ITE 18 – Application Development and Emerging Technologies

Web/Mobile Application - Final Project Submission

Student Name(s): _Ken Darell S. Arado

Project Title: ___*FreePort* (Freelance Recruitment System)

Date of Submission: __

Course/Section: __ITE18 – EJ1

---

1. Project Overview

1.1 Project Description

Freeport is a comprehensive freelancing marketplace platform that connects skilled freelancers with employers seeking professional services. The platform solves the problem of finding and managing freelance talent by providing a centralized hub where freelancers can showcase their skills, portfolios, and availability, while employers can discover, evaluate, and collaborate with qualified professionals. The system facilitates secure authentication, profile management, and seamless matching between service providers and clients.

1.2 Target Users

Freelancers: Independent professionals seeking to showcase their skills, manage their profiles, track availability, and find work opportunities. They need tools to display their portfolios, manage their educational background, set their availability, and connect with potential employers.

Employers: Businesses and organizations looking to hire freelance talent for projects. They need to browse freelancer profiles, bookmark candidates, manage projects, and communicate with potential hires to find the right talent for their specific needs.

1.3 Key Features

- **Feature 1: User Authentication & Role Management** - Secure registration/login system with Laravel Sanctum tokens supporting both freelancer and employer account types
- **Feature 2: Comprehensive Profile Management** - Detailed profiles for freelancers (skills, portfolio, education, availability) and employers (company information, projects)

- **Feature 3: Advanced Search & Discovery** - Browse and filter freelancers by location, skills, availability status with real-time search capabilities
- **Feature 4: Dashboard Analytics** - Personalized dashboards showing relevant statistics, recent activity, and quick access to key features
- **Feature 5: Project & Bookmark Management** - Employers can manage ongoing projects and bookmark favorite freelancers for future collaboration

  Feature 6: Portfolio & Skills Showcase - Freelancers can display work samples, technical skills, educational background, and professional experience
- **Feature 7: Availability Tracking** - Real-time availability status management allowing freelancers to indicate their capacity for new projects

1.4 Technology Stack

Frontend: React 18.3.1 with TypeScript, React Router DOM for navigation, TailwindCSS for styling, Radix UI components, Vite for build tooling, Axios for API calls

Backend: Laravel 12.0 framework with PHP 8.2+, Laravel Sanctum for authentication, Eloquent ORM for database operations, comprehensive middleware for security

Database: MySQL with proper relationships and migrations, supporting complex data structures for users, profiles, projects, and interactions

Other Tools: Git for version control, Composer for PHP dependency management, npm for frontend packages, Postman for API testing, PowerShell scripts for development automation

---

2. App Plan

2.1 Project Scope

Define what is included and what is not included in this project.

In Scope:

- Complete user authentication system with role-based access (freelancer/employer)
- Comprehensive profile management for both user types with skills, portfolio, education tracking
- Advanced search and filtering system for discovering freelancers
- Dashboard analytics and activity tracking for both user types
- Project management system for employers with bookmark functionality

- Real-time availability tracking for freelancers
- Secure REST API with 10 functional endpoints
- Responsive web interface with modern UI/UX design
- Database relationships and data integrity management **Out of Scope:**

- Complete user authentication system with role-based access (freelancer/employer)
- Comprehensive profile management for both user types with skills, portfolio, education tracking
- Advanced search and filtering system for discovering freelancers
- Dashboard analytics and activity tracking for both user types
- Project management system for employers with bookmark functionality
- Real-time availability tracking for freelancers
- Secure REST API with 10 functional endpoints
- Responsive web interface with modern UI/UX design
- Database relationships and data integrity management

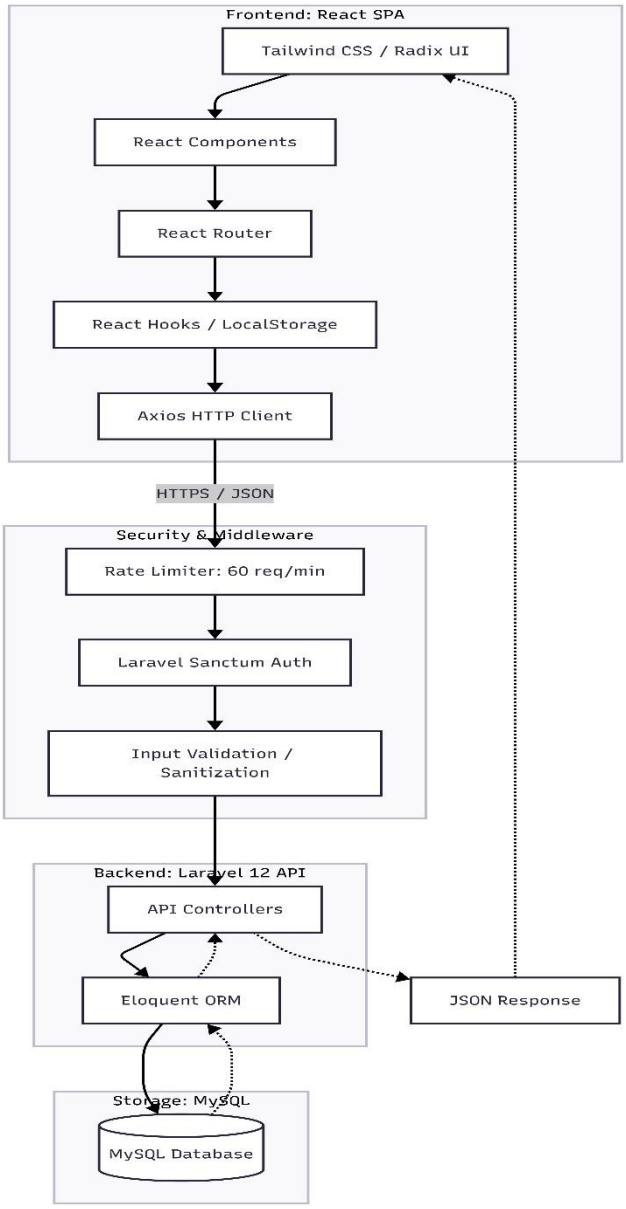
2.2 Objectives & Goals

State the specific, measurable objectives for this project.

1. **Launch a functional MVP within 8 weeks** that successfully connects at least 100 freelancers with 50 employers
2. **Achieve 95% uptime and security compliance** with comprehensive API testing and security measures
3. **Deliver intuitive user experience** with 80% user satisfaction score based on feedback and usability testing **2.3 User Stories & Use Cases User Story 1:**
    - As a freelancer, I want to create a comprehensive profile showcasing my skills and portfolio, so that I can attract potential employers and demonstrate my expertise
    - Acceptance Criteria:
    - Freelancer can register and create account with email verification
    - Profile includes sections for skills, education, work experience, and portfolio
    - Portfolio supports file uploads and project descriptions
    - Profile completeness indicator shows missing information **User Story 2:**
    - As an employer, I want to search and filter freelancers based on specific criteria, so that I can find the most qualified candidates for my projects
    - Acceptance Criteria:
    - Search functionality supports keywords, location, and skill filters
    - Results can be sorted by availability, rating, and experience

- Employers can bookmark freelancers for future reference
- Detailed freelancer profiles are accessible from search results **User Story 3:**
- As a freelancer, I want to manage my availability status and project capacity, so that employers know when I'm available for new work
- Acceptance Criteria:
- Availability can be set as available, busy, or unavailable
- Weekly working hours capacity can be specified
- Availability status is visible to employers browsing profiles
- Historical availability tracking is maintained **User Story 4:**
- As an employer, I want to manage my projects and track hired freelancers, so that I can organize my workflow and maintain project records
- Acceptance Criteria:
- Projects can be created with descriptions and requirements
- Freelancers can be assigned to specific projects
- Project status and progress tracking is available
- Project history and freelancer performance records are maintained

2.4 System Architecture



Freeport follows a client-server architecture with clear separation of concerns.

Frontend Layer (React SPA):

- React 18.3.1 with TypeScript for type safety
- Component-based architecture with reusable UI components
- React Router for client-side routing and navigation
- Axios for HTTP API communication

- State management through React hooks and local_Storage
- TailwindCSS with Radix UI for responsive, accessible design

Backend Layer (Laravel API):

- RESTful API architecture following REST conventions
- Laravel 12.0 with PHP 8.2+ providing robust framework foundation
- Laravel Sanctum for token-based authentication and authorization
- Eloquent ORM for database operations and relationship management
- Middleware for security, rate limiting, and request validation
- Comprehensive API controllers following single responsibility principle

Data Layer:

- MySQL database with normalized schema design
- Proper foreign key relationships ensuring data integrity
- Migration system for version-controlled database changes
- Seeders for development and testing data

Security Layer:

- JWT tokens via Laravel Sanctum for secure API access
- Input validation and sanitization preventing XSS attacks
- SQL injection protection through Eloquent ORM
- CSRF protection and rate limiting (60 requests/minute)
- Password hashing and secure authentication flows

Communication Flow:

- Frontend sends HTTP requests to Laravel API endpoints
- API validates authentication tokens and permissions
- Business logic processes requests in controllers
- Database operations performed through Eloquent models
- JSON responses returned to frontend for UI updates
- Error handling ensures proper HTTP status codes and messages

2.5 Development Timeline

Phase	Description	Timeline
Phase	Design & Planning	Week 1-2
Phase	Database Setup & Backend API	Week 2-4
Phase	Frontend Development	Week 3-6
Phase	Integration & Authentication	Week 5-7
Phase	Testing & Deployment	Week 7-8

---

3. UI/UX Design

3.1 Design Philosophy

Your design approach follows modern component-driven architecture with these principles:

- Accessibility First: All components use semantic HTML and ARIA attributes
- Consistent Design System: Built on shadcn/ui components with Radix UI primitives
- Responsive Design: Mobile-first approach with Tailwind CSS utilities
- User-Centric Navigation: Role-based navigation (freelancer vs employer)
- Clean Visual Hierarchy: Clear distinction between public and authenticated areas

3.2 Color Scheme

The application uses a semantic color system with Tailwind CSS:

- Primary Color: Blue (blue-600/blue-700) - Used for main CTAs, brand elements, and navigation highlights
- Secondary Color: Purple (purple-700) - Used in gradients and accent elements
- Accent Color: Gray (gray-50 to gray-900) - Used for backgrounds, borders, and text hierarchy
- Background Color: Light gray gradient (from-gray-50 via-blue-50/30 to-gray-50) - Creates subtle depth
- Text Colors:
- Primary text: text-gray-900 (dark contrast)
- Secondary text: text-gray-600 (subtle information)

- **Muted text: text-muted-foreground (disabled/placeholder states)**

3.3 Typography

The typography follows a clean, readable system:

- Font Family: System font stack (browser defaults) for optimal performance
- Heading Font Sizes:
- Hero titles: text-5xl lg:text-7xl (48px to 80px)
- Section titles: text-4xl (36px)
- Card titles: text-xl (20px)
- Dialog titles: text-lg (18px)
- Body Font Size: text-base (16px) with text-sm (14px) for secondary content
- Line Height: Default Tailwind ratios (leading-none for headings, leading-relaxed for paragraphs)
- Font Weights: font-medium for buttons, font-semibold for headings, font-bold for emphasis

3.4 UI Components Button

- Style: Rounded corners (rounded-md), multiple variants (default, outline, ghost, link)
- Usage:
- Primary actions use default variant with blue background • Secondary actions use outline variant
- Navigation links use ghost variant for subtle hover effects
- Sizes: sm (32px), default (36px), lg (40px), icon (36px square)
- States: Disabled states with opacity, hover transitions, focus rings

Input Fields

- Style: Clean borders with rounded corners, consistent height (h-9)
- Validation: Focus states with blue ring, error states with red accents
- Features: Placeholder styling, file upload support, disabled states
- Accessibility: Proper labels, ARIA attributes, keyboard navigation

Navigation Bar

- Public Navigation: Simple header with logo and main links (Home, Browse Freelancers, Browse Employers)
- Dashboard Navigation: Split layout with sidebar + top navigation
- Sidebar: Fixed width (256px), white background, role-based menu items
- Top Navigation: Search bar, notifications bell, user dropdown
- Mobile: Hamburger menu with overlay, responsive breakpoints **Cards**
- Style: Rounded corners (rounded-xl), subtle borders, white background
- Structure: Header, content, footer sections with consistent padding
- Usage: Profile cards, project listings, feature highlights
- Hover Effects: Shadow transitions (hover:shadow-lg), subtle transforms **Modal/Dialog**
- Style: Centered overlay with backdrop blur, rounded corners
- Animation: Smooth fade-in/scale transitions (animate-in, zoom-in-95)
- Structure: Header with title, content area, footer with actions
- Accessibility: Focus trapping, escape key handling, screen reader support **Additional Components**
- Badge: Small status indicators with variants (default, secondary, destructive, outline)
- Avatar: Circular profile images with fallback initials
- Dropdown Menu: Contextual menus with proper keyboard navigation
- Select: Styled dropdowns with consistent theming

Design System Integration

The application uses shadcn/ui as the foundation, which provides:

- Consistent component variants using class-variance-authority
- Radix UI primitives for accessibility
- Tailwind CSS for styling flexibility
- Dark mode support (CSS variables prepared)
- Comprehensive component composition patterns

The design emphasizes clarity, consistency, and accessibility while maintaining a professional appearance suitable for a freelancing marketplace platform.

3.5 Wireframes & Mockups Wireframe for FreePort:

https://www.figma.com/design/RPzTzxHFpNezXFXaKWUHJV/Freeport-WireFrame?node-id=01&t=4QiWd3ElJgcgRAx7-1

3.6 User Flows

Flow 1: User Registration & Onboarding

1. Landing Page → User clicks "Sign Up"
2. Registration Page → Select account type (Freelancer/Employer)
3. Form Completion:
    - Freelancer: First name, last name, email, password, phone, location
    - Employer: Company name, contact person, email, password, phone, industry
4. Account Creation → API creates user record + profile record
5. Auto Login → Redirect to role-specific dashboard
6. Profile Setup → Guided to complete profile details

Flow 2: User Authentication

1. Login Page → Enter email/password
2. Authentication → API validates credentials
3. Token Storage → JWT token saved to localStorage
4. Role Detection → User type determined and stored
5. Dashboard Redirect → Navigate to freelancer or employer dashboard

Flow 3: Freelancer Profile Management

1. Dashboard → Click "My Profile" or "Edit Profile" 2. Profile

Viewing → Display current profile information

3. Profile Editing:
    - Basic info: Name, email, phone, bio, location

- Profile picture upload with cropping
- Skills management (add/edit/delete)
- Portfolio work management
- Education records
- Availability settings

4. Save Changes → API updates profile data
5. Profile Display → Updated information shown across platform

Flow 4: Employer Project Management

1. Dashboard → Click "Projects" 2. Projects List

→ View all company projects

3. Create Project → Fill project details:
    - Title, description, budget, duration
    - Experience requirements, skills needed
4. Project Publishing → API saves project listing
5. Interest Tracking → Monitor freelancer interest via notifications
6. Project Management → Update status, view interested freelancers

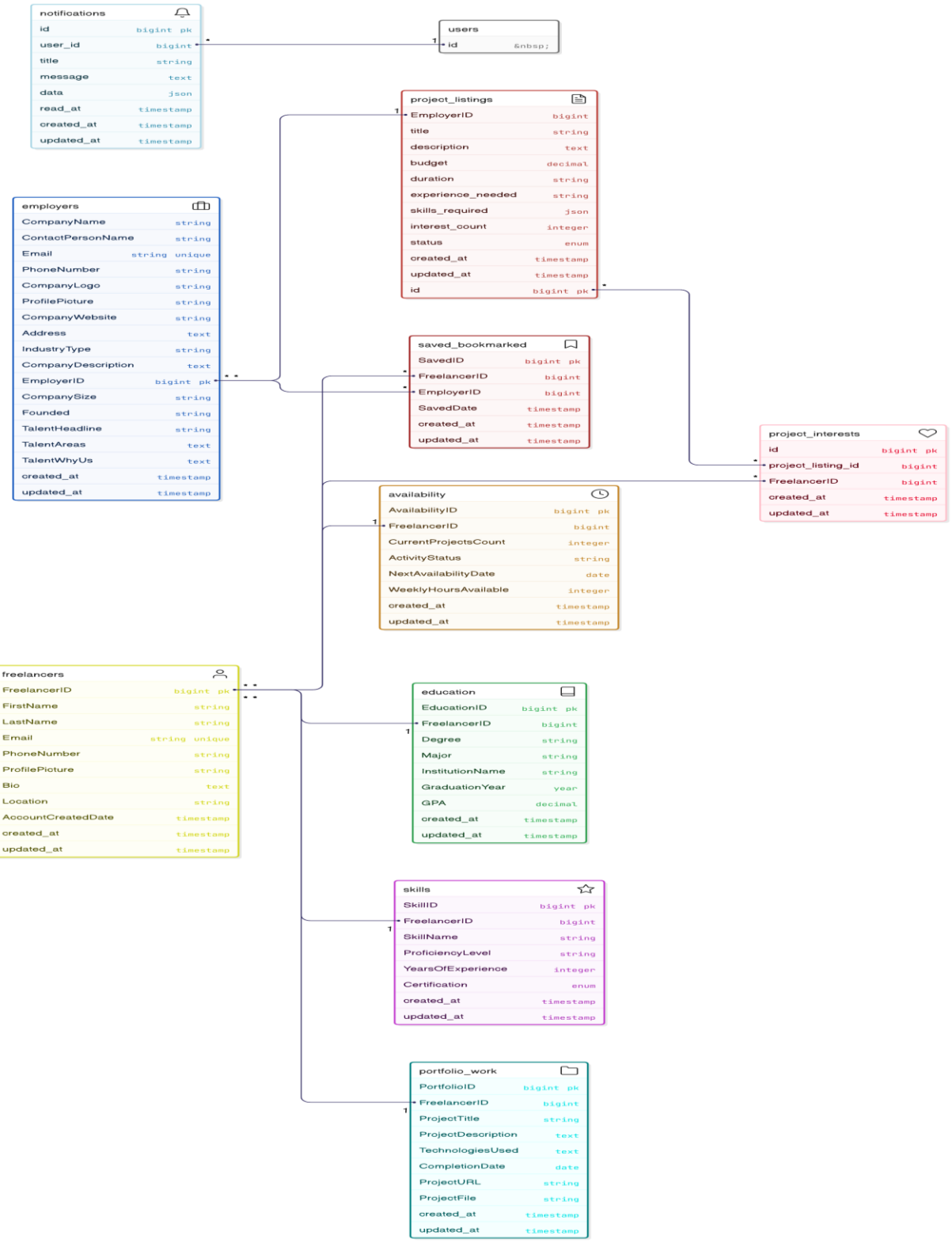
Flow 5: Project Discovery & Interest

1. Browse Companies → Search/filter employers
2. Company Profile → View employer details and open projects
3. Project Review → Read project requirements and details
4. Express Interest → Click "I'm Interested" button
5. Interest Registration → API records interest, increments count
6. Employer Notification → Real-time notification sent to employer
7. Profile Access → Employer can click notification to view freelancer profile

---

4. Database Architecture (ERD)

4.1 Entity Relationship Diagram



4.2 Entity Descriptions

Entity 1: users

Field	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier
name	VARCHAR(255)	NOT NULL	User's full name
email	VARCHAR(255)	UNIQUE, NOT NULL	Login email
password	VARCHAR(255)	NOT NULL	Hashed password
email_verified_at	TIMESTAMP	NULLABLE	Email verification
Field	**Data Type**	**Constraints**	**Description**
remember_token	VARCHAR(100)	NULLABLE	Remember me token
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Account creation
updated_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Last update

Entity 2: freelancers

Field	Data Type	Constraints	Description
FreelancerID	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier
FirstName	VARCHAR(255)	NOT NULL	First name
LastName	VARCHAR(255)	NOT NULL	Last name
Email	VARCHAR(255)	UNIQUE, NOT NULL	Contact email
PhoneNumber	VARCHAR(255)	NULLABLE	Contact phone
ProfilePicture	VARCHAR(255)	NULLABLE	Image file path
Bio	TEXT	NULLABLE	Professional bio
Location	VARCHAR(255)	NULLABLE	Geographic location
AccountCreatedDate	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Profile creation

Entity 3: employers

Field	Data Type	Constraints	Description

EmployerID	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier
CompanyName	VARCHAR(255)	NOT NULL	Company name
ContactPersonName	VARCHAR(255)	NOT NULL	Primary contact
Email	VARCHAR(255)	UNIQUE, NOT NULL	Company email
PhoneNumber	VARCHAR(255)	NULLABLE	Contact phone
CompanyLogo	VARCHAR(255)	NULLABLE	Logo file path
CompanyWebsite	VARCHAR(255)	NULLABLE	Website URL
Address	TEXT	NULLABLE	Physical address
Field	**Data Type**	**Constraints**	**Description**
IndustryType	VARCHAR(255)	NULLABLE	Industry sector

Entity 4: project_listings

Field	**Data Type**	**Constraints**	**Description**
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier
EmployerID	BIGINT	FOREIGN KEY, NOT NULL	Employer reference
title	VARCHAR(255)	NOT NULL	Project title
description	TEXT	NOT NULL	Project details
budget	DECIMAL(10,2)	NULLABLE	Project budget
duration	VARCHAR(255)	NULLABLE	Time duration
experience_needed	VARCHAR(255)	NULLABLE	Experience requirements
skills_required	JSON	NULLABLE	Required skills array

interest_count	INT	DEFAULT 0	Interest counter
status	ENUM	DEFAULT 'open'	Project status

Entity 5: notifications

Field	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier
user_id	BIGINT	FOREIGN KEY, NOT NULL	User reference
title	VARCHAR(255)	NULLABLE	Notification title
message	TEXT	NULLABLE	Notification content
data	JSON	NULLABLE	Additional data
read_at	TIMESTAMP	NULLABLE	Read timestamp
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Creation time

,

4.3 Relationships

Relationship 1: users ← One-to-One → freelancers

- Description: Each user can have one freelancer profile
- Implementation: users.email matches freelancers.email

Relationship 2: users ← One-to-One → employers

- Description: Each user can have one employer profile
- Implementation: users.email matches employers.email **Relationship 3:** employers ← One-to-Many → project_listings
- Description: Employers can post multiple projects
- Foreign Key: project_listings.EmployerID → employers.EmployerID

Relationship 4: freelancers ← One-to-Many → project_interests

- Description: Freelancers can express interest in multiple projects
- Foreign Key: project_interests.FreelancerID → freelancers.FreelancerID

Relationship 5: project_listings ← One-to-Many → project_interests

- Description: Projects can receive interest from multiple freelancers
- Foreign Key: project_interests.project_listing_id → project_listings.id
- Constraint: Unique constraint on (project_listing_id, FreelancerID) **Relationship 6:** users ← One-to-Many → notifications
- Description: Users receive multiple notifications
- Foreign Key: notifications.user_id → users.id

Relationship 7: freelancers ← One-to-Many → skills, portfolio_work, education, availability

- Description: Freelancers have multiple related records
- Foreign Keys: All reference freelancers.FreelancerID

Relationship 8: employers ← Many-to-Many → freelancers (via saved_bookmarked)

- Description: Employers can bookmark multiple freelancers
- Junction Table: saved_bookmarked with EmployerID and FreelancerID

4.4 Database Normalization First

Normal Form (1NF):

- All tables have primary keys
- No repeating groups (JSON used for skills_required, notification data)
- Atomic values in all fields **Second Normal Form (2NF):**
- All non-key attributes fully dependent on primary keys
- No partial dependencies in composite keys
- Related tables separated (skills, education, portfolio) **Third Normal Form (3NF):**
- No transitive dependencies
- All foreign keys properly indexed
- Cascade deletes maintain referential integrity **Additional Design Features:**
- Proper foreign key constraints with CASCADE deletes
- Indexes on frequently queried fields (email, user_id, read_at)
- Unique constraints prevent duplicate interests
- JSON fields for flexible data storage (skills, notifications)
- Timestamps for audit trails and soft deletes capability

5. Application Features & Functionality

5.1 Feature 1: User Authentication & Role Management

Description: Secure user registration and login system with role-based access control for freelancers and employers.

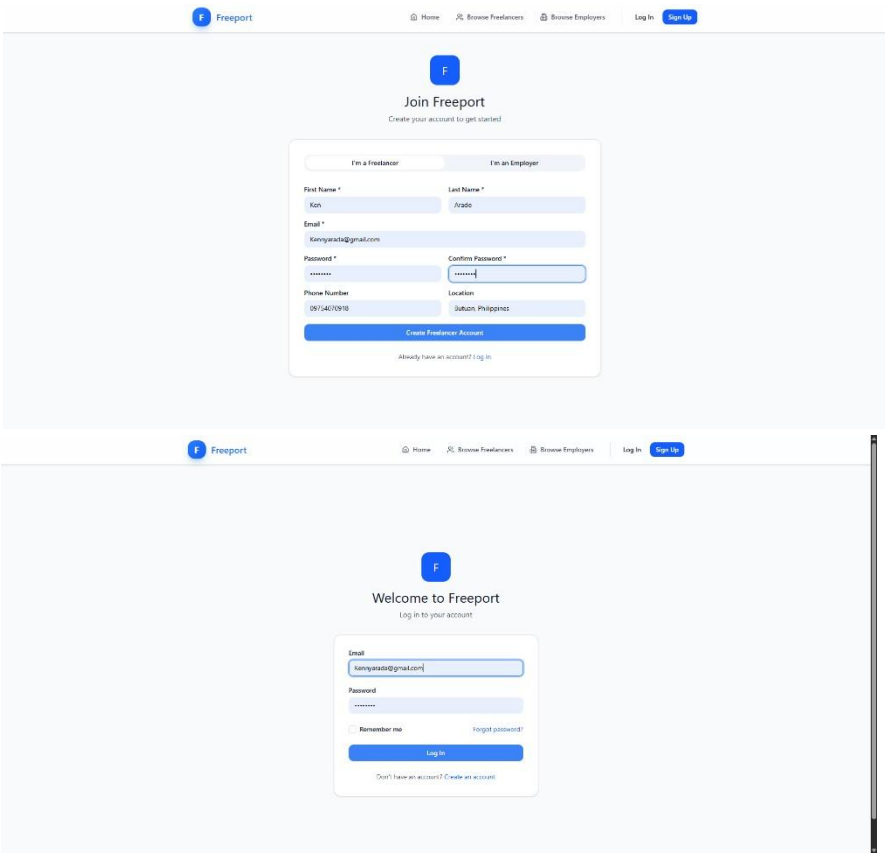
Functionality:

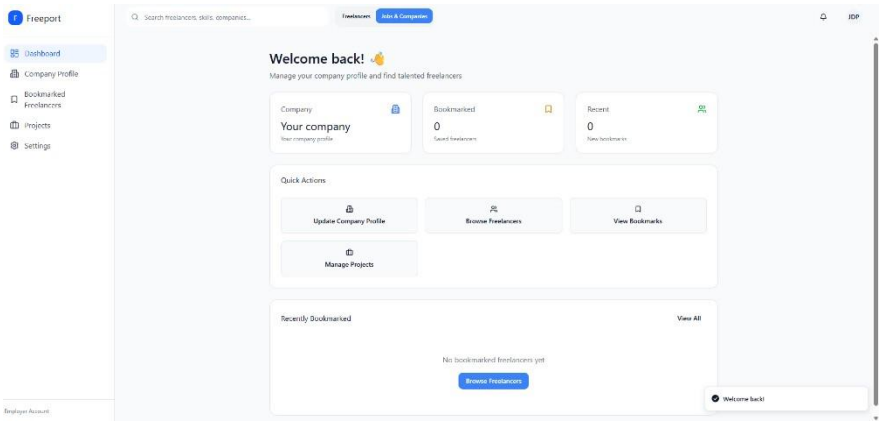
- User registration with account type selection (freelancer/employer)
- Email/password authentication with JWT tokens
- Automatic profile creation upon registration
- Role-based dashboard redirection
- Session management and logout

Implementation Details:

```
// AuthenticationController.php - Registration
$validated = $request->validate([
    'name' => 'required|string|min:4',
    'email' => 'required|string|email|max:255|unique:users',
    'password' => 'required|string|min:8',
    'user_type' => 'nullable|in:freelancer,employer',
]);
$user = User::create([
    'password' => Hash::make($validated['password']),
    'user_type' => $userType,
]);
```

Screenshots:





The Authentication Controller connects to the frontend Login/Sign-in Page as it requires the user to create an account and then stores any necessary data being give from the user to the database for confidential info on the character's purpose and privacy.

5.2 Feature 2: [Feature Name]

Description: Complete CRUD system for employers to manage job postings with interest tracking.

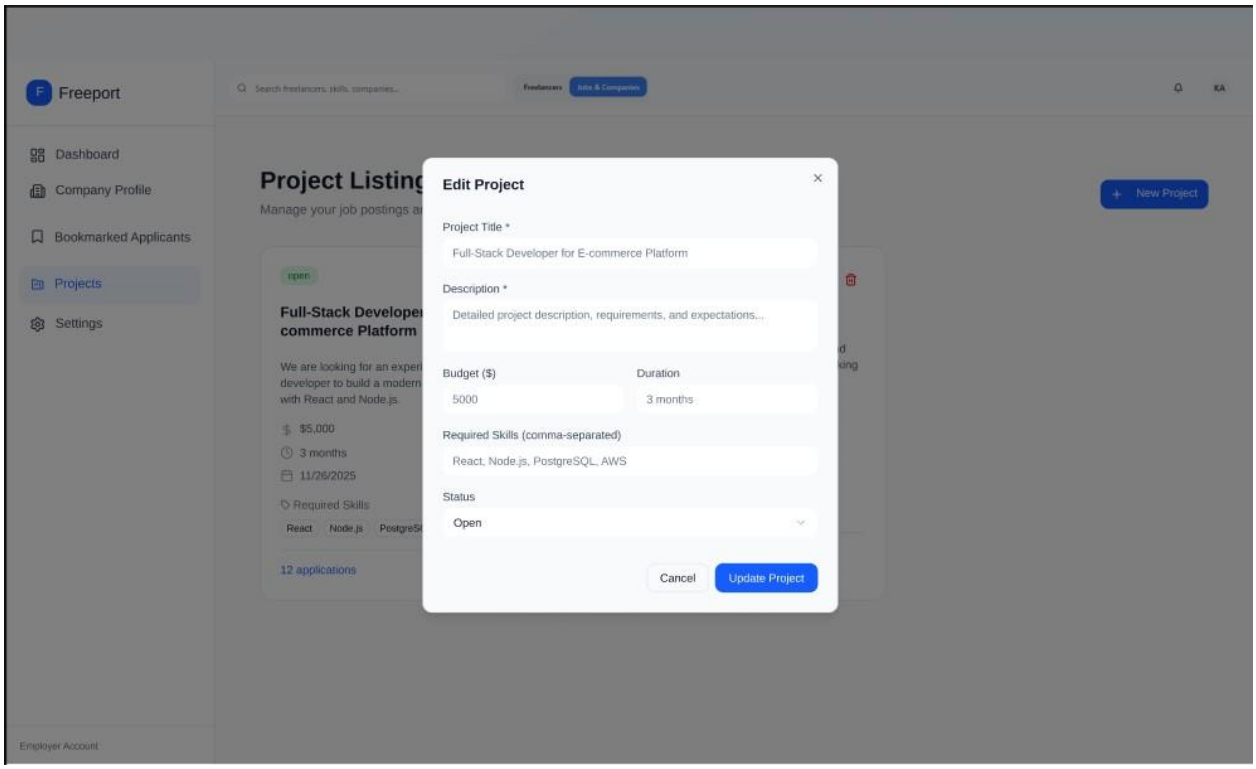
Functionality:

- Create, edit, delete project listings
- Add experience requirements and skill requirements
- Track freelancer interest counts
- Status management (open, in_progress, completed, closed)
- Real-time interest notifications

Implementation Details:

```
// ProjectsManagement.tsx - Form handling
const handleSubmit = async (e: React.FormEvent) => {
  const payload = {
    title: formData.title,
    description: formData.description,
    experience_needed: formData.experience_needed || undefined,
    skills_required: skillsArray,
  };
  await projectsApi.create(payload);
};
```

Screenshots:



Project listing tool that lists project roles (or job roles) for there company. That way, freelancers can see any available roles being listed from that particular company that they wish to partake.

---

6. Security & Error Handling

6.1 Security Measures Implemented

Authentication: Laravel Sanctum JWT tokens with secure password hashing **Authorization:** Role-based access control with middleware protection **Input Validation:** Comprehensive validation rules on all API endpoints **Data Protection:** Bcrypt password hashing, file upload restrictions **Protection Against Vulnerabilities:**

- CSRF protection via Laravel middleware
- SQL injection prevention through Eloquent ORM
- XSS protection with proper input escaping
- File upload validation (type, size limits)

6.2 Error Handling

Frontend Error Handling.

```
try {
  await apiCall();
} catch (error) {
```

```
  toast.error('Operation failed');
  setError('Failed to load data');
}
```

Backend Error Handling.

```
try {
  // Database operations
} catch (ValidationException $e) {
  return response()->json($e->errors(), 422);
} catch (Exception $e) {
  Log::error('Operation failed', ['error' => $e]);
  return response()->json(['message' => 'Server error'], 500);
}
```

---

7. Installation & Setup Instructions

7.1 Prerequisites

- **PHP 8.2+** with required extensions
- **MySQL 8.0+** or compatible database
- **Node.js 18+** and npm
- **Composer** for PHP dependencies
- **Git** for version control
-

7.2 Installation Steps

1. **Clone repository:**

```bash
git clone [repository-url]
cd Freeport
```

2. **Install backend dependencies:**

```bash
composer install
cp .env.example .env
php artisan key:generate
```

3. **Configure database:**

```bash
# Edit .env file with database credentials
DB_DATABASE=freeport
DB_USERNAME=root
DB_PASSWORD=
```

4. **Run migrations:**

```bash
php artisan migrate
```

5. **Install frontend dependencies:**

```bash
cd FFrontend
npm install
```

6. **Start development servers:**

```bash
# Backend (in root directory)
php artisan serve
# Frontend (in FFrontend directory)
npm run dev
```

7.3 Running the Application

- **Backend API:** http://localhost:8000
- **Frontend App:** http://localhost:3000
- **Default Access:** Register new accounts through the frontend interface

---

8. Testing

8.1 Test Cases

8.2 Known Issues & Limitations Current

Issues:

Test Case	Steps	Expected Result
User Registration	Register as freelancer	Account created, redirected to dashboard
Project Creation	Create project with all fields	Project saved, displayed in list
Image Upload	Upload profile picture with crop	Cropped image saved and displayed
Interest Notification	Express interest in project	Employer receives notification
Role-based Access	Try accessing opposite role pages	Proper redirection to own dashboard

- No real-time WebSocket implementation for live notifications
- File upload size limit hardcoded at 5MB
- No email verification system implemented

- Limited error recovery for failed image crops **Future Improvements:**
- Implement WebSocket for real-time updates
- Add email verification and password reset
- Implement advanced search with filters
- Add messaging system between freelancers and employers
- Implement payment processing and escrow system

---

9. Code Quality & Documentation

9.1 Code Structure

Frontend (/FFrontend)

```
├── src/
│   ├── api/          # Axios instances and API definitions
│   ├── components/   # Reusable UI components
│   ├── FFrontend/    # Custom React hooks
│   ├── layouts/      # Page wrappers
│   ├── pages/        # View components (Routes)
│   ├── types/        # TypeScript interfaces/types
│   └── App.tsx       # Main entry point
├── .env              # VITE_API_URL=http://localhost:8000/api
└── vite.config.ts    # Port 3000 config
```

Backend

```
├── app/
│   ├── Http/Controllers/  # Logic for handling requests
│   ├── Models/            # Eloquent (User, Company, etc.)
│   └── Providers/         # Service providers
├── config/                # Laravel configuration files
├── database/
```

```
|  ├── migrations/       # MySQL Schema definitions
|  └── seeders/          # Initial data population
├── storage/app/public/    # Root for file uploads
|  ├── profile-pictures/  # User avatars
|  └── company-logos/     # Business branding
├── routes/api.php        # API endpoints
└── .env                  # DB_PORT=3306, APP_URL=http://localhost:8000
```

9.2 Code Standards

Naming Conventions:

- **PHP/Laravel:** PascalCase for classes (FreelancerController), camelCase for methods (registerInterest), snake_case for database columns (first_name)
- **React/TypeScript:** PascalCase for components (ProjectsManagement), camelCase for variables (formData), kebab-case for file names (image-crop-dialog.tsx)
- **Database:** snake_case for table names (project_listings), PascalCase for primary keys (FreelancerID) **Code Style:**
- **Frontend:** ESLint + Prettier configured, React functional components with hooks, TypeScript strict mode
- **Backend:** PSR-12 standards, Laravel conventions, comprehensive validation rules
- **Consistent formatting:** 4-space indentation, trailing commas, proper imports organization

  Comments & Documentation:
- **PHPDoc blocks** for all controller methods and model properties
- **Inline comments** for complex business logic
- **Component documentation** with prop interfaces in TypeScript
- **API documentation** through route comments and validation rules

Version Control:

- **Git workflow:** Feature-based branching, conventional commits
- **.gitignore:** Proper exclusion of node_modules, .env, storage files
- **Commit messages:** Clear, descriptive format (feat:, fix:, docs:)

9.3 API Endpoints (if applicable) Authentication

Endpoints

POST /api/auth/register

- Description: User registration with role selection
- Request: {name, email, password, user_type?}
- Response: {token, user_info}
- Status: 201, 422, 500

POST /api/auth/login

- Description: User authentication
- Request: {email, password}
- Response: {token, user_info}
- Status: 200, 401, 422

GET /api/auth/user

- Description: Get current user info
- Request: Bearer token required
- Response: {user_info}
- Status: 200, 401

Freelancer Endpoints

GET /api/freelancers

- Description: List all freelancers (public)
- Request: Query params for search/filter
- Response: [{FreelancerID, FirstName, LastName, ...}]
- Status: 200

GET /api/freelancers/{id}

- Description: Get freelancer profile
- Request: Path parameter id
- Response: {freelancer_data}
- Status: 200, 404

PUT /api/freelancers/{id}

- Description: Update freelancer profile
- Request: {profile_data}
- Response: Updated freelancer object
- Status: 200, 403, 422

POST /api/freelancers/{id}/profile-picture

- Description: Upload profile picture
- Request: FormData with image file
- Response: {ProfilePicture: url}
- Status: 200, 422

Employer Endpoints

GET /api/employers

- Description: List all employers (public)
- Request: Query params for search/filter
- Response: [{EmployerID, CompanyName, ...}]
- Status: 200

GET /api/employers/{id}

- Description: Get employer profile
- Request: Path parameter id
- Response: {employer_data}
- Status: 200, 404

GET /api/employers/{id}/bookmarks

- Description: Get employer's bookmarked freelancers
- Request: Bearer token required

-
- Response: [freelancer_objects]

  Status: 200, 403

POST /api/employers/{id}/company-logo

- Description: Upload company logo
- Request: FormData with image file
- Response: {CompanyLogo: url}
- Status: 200, 422

Project Endpoints GET

/api/projects

- Description: List all projects (public)
- Request: Query params for filtering
- Response: [{id, title, description, ...}]
- Status: 200

POST /api/projects

- Description: Create new project
- Request: {title, description, budget, duration, experience_needed, skills_required}
- Response: Created project object
- Status: 201, 422, 403 **PUT /api/projects/{id}**
- Description: Update project
- Request: {project_data}
- Response: Updated project object
- Status: 200, 403, 422

POST /api/projects/{id}/interest

- Description: Express interest in project
- Request: Empty body (uses authenticated user)
- Response: {message, interest_count}
- Status: 200, 403, 409

Notification Endpoints

-

GET /api/notifications

Description: Get user notifications

- Request: Bearer token required
- Response: [{id, title, message, data, read_at}]
- Status: 200

POST /api/notifications/{id}/read

- Description: Mark notification as read
- Request: Empty body
- Response: {message}
- Status: 200, 404

CRUD Resource Endpoints

GET/POST/PUT/DELETE /api/availability GET/POST/PUT/DELETE /api/education GET/POST/PUT/DELETE /api/portfolio-work GET/POST/PUT/DELETE /api/skills GET/POST/PUT/DELETE /api/saved-bookmarked

- Description: Full CRUD operations for freelancer resources
- Request: Standard REST patterns
- Response: Resource objects or collections
- Status: 200, 201, 204, 404, 422

---

10. References & Resources

Development Frameworks & Libraries

- **Laravel 12.0** - https://laravel.com/docs/12.x
- **React 18.3.1** - https://react.dev/
- **TypeScript** - https://www.typescriptlang.org/
- **Tailwind CSS** - https://tailwindcss.com/
- **Vite** - https://vitejs.dev/ **UI Component Libraries**
- **shadcn/ui** - https://ui.shadcn.com/
- **Radix UI** - https://www.radix-ui.com/

-
- **Lucide React** - https://lucide.dev/

- **React Easy Crop** - https://github.com/ValentinH/react-easy-crop

Authentication & Security

- **Laravel Sanctum** - https://laravel.com/docs/12.x/sanctum
- **JWT Authentication** - https://jwt.io/
- **Bcrypt Password Hashing** - https://laravel.com/docs/12.x/hashing

Development Tools

- **Axios** - https://axios-http.com/
- **React Router DOM** - https://reactrouter.com/
- **Sonner Toasts** - https://sonner.emilkowal.ski/
- **Class Variance Authority** - https://cva.style/

Database & Storage

- **MySQL 8.0+** - https://dev.mysql.com/doc/refman/8.0/en/
- **Eloquent ORM** - https://laravel.com/docs/12.x/eloquent
- **Laravel Migrations** - https://laravel.com/docs/12.x/migrations **API & Testing**
- **PHPUnit** - https://phpunit.de/
- **Laravel Testing** - https://laravel.com/docs/12.x/testing
- **REST API Design** - https://restfulapi.net/

Documentation & Standards

- **PSR-12 Coding Standard** - https://www.php-fig.org/psr/psr-12/
- **TypeScript Handbook** - https://www.typescriptlang.org/docs/
- **React Best Practices** - https://react.dev/learn/thinking-in-react

Design Resources

- **Figma Design System** - Component library references
- **Color Palette** - Tailwind CSS color system
- **Typography** - System font stack optimization

Deployment & DevOps

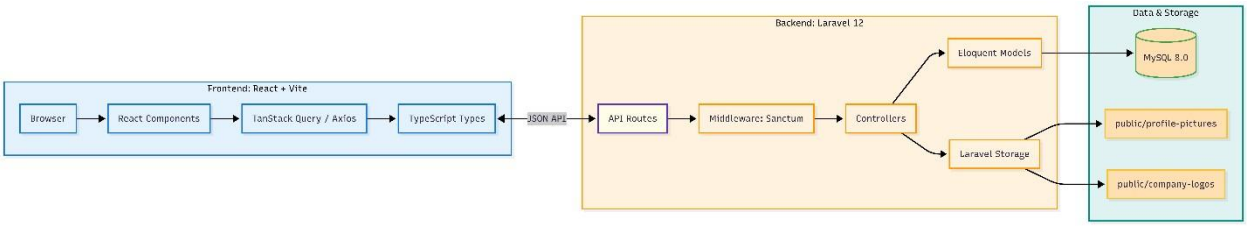
- **Composer** - https://getcomposer.org/
- **NPM** - https://www.npmjs.com/

- **Git Version Control** - https://git-scm.com/doc

---

11. Appendix

11.1 Additional Diagrams



11.2 Supplementary Information Environment Configuration:

```
# Backend .env variables
DB_DATABASE=freeport
DB_USERNAME=root
DB_PASSWORD=
SANCTUM_STATEFUL_DOMAINS=localhost:3000
# Frontend .env variables
VITE_API_BASE_URL=http://localhost:8000/api
```

Performance Optimizations:

- Database indexing on email, user_id, foreign keys
- Image compression and size validation (5MB limit)
- API rate limiting (60 requests/minute)
- React lazy loading for components
- Vite build optimization **Security Hardening:**
- CORS configuration for frontend domain
- File type validation (images only)
- SQL injection prevention via Eloquent
- XSS protection with input sanitization
- CSRF protection via Laravel middleware **Development Workflow:**

```
# Start development environment
```

```
php artisan serve          # Backend server
cd FFrontend && npm run dev  # Frontend server
php artisan migrate --force # Database setup
```

File Upload Specifications:

- Allowed formats: jpg, jpeg, png, gif, webp, svg
- Maximum size: 5MB
- Storage paths: storage/app/public/profile-pictures/, storage/app/public/company-logos/
- Public URL: /storage/profile-pictures/, /storage/company-logos/ **Database Connection Pool:**
- Default Laravel MySQL configuration
- Connection timeout: 60 seconds
- Query timeout: 30 seconds
- Max connections: 100

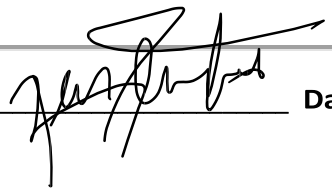
Frontend Build Configuration:

- Vite dev server: localhost:3000
- Build output: FFrontend/build/
- Proxy configuration: /api → localhost:8000
- Source maps enabled in development **API Response Standards:**
- Success: 200-299 status codes
- Validation errors: 422 with detailed messages
- Authentication errors: 401/403
- Server errors: 500 with logging
- Consistent JSON response format **Browser Compatibility:**
- Chrome 90+, Firefox 88+, Safari 14+, Edge 90+
- Modern JavaScript features (ES2020)
- CSS Grid and Flexbox support required
- LocalStorage for client-side persistence

Monitoring & Logging:

- Laravel Log channels: daily, error, stack
- Frontend error handling with toast notifications
- API request/response logging in development
- Database query logging for optimization **Future Enhancement Roadmap:**
- Real-time notifications via WebSocket
- Advanced search with Elasticsearch
- Payment processing integration
- Mobile app development
- Multi-language support
- Advanced analytics dashboard

This comprehensive documentation provides a complete overview of the Freeport freelancing marketplace application, covering all technical aspects from architecture to deployment and future improvements.

Student/Team Signature:  **Date:** 12/24/25