → Deploy to Staging
    ↓
Staging Tests → Manual Approval → Production Deployment → Health Checks
```

Quality Gates
- Automated unit and integration tests
- Code quality analysis
- Security vulnerability scanning
- Performance regression testing

9. Internationalization Architecture

9.1 Multi-language Support

Frontend Localization
- Vue I18n for component translations
- Dynamic locale switching
- Date and number formatting
- Right-to-left language support

Backend Localization
- Laravel localization files
- Database content translation
- API response localization
- Email template translation

9.2 Regional Considerations

Content Adaptation
- Carbon footprint regional factors
- Cultural sensitivity in content
- Legal compliance variations
- Currency and unit conversions

10. Future Architecture Considerations

10.1 Technology Evolution

Framework Updates
- Laravel upgrade pathways
- Vue ecosystem evolution
- Database technology advancement
- Cloud service integration

10.2 Feature Expansion

Potential Integrations
- Third-party carbon calculators
- Social media platform connections
- IoT device data integration
- Mobile application development
- AI-powered insights and recommendations