ApplyForge - Comprehensive Technical Overview

Table of Contents

Project Overview

ApplyForge is a comprehensive job application automation platform designed to streamline the job search and application process. The platform consists of three main components:

• **Client (Frontend)**: React-based web application

• **Server (Backend)**: FastAPI-based REST API

• **Agent**: Python-based automation agent using Playwright

Core Purpose

ApplyForge automates the job application process by:

• Scraping job listings from multiple portals (LinkedIn, Naukri, etc.)

• Parsing and managing user resumes and profiles

• Automating job applications using AI-powered form filling

• Tracking application status and responses

• Managing user profiles, skills, and experiences

Target Audience

ApplyForge is specifically designed for **early-career software developers and IT professionals in India** with 6-24 months of experience, transforming the tedious manual job search process into an intelligent, automated workflow.

AI-Powered Features

ApplyForge leverages advanced AI and machine learning technologies to provide intelligent automation throughout the job application lifecycle.

1. Intelligent Job Discovery Engine

Multi-Portal Scraping

• **Continuous Monitoring**: Automatically scrapes LinkedIn, Naukri, Indeed, and other Indian job portals using Playwright for robust, headless browser automation

• **Real-Time Updates**: Maintains job posting timestamps and updates status to ensure users only see active, relevant opportunities

• **Anti-Bot Resistance**: Uses human-like interaction patterns (random delays, mouse movements) to avoid detection

Smart Deduplication

• **URL-Based Matching**: Automatically identifies and removes duplicate job postings by URL to maintain clean data

• **Cross-Portal Detection**: Recognizes the same job posted across multiple platforms

NLP-Powered Extraction

• **Unstructured Data Processing**: Uses natural language processing to extract skills, roles, experience requirements, and job metadata from job descriptions

• **Keyword Identification**: Automatically identifies critical keywords and technical requirements

• **Metadata Enrichment**: Extracts company information, location details, and job type classifications

Freshness Tracking

• **Posting Age Monitoring**: Tracks when jobs were first posted and last updated

• **Stale Job Removal**: Automatically archives or removes outdated job listings

• **Priority Scoring**: Newer jobs receive higher priority in recommendations

2. AI-Powered Matching System

ApplyForge uses a sophisticated **weighted scoring algorithm** that analyzes multiple dimensions to match users with the most relevant job opportunities:

Matching Dimensions

Role Alignment (40% weight)

• Compares user's preferred roles against job role requirements

• Uses normalized `user_role_map` and `job_role_map` tables for accurate matching

• Supports role synonyms and related positions

Skills Matching (30% weight)

• Evaluates technical skill overlap through `user_skill_map` and `job_skill_map`

• Supports skill synonyms and related technologies (e.g., React.js ≈ ReactJS)

• Identifies transferable skills and frameworks

Experience Validation (15% weight)

• Checks if user's experience timeline meets minimum requirements

• Considers internship experience and current role relevance

• Validates years of experience in specific technologies

Location & Remote Preferences (10% weight)

• Considers geographic preferences and commute distance

• Prioritizes remote work options when user prefers remote

• Supports hybrid work arrangements

Current Role Relevance (5% weight)

• Prioritizes jobs related to user's current or most recent position

• Considers career progression and growth opportunities

Match Score Output

• **Score Range**: 0-100 for each job

• **Skill Gap Analysis**: Detailed breakdown showing which competencies to highlight or develop

• **Recommendation Engine**: Suggests skills to learn based on frequently missed opportunities

3. Dynamic Resume Generation & Tailoring

Unlike traditional platforms that use static resume templates, ApplyForge generates a **custom resume for each application** tailored to the specific job description.

LangChain AI Agent

• **Orchestration**: Uses LangChain with tool calling to orchestrate resume generation through structured workflows

• **Context-Aware**: Analyzes job description to understand employer priorities

• **Multi-Model Support**: Compatible with OpenAI GPT-4 or open-source LLMs

Keyword Optimization

• **ATS Compatibility**: Extracts critical keywords from job descriptions

• **Strategic Placement**: Incorporates keywords 2-3 times throughout the resume to satisfy ATS algorithms

• **Natural Integration**: Avoids keyword stuffing while maintaining readability

ATS-Friendly Formatting

• **Parser-Optimized**: Enforces strict formatting rules for maximum ATS compatibility

• **Simple Layouts**: Uses standard fonts (Arial, Calibri) and clear section headers

• **No Complex Elements**: Avoids images, tables, columns, and graphics that confuse parsers

• **Bullet Point Structure**: Uses consistent bullet points for easy parsing

Content Rewriting

• **Achievement Emphasis**: AI rewrites experience bullet points to emphasize relevant achievements

• **Job-Specific Terminology**: Weaves in job-specific terminology while maintaining authenticity

• **Quantifiable Results**: Highlights metrics and measurable outcomes

Multi-Version Support

• **Base Profiles**: Stores multiple base resume profiles (frontend-focused, backend-focused, full-stack)

• **Dynamic Assembly**: Assembles the optimal combination for each application

• **Version Tracking**: Links each application to the specific resume version used

4. Automated Application Management

LangChain Tool Calling Architecture

ApplyForge implements a structured AI agent with discrete tools for end-to-end automation:

Available Tools:

• `search_jobs(query: str)` → Retrieves matching job listings from database

• `extract_job_details(url: str)` → Parses job requirements and metadata from job pages

• `rewrite_resume(job_desc: str, profile: dict)` → Generates tailored resume for specific job

• `generate_cover_letter(job_desc: str, resume: str)` → Creates personalized cover letter

• `submit_application(job_url: str, resume: str, cover_letter: str)` → Automates form filling and submission

One-Click Apply

• **Review Interface**: Users review AI-generated materials (resume + cover letter)

• **Single-Click Submission**: Apply with a single click after approval

• **Bulk Operations**: Apply to top 10-20 matches simultaneously

Full Lifecycle Tracking

• **Status Monitoring**: Tracks applications through complete funnel:

- Submitted → Shortlisted → Interview → Rejected → Offer

• **Response Detection**: Monitors email for company responses

• **Timeline Tracking**: Records all status changes with timestamps

Bulk Application Support

• **Batch Processing**: Apply to multiple jobs simultaneously

• **Rate Limiting**: Throttled submission rates to avoid portal rate limiting

• **Queue Management**: Manages application queue with priority scoring

Automated Follow-ups

• **Reminder System**: Generates notifications for following up with companies

• **Configurable Timing**: User-defined time periods for follow-up reminders

• **Template Generation**: AI-generated follow-up email templates

5. Analytics & Insights Dashboard

Match Score Trends

• **Profile Quality**: Visualizes how profile quality improves over time

• **Average Match Scores**: Tracks average match scores across all jobs

• **Improvement Suggestions**: AI-powered recommendations for profile enhancement

Application Conversion Funnel

• **Conversion Rates**: Tracks conversion from application → shortlist → interview → offer

• **Success Metrics**: Identifies which types of applications have highest success rates

• **Bottleneck Analysis**: Highlights where applications are getting stuck

Role Demand Analysis

• **Market Trends**: Shows which roles are most frequently posted

• **Competition Levels**: Indicates where competition is highest

• **Emerging Opportunities**: Identifies growing job categories

Company Response Metrics

• **Response Rates**: Tracks which companies respond most frequently

• **Average Response Time**: Measures how long companies take to respond

• **Profile Fit**: Identifies which companies are best matches for user profile

Skill Gap Recommendations

• **Missing Skills**: AI-powered suggestions for skills to learn

• **Impact Analysis**: Shows which skills would unlock the most opportunities

• **Learning Resources**: Provides links to courses and tutorials

6. Comprehensive User Profiles

Users build rich, detailed profiles that power the matching and resume generation systems:

Personal Details

• Name, email, phone, professional summary

• Social links (GitHub, LinkedIn, Naukri portfolio, personal website)

• Location preferences and remote work availability

Skills & Roles

• **Normalized Storage**: Dedicated skills and roles tables with many-to-many mappings

• **Proficiency Levels**: Self-assessed skill proficiency (beginner, intermediate, advanced, expert)

• **Skill Categories**: Organized by category (languages, frameworks, tools, soft skills)

• **Role Preferences**: Primary and secondary role preferences

Work Experience Timeline

• **Detailed Employment History**: Start/end dates, company names, job titles

• **Responsibilities**: Bullet-pointed responsibilities and achievements

• **Internship Flags**: Distinguishes internships from full-time roles

• **Current Role Indicator**: Marks current employment

• **Skill Mappings**: Links specific skills used in each role

Education & Certifications

• **Academic Background**: Degrees, institutions, graduation dates, GPA

• **Professional Certifications**: Certification name, issuing organization, date obtained

• **Ongoing Education**: Tracks in-progress courses and certifications

Resume Versions

• **Multiple Files**: Stores different resume versions for different job types

• **Base vs. Tailored**: Distinguishes between base resumes and job-specific tailored versions

• **Version History**: Maintains history of all generated resumes

• **File Storage**: Resumes stored in Cloudflare R2 with secure URLs

Architecture

High-Level Architecture

Component Breakdown

1. **Client (React + Vite)**

• **Location**: `/applyforge-docker/client`

• **Port**: 3000 (via Nginx)

• **Build Tool**: Vite

• **State Management**: Zustand

• **Data Fetching**: TanStack Query (React Query)

• **Routing**: React Router v7

• **UI Framework**: Radix UI + Tailwind CSS

• **Form Handling**: React Hook Form + Zod validation

2. **Server (FastAPI)**

• **Location**: `/applyforge-docker/server`

• **Port**: 8000

• **Framework**: FastAPI 0.122.0

• **ORM**: SQLAlchemy 2.0.35 (async)

• **Database Driver**: asyncpg

• **Task Queue**: Celery 5.3.6

• **Cache**: Redis 5.0.1

• **AI/ML**: LangChain, Google Gemini, spaCy

3. **Agent (Playwright)**

• **Location**: `/agent`

• **Framework**: Playwright (browser automation)

• **GUI**: Flet (optional)

• **Purpose**: Job scraping and automated applications

Technology Stack

Frontend Dependencies

```
{
  "core": {
    "react": "19.2.0",
    "react-router-dom": "7.9.6",
    "vite": "7.2.4"
  },
  "state": {
    "zustand": "5.0.8",
    "@tanstack/react-query": "5.90.11"
  },
  "ui": {
    "@radix-ui/*": "Multiple components",
    "tailwindcss": "3.4.17",
    "lucide-react": "0.555.0",
    "framer-motion": "12.23.26"
  },
  "forms": {
    "react-hook-form": "7.66.1",
    "zod": "4.1.13",
    "@hookform/resolvers": "5.2.2"
  },
```

```
  "utilities": {
    "axios": "1.13.2",
    "date-fns": "4.1.0",
    "react-phone-input-2": "2.15.1"
  }
}
```

Backend Dependencies

```
# Core Framework
fastapi==0.122.0
uvicorn==0.38.0
pydantic==2.12.5

# Database
sqlalchemy==2.0.35
asyncpg==0.30.0
alembic==1.17.2
psycopg2-binary==2.9.10

# Authentication & Security
python-jose==3.3.0
passlib==1.7.4
bcrypt==4.1.3

# Task Queue
celery==5.3.6
redis==5.0.1

# AI & NLP
langchain==0.3.13
langchain-google-genai==2.0.8
spacy==3.8.3
ollama==0.4.4
nltk==3.9.1
scikit-learn==1.6.0

# File Processing
pypdf==4.2.0
... (truncated)
```

Database Schema

Core Tables Overview

The database consists of **45+ models** organized into the following categories:

1. **User Management**

• `users` - Core user accounts

• `user_profiles` - Extended user information

• `user_sessions` - Active user sessions

• `user_notifications` - User notification system

2. **Profile Data**

• `user_experiences` - Work experience

• `user_educations` - Educational background

• `user_projects` - Personal/professional projects

• `user_certifications` - Certifications and licenses

• `user_accomplishments` - Achievements and awards

• `user_languages` - Language proficiencies

• `user_resumes` - Resume files and metadata

3. **Skills System**

- `skills` - Master skills library
- `user_skills` - User skill mappings
- `experience_skills` - Skills linked to experiences
- `project_skills` - Skills linked to projects

4. **Job Management**

- `jobs` - Job listings
- `job_applications` - Application tracking
- `job_matches` - Job-user matching scores
- `job_portals` - Supported job portals
- `user_targeting_profiles` - Job search preferences
- `generated_cover_letters` - AI-generated cover letters

5. **Agent System**

- `agents` - Registered agent instances
- `agent_api_keys` - Agent authentication keys
- `agent_forge_tasks` - Agent task queue
- `portal_configs` - Portal scraping configurations

6. **Billing & Subscriptions**

- `subscriptions` - User subscription plans
- `payments` - Payment transactions
- `system_usage_credits` - Usage tracking

7. **System**

- `system_logs` - Audit logs
- `alerts` - System alerts
- `activities` - User activity tracking

Key Model Details

User Model (`users`)

```
class User:
    # Identity
    id: UUID (PK)
    email: str (unique, indexed)
    password_hash: str

    # Verification
    email_verified: bool
    email_verification_token: str
    email_verification_token_expires_at: datetime

    # OAuth
    oauth_provider: OAuthProvider (google, github, linkedin)
    oauth_provider_id: str
    oauth_access_token: str
    oauth_refresh_token: str

    # Account Status
    account_status: AccountStatus (pending, active, suspended, deleted)
    role: UserRole (user, admin, super-admin)

    # Security
    failed_login_attempts: int
```

```
    locked_until: datetime
    last_login_at: datetime
    last_login_ip: inet
    two_factor_enabled: bool
    two_factor_secret: str

    # Session Management
... (truncated)
```

Job Application Model (`job_applications`)

```
class JobApplication:
    # Identity
    id: UUID (PK)
    user_id: UUID (FK -> users)
    job_id: UUID (FK -> jobs)

    # Status
    status: JobApplicationStatus (saved, applied, interviewing, rejected, accepted)
    applied_at: datetime
    response_received_at: datetime
    interview_scheduled_at: datetime
    offer_received_at: datetime

    # Documents
    resume_used_id: UUID (FK -> user_resumes)
    cover_letter_used_id: UUID (FK -> user_resumes)

    # Tracking
    application_source: ApplicationSource (manual, automated, imported)
    notes: text
    next_followup_at: datetime
    response_received: bool

    # Automation
    automation_status: str (started, completed, failed)
    automation_logs: jsonb
    screenshots: jsonb (list of file keys)
    error_message: text
    preferred_method: ApplicationMethod (auto, manual, easy_apply)

... (truncated)
```

Agent Model (`agents`)

```
class Agent:
    # Identity
    id: UUID (PK)
    agent_id: str (unique, indexed)
    user_id: UUID (FK -> users)
    api_key_id: UUID (FK -> agent_api_keys)

    # Metadata
    name: str (user-friendly name)
    hostname: str
    platform: str (Windows, Linux, Darwin)
    version: str

    # Status
    status: str (online, offline, idle)
    last_heartbeat: datetime
    last_task_id: UUID (FK -> agent_forge_tasks)

    # Performance Metrics
    total_tasks_assigned: int
    total_tasks_completed: int
    total_tasks_failed: int
    success_rate: float
    total_execution_time_seconds: int
    average_execution_time_seconds: float
    last_task_completed_at: datetime

    # Rate Limiting
    max_tasks_per_hour: int (default: 60)
    tasks_this_hour: int
... (truncated)
```

Database Enums

User Enums

```
class AccountStatus(str, Enum):
    PENDING_VERIFICATION = "pending-verification"
    ACTIVE = "active"
    SUSPENDED = "suspended"
    DELETED = "deleted"

class UserRole(str, Enum):
    USER = "user"
    ADMIN = "admin"
    SUPER_ADMIN = "super-admin"

class OAuthProvider(str, Enum):
    GOOGLE = "google"
    GITHUB = "github"
    LINKEDIN = "linkedin"
```

Job Enums

```
class JobApplicationStatus(str, Enum):
    SAVED = "saved"
    APPLIED = "applied"
    INTERVIEWING = "interviewing"
    REJECTED = "rejected"
    OFFER_RECEIVED = "offer-received"
    ACCEPTED = "accepted"
    WITHDRAWN = "withdrawn"

class ApplicationSource(str, Enum):
    MANUAL = "manual"
    AUTOMATED = "automated"
    IMPORTED = "imported"

class ApplicationMethod(str, Enum):
    AUTO = "auto"
    MANUAL = "manual"
    EASY_APPLY = "easy-apply"
```

Agent Enums

```
class TaskType(str, Enum):
    SCRAPE = "scrape-jobs"
    APPLY = "apply-job"

class TaskStatus(str, Enum):
    PENDING = "pending"
    ASSIGNED = "assigned"
    IN_PROGRESS = "in-progress"
    COMPLETED = "completed"
    FAILED = "failed"
    CANCELLED = "cancelled"
```

API Endpoints

API Structure

• **Base URL**: `http://localhost:8000`

• **API Version**: `/api/v1`

• **Documentation**: `/docs` (Swagger UI), `/redoc` (ReDoc)

Endpoint Categories

1. **Authentication** (`/api/v1/auth`)

```
POST   /auth/register          - User registration
POST   /auth/login             - User login (cookie-based)
POST   /auth/logout            - User logout
POST   /auth/refresh           - Refresh access token
POST   /auth/verify-email      - Verify email address
POST   /auth/forgot-password   - Request password reset
POST   /auth/reset-password    - Reset password with token
GET    /auth/me                - Get current user info
```

2. **Users** (`/api/v1/users`)

```
GET    /users/me               - Get current user profile
PUT    /users/me               - Update current user
DELETE /users/me               - Delete account
GET    /users/me/profile       - Get detailed profile
POST   /users/me/profile       - Create/update profile
```

3. **Profile Management** (`/api/v1/profile`)

```
# Experience
GET    /profile/experiences    - List experiences
POST   /profile/experiences    - Create experience
PUT    /profile/experiences/{id} - Update experience
DELETE /profile/experiences/{id} - Delete experience

# Education
GET    /profile/educations     - List educations
POST   /profile/educations     - Create education
PUT    /profile/educations/{id} - Update education
DELETE /profile/educations/{id} - Delete education

# Projects
GET    /profile/projects       - List projects
POST   /profile/projects       - Create project
PUT    /profile/projects/{id}  - Update project
DELETE /profile/projects/{id}  - Delete project

# Certifications
GET    /profile/certifications - List certifications
POST   /profile/certifications - Create certification
PUT    /profile/certifications/{id} - Update certification
DELETE /profile/certifications/{id} - Delete certification

# Accomplishments
GET    /profile/accomplishments - List accomplishments
POST   /profile/accomplishments - Create accomplishment
PUT    /profile/accomplishments/{id} - Update accomplishment
DELETE /profile/accomplishments/{id} - Delete accomplishment

... (truncated)
```

4. **Resumes** (`/api/v1/resumes`)

```
GET    /resumes                - List user resumes
POST   /resumes                - Upload resume
GET    /resumes/{id}           - Get resume details
DELETE /resumes/{id}           - Delete resume
GET    /resumes/{id}/download  - Download resume file
POST   /resumes/{id}/parse     - Trigger resume parsing
```

5. **Skills** (`/api/v1/skills`)

```
GET    /skills/search          - Search skills library
GET    /skills/user            - Get user skills
POST   /skills/user            - Add user skill
DELETE /skills/user/{id}       - Remove user skill
GET    /skills/suggestions     - Get skill suggestions
```

6. **Jobs** (`/api/v1/jobs`)

```
GET    /jobs                   - List available jobs
GET    /jobs/{id}              - Get job details
POST   /jobs/search            - Search jobs
GET    /jobs/portals           - List supported portals
```

7. **Applications** (`/api/v1/applications`)

```
GET    /applications           - List user applications
POST   /applications           - Create application
GET    /applications/{id}      - Get application details
PUT    /applications/{id}      - Update application
DELETE /applications/{id}      - Delete application
POST   /applications/{id}/apply - Trigger automated application
```

8. **Targeting** (`/api/v1/targeting`)

```
GET    /targeting/profiles     - List targeting profiles
POST   /targeting/profiles     - Create targeting profile
PUT    /targeting/profiles/{id} - Update targeting profile
DELETE /targeting/profiles/{id} - Delete targeting profile
```

9. **Notifications** (`/api/v1/notifications`)

```
GET    /notifications          - List notifications
PUT    /notifications/{id}/read - Mark as read
DELETE /notifications/{id}     - Delete notification
POST   /notifications/read-all - Mark all as read
```

10. **Dashboard** (`/api/v1/dashboard`)

```
GET    /dashboard/stats        - Get dashboard statistics
GET    /dashboard/recent-activity - Get recent activity
GET    /dashboard/insights     - Get AI insights
```

11. **Agent Management** (`/api/v1/agents`)

```
GET    /agents                 - List user agents
POST   /agents/register        - Register new agent
PUT    /agents/{id}            - Update agent
DELETE /agents/{id}            - Delete agent
POST   /agents/{id}/heartbeat  - Send heartbeat
```

12. **Agent Keys** (`/api/v1/agent-keys`)

```
GET    /agent-keys             - List API keys
POST   /agent-keys             - Create API key
DELETE /agent-keys/{id}        - Revoke API key
```

13. **Agent Forge** (`/api/v1/agent-forge`)

```
GET    /agent-forge/tasks      - Get pending tasks
POST   /agent-forge/tasks/{id}/result - Submit task result
GET    /agent-forge/blueprints - Get portal blueprints
GET    /agent-forge/resume     - Get user resume for agent
GET    /agent-forge/profile    - Get user profile for agent
```

14. **Admin** (`/api/v1/admin`)

```
GET    /admin/users            - List all users
GET    /admin/users/{id}       - Get user details
PUT    /admin/users/{id}       - Update user
DELETE /admin/users/{id}       - Delete user
POST   /admin/users            - Create user
GET    /admin/logs             - Get system logs
GET    /admin/stats            - Get system statistics
POST   /admin/config           - Update system config
GET    /admin/config           - Get system config
```

15. **Scrapers** (`/api/v1/scrapers`)

```
POST   /scrapers/trigger       - Trigger manual scrape
GET    /scrapers/status        - Get scraper status
```

Authentication Flow

Frontend Architecture

Directory Structure

```
client/
■■■ public/
■■■ src/
■   ■■■ assets/          # Images, fonts, static files
■   ■■■ components/      # Reusable UI components (36 files)
■   ■   ■■■ ui/          # Radix UI wrappers
■   ■   ■■■ forms/       # Form components
■   ■   ■■■ layout/      # Layout components
■   ■■■ constants/       # App constants
■   ■■■ features/        # Feature modules (15 files)
■   ■   ■■■ auth/
■   ■   ■■■ layout/
■   ■   ■■■ profile/
■   ■■■ hooks/           # Custom React hooks
■   ■■■ lib/             # Utilities
■   ■■■ pages/           # Page components (55 files)
■   ■   ■■■ public/      # Landing, About
■   ■   ■■■ auth/        # Login, Signup, etc.
■   ■   ■■■ protected/   # Dashboard, Profile, etc.
■   ■   ■   ■■■ user/
```

```
■   ■   ■   ■■■ admin/
■   ■   ■■■ error/       # Error pages
■   ■■■ routes/          # Routing configuration
■   ■■■ services/        # API services (24 files)
■   ■   ■■■ api/         # API client modules
■   ■■■ stores/          # Zustand stores
■   ■■■ App.jsx
■   ■■■ main.jsx
■   ■■■ index.css
■■■ Dockerfile
... (truncated)
```

State Management (Zustand)

Auth Store

```
// stores/authStore.js
const useAuthStore = create((set) => ({
  user: null,
  isAuthenticated: false,
  isLoading: true,

  login: async (credentials) => { /* ... */ },
  logout: async () => { /* ... */ },
  initAuth: async () => { /* ... */ },
  updateUser: (userData) => { /* ... */ }
}));
```

Routing Structure

Public Routes

- `/` - Landing page
- `/about` - About page
- `/login` - Login page
- `/signup` - Signup page
- `/forgot-password` - Password reset request
- `/reset-password` - Password reset form
- `/verify-email` - Email verification

Protected Routes (User)

- `/welcome` - Welcome/onboarding page
- `/dashboard` - User dashboard
- `/profile-setup/*` - Profile setup wizard
- `/profile-setup/resume` - Resume upload
- `/profile-setup/personal` - Personal info
- `/profile-setup/education` - Education
- `/profile-setup/skills` - Skills
- `/profile-setup/experience` - Experience
- `/profile-setup/certifications` - Certifications
- `/profile-setup/accomplishments` - Accomplishments
- `/profile-setup/projects` - Projects
- `/applications` - Job applications
- `/jobs` - Job listings
- `/apply` - Apply page (agent control)
- `/notifications` - Notifications

• `/agent/pair` - Agent pairing
• `/settings/*` - Settings
- `/settings/account` - Account settings
- `/settings/security` - Security settings
- `/settings/billing` - Billing settings
- `/settings/notifications` - Notification preferences
- `/settings/privacy` - Privacy settings
- `/settings/advanced` - Advanced settings
• `/help` - Help center

Admin Routes

• `/admin/dashboard` - Admin dashboard
• `/admin/users` - User management
• `/admin/users/:userId` - User details
• `/admin/users/new` - Add new user
• `/admin/audit-logs` - Audit logs
• `/admin/config` - System configuration
• `/admin/danger` - Danger zone (destructive actions)

Key UI Components

Form Components

• `Input` - Text input with validation
• `Select` - Dropdown select
• `Checkbox` - Checkbox input
• `Switch` - Toggle switch
• `PhoneInput` - Phone number input with country code
• `DatePicker` - Date selection

Layout Components

• `ProtectedLayout` - Layout for authenticated users
• `AdminLayout` - Layout for admin pages
• `Sidebar` - Navigation sidebar
• `TopBar` - Top navigation bar

UI Components (Radix UI)

• `Dialog` - Modal dialogs
• `AlertDialog` - Confirmation dialogs
• `Dropdown` - Dropdown menus
• `Toast` - Toast notifications
• `Accordion` - Collapsible sections
• `Tabs` - Tabbed interface
• `Progress` - Progress indicators

- `Avatar` - User avatars
- `Popover` - Popover menus
- `ScrollArea` - Custom scrollbars

API Service Layer

```
// services/api/auth.api.js
export const authAPI = {
  login: (credentials) => axios.post('/auth/login', credentials),
  logout: () => axios.post('/auth/logout'),
  register: (data) => axios.post('/auth/register', data),
  // ...
};

// services/api/profile.api.js
export const profileAPI = {
  getProfile: () => axios.get('/users/me/profile'),
  updateProfile: (data) => axios.put('/users/me/profile', data),
  // ...
};
```

TanStack Query Integration

```
// Example usage in components
const { data: profile, isLoading } = useQuery({
  queryKey: ['profile'],
  queryFn: profileAPI.getProfile
});

const updateMutation = useMutation({
  mutationFn: profileAPI.updateProfile,
  onSuccess: () => {
    queryClient.invalidateQueries(['profile']);
  }
});
```

Agent System

Agent Architecture

Agent Components

1. **Main Entry Point** (`main.py`)

- Initializes agent with unique ID
- Handles first-run setup (browser authentication)
- Manages agent loop (polling for tasks)
- Sends heartbeat to server
- Processes tasks (SCRAPE and APPLY)

2. **API Client** (`client.py`)

```
class APIClient:
    def login(self) -> bool
    def register_agent(self) -> dict
    def send_heartbeat(self) -> None
    def get_pending_tasks(self) -> List[dict]
    def submit_result(task_id, status, data, error_log) -> None
    def get_blueprint(portal) -> dict
    def get_user_profile(self) -> dict
```

3. **Scrapers**

LinkedIn Scraper (`scrapers/linkedin.py`)

• Searches for jobs using keywords and location

• Extracts job details (title, company, location, description)

• Handles pagination

• Returns structured job data

Naukri Scraper (`scrapers/naukri.py`)

• Similar to LinkedIn scraper

• Supports Naukri-specific selectors

• Handles internship listings

Scraper Executor (`scrapers/executor.py`)

```
class ScraperExecutor:
    async def scrape(url, blueprint) -> List[dict]:
        # Initialize browser
        # Navigate to URL
        # Extract jobs using blueprint selectors
        # Return job data
```

4. **Application Handler** (`handlers/applier.py`)

```
class ApplicationHandler:
    async def apply(payload) -> dict:
        # Navigate to job URL
        # Fill application form
        # Upload resume
        # Submit application
        # Return result with screenshots
```

5. **Browser Service** (`core/browser_service.py`)

• Manages Playwright browser instances

• Handles session persistence

• Provides context and page management

6. **State Manager** (`core/state_manager.py`)

• Stores agent ID

• Saves user profile data

• Manages browser session storage

Agent Task Flow

Agent Configuration

```
# config.py
class Settings:
    SERVER_URL: str = "http://localhost:8000"
    POLL_INTERVAL: int = 10  # seconds
    HEADLESS: bool = True
    DEBUG: bool = False
```

Portal Blueprints

Portal blueprints define scraping selectors for each job portal:

```
{
  "portal": "linkedin",
  "domain": "linkedin.com",
  "search_url_template": "https://www.linkedin.com/jobs/search/?keywords={keywords}&location={location}",
  "selectors": {
```

```
      "job_card": ".job-card-container",
      "title": ".job-card-list__title",
      "company": ".job-card-container__company-name",
      "location": ".job-card-container__metadata-item",
      "link": ".job-card-list__title",
      "description": ".jobs-description__content"
    }
  }
```

Authentication & Authorization

Authentication Methods

Email/Password

- Password hashing: bcrypt

- Minimum password length: 8 characters

- Failed login attempts tracking

- Account lockout after 5 failed attempts (15 minutes)

OAuth Providers

- Google

- GitHub

- LinkedIn

Two-Factor Authentication (2FA)

- TOTP-based

- Backup codes support

Token-Based Authentication

JWT Tokens

```
# Access Token
{
  "sub": "user_id",
  "email": "user@example.com",
  "role": "user",
  "exp": 1234567890  # 30 minutes
}

# Refresh Token
{
  "sub": "user_id",
  "type": "refresh",
  "exp": 1234567890  # 7 days
}
```

Cookie-Based Sessions

• `access_token` - HTTP-only, Secure, SameSite=Lax

• `refresh_token` - HTTP-only, Secure, SameSite=Lax

• Session stored in Redis for quick validation

Authorization Levels

User Roles

User (default)

- Access to own profile and data
- Can create applications
- Can manage agents

Admin

- All user permissions
- View all users
- View system logs
- Manage system configuration

Super Admin

- All admin permissions
- Delete users
- Access danger zone
- Modify system-critical settings

Security Features

Session Management

- Maximum 5 concurrent sessions per user
- Session expiry tracking
- Automatic cleanup of expired sessions

Rate Limiting

- Login attempts: 5/minute, 20/hour
- General API: 120/minute, 1000/hour, 5000/day
- Skill search: 30/minute
- Resume uploads: 10/day

GDPR Compliance

- Terms acceptance tracking
- Privacy policy acceptance
- Marketing consent management
- Data processing consent
- Soft delete for user accounts

Security Headers

- CORS configuration
- Trusted host middleware
- Content Security Policy (via Nginx)

Deployment

Docker Compose Architecture

```
services:
  # Database
  db:
    image: postgres:16-alpine
    ports: ["5433:5432"]

  # Cache
  redis:
    image: redis:7-alpine
    ports: ["6379:6379"]

  # Backend
  server:
    build: ./server
    ports: ["8000:8000"]
    depends_on: [db, redis]

  # Worker
  worker:
    build: ./server
    command: celery -A app.core.celery_app worker
    depends_on: [redis, db]

  # Monitor
  flower:
    image: mher/flower
    ports: ["5555:5555"]
    depends_on: [redis, worker]

  # Frontend
... (truncated)
```

Environment Variables

Server (.env)

```
# Database
DATABASE_URL=postgresql://user:pass@host/db

# Security
SECRET_KEY=your-secret-key-here
DEBUG=False
ENVIRONMENT=production

# Redis
REDIS_URL=redis://redis:6379/0

# Storage (Cloudflare R2)
R2_ENDPOINT_URL=https://...
R2_ACCESS_KEY_ID=...
R2_SECRET_ACCESS_KEY=...
R2_BUCKET_NAME=applyforge

# AI
GOOGLE_API_KEY=...

# SMTP
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=...
SMTP_PASSWORD=...
```

Client (.env.local)

```
VITE_API_URL=http://localhost:8000
VITE_APP_NAME=ApplyForge
```

Agent (.env)

```
SERVER_URL=http://localhost:8000
AGENT_EMAIL=user@example.com
AGENT_PASSWORD=password
```

Deployment Steps

Database Setup

```bash
# Run migrations
cd server
alembic upgrade head

# Create superuser
python create_superuser.py
```

Build and Start Services

```bash
docker-compose up -d --build
```

Verify Services

- Frontend: http://localhost:3000
- Backend API: http://localhost:8000/docs
- Flower Monitor: http://localhost:5555
- Adminer: http://localhost:8080

Production Considerations

Database

- Use managed PostgreSQL (Neon.tech, AWS RDS, etc.)
- Enable connection pooling
- Set up automated backups

File Storage

- Configure Cloudflare R2 for resume storage
- Set up CDN for static assets

Monitoring

- Sentry for error tracking
- Flower for Celery monitoring
- System logs in `/logs` directory

Security

- Use HTTPS in production
- Configure proper CORS origins
- Enable rate limiting
- Set strong SECRET_KEY

Key Features

1. **Resume Parsing**

• Upload PDF/DOCX resumes
• AI-powered parsing using Google Gemini
• Automatic extraction of:
- Personal information
- Work experience
- Education
- Skills
- Projects
- Certifications
• Sync parsed data to user profile

2. **Job Scraping**

• Multi-portal support (LinkedIn, Naukri)
• Keyword and location-based search
• Automated job discovery
• Job matching based on user profile

3. **Automated Applications**

• Browser automation using Playwright
• Form auto-fill
• Resume upload
• Screenshot capture
• Error handling and retry logic

4. **Profile Management**

• Comprehensive profile builder
• Skills library with autocomplete
• Experience tracking
• Education history
• Project portfolio
• Certifications and accomplishments

5. **Application Tracking**

• Status tracking (saved, applied, interviewing, etc.)
• Response monitoring
• Follow-up reminders
• Notes and documentation

6. **Agent System**

• Desktop agent for automation
• Task queue management
• Heartbeat monitoring
• Performance metrics

• Rate limiting

7. **Admin Dashboard**

• User management

• System logs and audit trails

• Configuration management

• Analytics and insights

8. **Notification System**

• In-app notifications

• Email notifications (planned)

• Real-time updates

9. **Billing & Subscriptions** (Planned)

• Subscription tiers

• Usage tracking

• Payment processing

Development Workflow

Local Development Setup

Clone Repository

```bash
git clone <repository-url>
cd applyforge
```

Setup Server

```bash
cd applyforge-docker/server
python -m venv venv
source venv/bin/activate # or venv\Scripts\activate on Windows
pip install -r requirements.txt

# Create .env file
cp .env.example .env
# Edit .env with your configuration

# Run migrations
alembic upgrade head

# Start server
uvicorn app.main:app --reload
```

```

**Setup Client**

```bash
cd applyforge-docker/client
npm install

# Create .env.local file
cp .env.example .env.local

# Start dev server
npm run dev
```

Setup Agent

```bash
cd agent
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# Create .env file
cp .env.example .env

# Install Playwright browsers
playwright install

# Run agent
python main.py
```

Database Migrations

```
# Create new migration
alembic revision --autogenerate -m "description"

# Apply migrations
alembic upgrade head

# Rollback
alembic downgrade -1
```

Testing

```
# Server tests
cd server
pytest

# Client tests
cd client
npm test
```

System Workflow Architecture
```

This section outlines the end-to-end lifecycle of a user interaction with ApplyForge, emphasizing the division of labor between the **Central Server (Brain)** and the **Local Agent (Hands)**.

Architecture Overview: Distributed Forge

ApplyForge uses a distributed architecture where:

• **Server (Brain)**: Handles intelligence, data storage, AI processing, and coordination

• **Agent (Hands)**: Executes browser automation, scraping, and form filling on user's local machine

This separation ensures:

• **IP Authenticity**: All portal interactions use the user's residential IP

• **Session Persistence**: Browser sessions remain authenticated on user's machine

• **Privacy**: Sensitive credentials never leave the user's device

• **Anti-Bot Resistance**: Requests appear as legitimate user activity

Phase A: Onboarding & Initialization

1. User Registration

• User visits the Web Client and creates an account

• Email verification via Server-sent verification link

• Account activation and initial profile setup

2. Agent Installation

• User is prompted to download **Agent Forge** desktop application

• Bundled Python executable for easy installation (no Python required)

• Cross-platform support (Windows, Linux, macOS)

3. Resume Ingestion

• **Upload**: User uploads "Master Resume" (PDF/DOCX) via Web Client

• **Storage**: Server uploads file to **Cloudflare R2** and stores `file_url` in database (`user_resumes` table)

• **Parsing**: Server parses resume using LLM/OCR to extract structured data:

- Personal information (name, email, phone)

- Skills and proficiency levels

- Work experience with dates and responsibilities

- Education background

- Projects and certifications

• **Database Population**: Extracted data populates normalized tables:

- `user_profiles`, `user_experiences`, `user_educations`

- `user_skills`, `user_projects`, `user_certifications`

4. Data Completion

• User reviews parsed data on Web Client

• Manually fills missing required fields

• Validates and corrects any parsing errors

• Profile marked as complete when all required fields filled

Phase B: Agent Activation (Local Machine)

5. Agent Setup

• User launches **Agent Forge** on local machine

• **First-Run Authentication**:

- Agent opens Chromium browser window

- Prompts user to log in to:

- Email Provider (Gmail/Outlook) for verification code retrieval

- Job Portals (LinkedIn, Naukri) for application submission

- User completes login in browser

6. Session Persistence

• **Storage State Capture**: Agent saves browser `storage_state` (cookies/tokens) locally

• **Future Sessions**: Enables scraping and verification without re-login

• **Security**: Session data encrypted and stored in user's home directory

7. Polling Loop

• Agent enters infinite loop, polling Server Task Queue every 30 seconds

• **Endpoint**: `GET /api/v1/agent-forge/tasks`

• **Heartbeat**: Sends heartbeat every 30 seconds to indicate online status

• **Task Retrieval**: Receives pending tasks (SCRAPE_JOBS, APPLY_JOB)

8. Local Caching

• **Initial Fetch**: Agent fetches User's Structured Data (`user_data`) from Server

• **Cached Data Includes**:

- Personal Details (name, email, phone)

- Professional URLs (GitHub, LinkedIn, portfolio)

- Work History (companies, titles, dates)

- Education (degrees, institutions, dates)

- Skills (technical and soft skills)

- Projects (names, descriptions, technologies)

• **Excludes**: Resume files (fetched on-demand per application)

• **Optimization**: In-memory or SQLite cache for instant form filling

• **Cache Invalidation**: Server sends `CACHE_INVALIDATE` signal when user updates profile

Phase C: Job Discovery (Intelligence Gathering)

9. Targeting Configuration

• User navigates to `/apply` page on Web Client

• Defines search criteria:

- **Keywords**: Job titles, technologies, roles

- **Location**: City, state, remote preferences

- **Portals**: LinkedIn, Naukri, Indeed (select multiple)

- **Filters**: Experience level, job type, company size

10. Scrape Task Creation

• **Server** creates `SCRAPE_JOBS` task in queue (`agent_forge_tasks` table)

• Task contains:

- Search parameters (keywords, location)

- Target portals

- Targeting instructions (filters, preferences)

- Priority score

11. Execution (Agent-Side Scraping)

• **Agent** picks up `SCRAPE_JOBS` task from queue

• **Navigation**:

- Opens target portals (LinkedIn, Naukri) using authenticated browser session

- Uses user's local IP and browser fingerprint

- Avoids IP bans and rate limiting

• **Scraping**:

- Navigates search pages with human-like delays

- Extracts job listings using portal-specific selectors

- Collects: title, company, location, description, URL, posting date

• **Deduplication**: Removes duplicate jobs locally by URL

• **Reporting**: Sends raw job data back to Server via API

12. Ingestion

• **Server** validates scraped data

• **Processing**:

- Checks for existing jobs by URL (deduplication)

- Extracts skills and requirements using NLP

- Calculates match scores for user

- Enriches with company data

• **Storage**: Creates entries in `jobs` table for valid, unique listings

• **Notification**: Notifies user of new matching jobs

Phase D: Application Execution (The Core Loop)

13. Analysis & Optimization

• For each job user selects (or auto-selected by policy):

- **Job Description Extraction**: Server extracts full JD from job URL

- **Resume Tailoring**:

- Server invokes LLM Service (Google Gemini/GPT-4)

- Analyzes JD to identify key requirements and keywords

- Rewrites user's Master Resume specifically for this JD:

- Reorders sections to prioritize relevant experience
- Rewrites summary to match job requirements
- Emphasizes relevant skills and achievements
- Incorporates job-specific keywords 2-3 times
- Generates ATS-friendly PDF
- **Storage**: Uploads optimized resume to **Cloudflare R2**
- **Linking**: Links resume to specific `job_application` record

14. Application Task

• **Server** queues `APPLY_JOB` task containing:
- `job_url`: Direct link to application page
- `optimized_resume_url`: R2 URL of tailored resume
- `user_data`: Cached profile data
- `cover_letter` (optional): AI-generated cover letter

15. Agent Execution

• **Agent** receives task from queue
• **Download**: Downloads job-specific optimized resume from R2
• **Navigation**: Opens job URL in authenticated browser context
• **Form Filling**:
- Identifies form fields using selectors
- Fills fields using cached `user_data`:
- Personal info (name, email, phone)
- Work experience (current company, title)
- Education (degree, institution)
- Skills (from user profile)
- Handles dynamic forms and multi-step applications
• **Verification (If Needed)**:
- If portal requires email verification (e.g., Workday):
- Agent switches tabs to pre-authenticated Email session
- Retrieves verification code from latest email
- Returns to application and completes verification
• **Resume Upload**: Uploads the optimized resume file
• **Submission**: Clicks submit button and waits for confirmation
• **Screenshot Capture**: Takes screenshots of each step for audit trail

16. Completion & Feedback

• **Agent** reports `SUCCESS` or `FAILED` to Server with:
- Status (completed/failed)
- Screenshots (stored in R2)
- Error logs (if failed)
- Execution time

• **Server** updates application status:

- `job_applications.status` → `APPLIED`

- `job_applications.applied_at` → current timestamp

- `job_applications.automation_logs` → execution details

- `job_applications.screenshots` → R2 URLs

• **Real-Time Notification**:

- Server triggers WebSocket/Toast notification to Web Client

- Message: *"Application submitted to [Company Name] for [Job Title]"*

- User sees instant feedback in browser

Key Constraints & Design Decisions

IP & Identity

• **All portal interactions** (scraping and applying) happen on the **Agent (User's Device)**

• Ensures requests come from trusted residential IP

• Uses user's legitimate browser fingerprint

• Avoids detection as bot or automated system

Resume Strategy

• **Every application uses a unique, JD-optimized resume**

• Generated on-the-fly by Server before Agent applies

• Maximizes ATS compatibility and relevance

• Stored permanently for audit and reuse

Session Management

• User must be logged into portals (LinkedIn, Gmail) in Agent's browser **once**

• Agent maintains session via cookie persistence

• No need to re-login unless session expires

• Secure storage of session data on local machine

Error Handling

• Agent reports all failures with detailed logs

• Server can retry failed applications

• Screenshots provide visual debugging

• User notified of failures with actionable feedback

Rate Limiting

• Agent respects portal rate limits

• Configurable delays between applications

• Prevents account suspension

• Monitors hourly/daily application quotas

Future Enhancements

Planned Features

Email Integration

- Gmail/Outlook integration
- Application response tracking
- Automated follow-ups

Cover Letter Generation

- AI-powered cover letter creation
- Job-specific customization
- Template library

Interview Preparation

- Company research
- Common questions
- Mock interviews

Analytics Dashboard

- Application success rate
- Response time tracking
- Job market insights

Mobile App

- React Native application
- Push notifications
- On-the-go application management

Advanced Matching

- ML-based job recommendations
- Skill gap analysis
- Salary insights

Team Collaboration

- Recruiter portal
- Application sharing
- Team analytics

Technical Debt & Known Issues

Current Limitations

Agent Scalability

- Single agent per user
- No distributed task processing
- Limited error recovery

Resume Parsing

- Accuracy depends on resume format
- Limited support for non-standard formats
- No multi-language support

Job Portal Support

- Only LinkedIn and Naukri supported
- Portal changes can break scrapers
- No fallback mechanisms

Performance

- No caching for job listings
- Large profile queries can be slow
- No pagination on some endpoints

Planned Improvements

Implement Redis caching for job listings

Add pagination to all list endpoints

Improve error handling in agent

Add retry logic for failed tasks

Implement webhook support for real-time updates

Add comprehensive test coverage

Optimize database queries with proper indexing

Implement API versioning strategy

Conclusion

ApplyForge is a comprehensive job application automation platform built with modern technologies and best practices. The system is designed to scale and can be extended with additional features as needed.

Key Strengths

• **Modular Architecture**: Clear separation of concerns

• **Modern Tech Stack**: React, FastAPI, PostgreSQL, Redis

• **Automation**: Playwright-based browser automation

• **Security**: JWT authentication, role-based access control

• **Scalability**: Docker-based deployment, Celery workers

• **Extensibility**: Plugin-based scraper system

Contact & Support

For questions or issues, please refer to the project documentation or contact the development team.

Document Version: 1.0

Last Updated: December 29, 2025

Author: ApplyForge Development Team

Generated on December 29, 2025 at 23:09