JOB APPLICATION TRACKER API

Full Project Report

Developed By

Lokpavan P

Technology: ASP.NET Core 8 Web API

Database: SQLite with Entity Framework Core

Authentication: JWT Bearer Tokens

February 2026

1. Project Overview

The Job Application Tracker API is a production-ready RESTful web service built with ASP.NET Core 8. It enables users to efficiently manage their job search journey by tracking job applications, companies, and interview rounds through a secure and well-structured API.

The system was designed following industry best practices including clean architecture, layered separation of concerns, the repository pattern, and JWT-based authentication with role-based access control.

1.1 Project Goals

- Provide a complete backend API for tracking job applications
- Implement secure user authentication and authorization
- Support filtering, searching, and pagination of application data
- Enable interview round tracking per job application
- Follow clean architecture and SOLID principles throughout
- Be deployable to cloud platforms with minimal configuration

1.2 Live Deployment

The API has been successfully deployed to Render and is publicly accessible at:

`https://job-application-tracker-api-b9t2.onrender.com`

2. Technology Stack

The following technologies were carefully selected to build a robust, maintainable, and scalable API.

Category	Technology	Version	Purpose
Language	C#	12.0	Primary programming language
Framework	ASP.NET Core Web API	8.0 (LTS)	Web framework and HTTP pipeline
ORM	Entity Framework Core	8.0.0	Database access and migrations
Database	SQLite	Latest	Lightweight local/production database
Authentication	JWT Bearer	8.0.0	Stateless token-based auth
Password Hashing	BCrypt.Net-Next	4.0.3	Secure password hashing (cost 11)
Object Mapping	AutoMapper	12.0.1	Entity-to-DTO mapping
API Documentation	Swashbuckle / Swagger	6.6.2	Interactive API docs and testing
Validation	Data Annotations	Built-in	Request model validation
Containerization	Docker	Latest	Container image for deployment
Cloud Hosting	Render	N/A	Free-tier cloud deployment platform
Version Control	Git + GitHub	N/A	Source control and CI/CD trigger

3. System Architecture

The API follows a strict 4-layer clean architecture where each layer only communicates with the layer directly below it through interfaces, ensuring loose coupling and high testability.

3.1 Layer Overview

Layer	Folder	Responsibility
Presentation	Controllers/	Handles HTTP requests, validates input, returns HTTP responses
Business Logic	Services/	Implements all business rules and orchestrates operations
Data Access	Repositories/	Abstracts all database queries using EF Core
Data / Domain	Models/Entities/	Defines database entities and enumerations
Transfer Objects	Models/DTOs/	Defines request and response contract shapes
Infrastructure	Data/, Middleware/, Mapping/	DbContext, exception handling, AutoMapper profiles

3.2 Request Flow

Every API request follows this exact path through the system:

```
HTTP Request -> GlobalExceptionMiddleware -> JWT Authentication -> Controller
-> Service -> Repository -> EF Core -> SQLite Database
```

3.3 Design Patterns Used

Pattern	Where Applied	Benefit
Repository Pattern	All Repositories	Decouples data access from business logic
Dependency Injection	Throughout all layers	Loose coupling, testability, lifetime management
DTO Pattern	All Controllers/Services	Separates API contracts from database entities
Generic Repository	Repository<T> base class	Reduces code duplication for CRUD operations
Middleware Pipeline	GlobalExceptionMiddleware	Centralized error handling across all endpoints
Options Pattern	JwtSettings	Strongly-typed configuration binding

4. Project Structure

The solution is organized into two projects for clean separation between application code and test code.

4.1 Main Project: JobTrackerAPI

Folder / File	Contents
Controllers/	AuthController, CompaniesController, JobApplicationsController, InterviewRoundsController, HealthController
Services/	AuthService, CompanyService, JobApplicationService, InterviewRoundService, JwtService
Services/Interfaces/	IAuthService, ICompanyService, IJobApplicationService, IInterviewRoundService
Repositories/	Repository<T> (base), UserRepository, CompanyRepository, JobApplicationRepository, InterviewRoundRepository
Repositories/Interfaces/	IRepository<T>, IUserRepository, ICompanyRepository, IJobApplicationRepository, IInterviewRoundRepository
Models/Entities/	User, Company, JobApplication, InterviewRound, ApplicationStatus (enum), InterviewType (enum), InterviewResult (enum)
Models/DTOs/Auth/	RegisterRequestDto, LoginRequestDto, AuthResponseDto, UserDto
Models/DTOs/Company/	CreateCompanyDto, UpdateCompanyDto, CompanyResponseDto
Models/DTOs/JobApplication/	CreateJobApplicationDto, UpdateJobApplicationDto, JobApplicationResponseDto, JobApplicationQueryParams, PagedResult<T>
Models/DTOs/Interview/	CreateInterviewRoundDto, UpdateInterviewRoundDto, InterviewRoundResponseDto
Data/	ApplicationDbContext (EF Core DbContext with seed data)
Mapping/	MappingProfile (AutoMapper configuration)
Middleware/	GlobalExceptionMiddleware
Extensions/	JwtSettings, ClaimsExtensions
Migrations/	EF Core auto-generated migration files
wwwroot/swagger-ui/	custom.css, custom.js (Swagger UI customization)
Program.cs	Application entry point, DI registration, middleware pipeline
appsettings.json	Configuration: connection string, JWT settings, logging
Dockerfile	Multi-stage Docker build for containerized deployment

4.2 Test Project: JobTrackerAPI.Tests

File	Contents
AuthServiceTests.cs	4 unit tests covering registration, login, duplicate email, and wrong password scenarios
JobTrackerAPI.Tests.csproj	Test project config with xUnit, Moq, FluentAssertions packages

5. Database Design

The database uses SQLite managed through Entity Framework Core with code-first migrations. All relationships are enforced at the database level with foreign key constraints.

5.1 Entity Relationship Summary

Relationship	Type	On Delete
User -> JobApplications	One-to-Many	Cascade (delete user = delete all their applications)
Company -> JobApplications	One-to-Many	Set Null (delete company = keep applications, CompanyId = null)
JobApplication -> InterviewRounds	One-to-Many	Cascade (delete application = delete all its interviews)

5.2 Entities and Fields

User

Field	Type	Notes
Id	int	Primary key, auto-increment
FirstName	string (100)	Required
LastName	string (100)	Required
Email	string (255)	Required, unique index
PasswordHash	string	BCrypt hash, cost factor 11
Role	string (50)	Default: 'User'. Options: 'User', 'Admin'
CreatedAt / UpdatedAt	DateTime	UTC timestamps

Company

Field	Type	Notes
Id	int	Primary key, auto-increment
Name	string (200)	Required
Website	string (500)	Optional
Industry	string (100)	Optional
Location	string (200)	Optional
Notes	string	Optional, free text

Field	Type	Notes
CreatedAt / UpdatedAt	DateTime	UTC timestamps

JobApplication

Field	Type	Notes
Id	int	Primary key, auto-increment
JobTitle	string (200)	Required
CompanyName	string (200)	Required, denormalized for display
JobLocation	string (200)	Optional
JobUrl	string (1000)	Optional link to job posting
ApplicationDate	DateTime	Required
Status	enum (string)	Applied, Interviewing, Offer, Rejected, Withdrawn, Ghosted
Notes	string (5000)	Optional
UserId	int (FK)	Foreign key to User
CompanyId	int? (FK)	Nullable foreign key to Company
CreatedAt / UpdatedAt	DateTime	UTC timestamps

InterviewRound

Field	Type	Notes
Id	int	Primary key, auto-increment
InterviewDate	DateTime	Required
InterviewType	enum (string)	HR, Technical, Managerial, Cultural, Final, Other
Result	enum (string)	Pending, Passed, Failed, Cancelled
Interviewer	string (200)	Optional, interviewer name
Feedback	string (3000)	Optional
Notes	string (3000)	Optional
JobApplicationId	int (FK)	Foreign key to JobApplication
CreatedAt / UpdatedAt	DateTime	UTC timestamps

5.3 Seed Data

The following data is automatically inserted on first startup:

- Admin user: admin@jobtracker.com / Admin@12345 (Role: Admin)

- Company: Acme Corp (Technology, San Francisco, CA)
- Company: Tech Solutions Inc. (Software, New York, NY)

6. API Endpoints

6.1 Authentication Endpoints

Method	Route	Auth Required	Description
POST	/api/auth/register	No	Register a new user account
POST	/api/auth/login	No	Login and receive JWT token

6.2 Job Application Endpoints

Method	Route	Auth	Description
GET	/api/jobapplications	User	Get all applications (paginated, filtered)
GET	/api/jobapplications/{id}	User	Get single application by ID
POST	/api/jobapplications	User	Create new job application
PATCH	/api/jobapplications/{id}	User	Partially update application
DELETE	/api/jobapplications/{id}	User	Delete application

6.3 Company Endpoints

Method	Route	Auth	Description
GET	/api/companies	User	Get all companies (with optional search)
GET	/api/companies/{id}	User	Get single company by ID
POST	/api/companies	Admin Only	Create new company
PATCH	/api/companies/{id}	Admin Only	Update company details
DELETE	/api/companies/{id}	Admin Only	Delete a company

6.4 Interview Round Endpoints

Method	Route	Auth	Description
GET	/api/jobapplications/{appId}/interviews	User	Get all interview rounds for an application
GET	/api/jobapplications/{appId}/interviews/{id}	User	Get single interview round
POST	/api/jobapplications/{appId}/interviews	User	Add new interview round
PATCH	/api/jobapplications/{appId}/interviews/{id}	User	Update interview round

Method	Route	Auth	Description
DELETE	/api/jobapplications/{appId}/interviews/{id}	User	Delete interview round

6.5 System Endpoints

Method	Route	Auth	Description
GET	/health	No	Health check - returns API status
GET	/	No	Swagger UI - interactive API documentation

6.6 Query Parameters for Job Applications

The GET /api/jobapplications endpoint supports the following query parameters:

Parameter	Type	Default	Description
page	int	1	Page number for pagination
pageSize	int	10	Number of items per page (max 100)
status	string	null	Filter by status: Applied, Interviewing, Offer, Rejected, Withdrawn, Ghosted
searchTerm	string	null	Search in job title, company name, and location
sortBy	string	ApplicationDate	Sort field: ApplicationDate, JobTitle, CompanyName, Status
sortDescending	bool	true	Sort direction

7. Security Implementation

7.1 JWT Authentication

The API uses JSON Web Tokens (JWT) with HMAC-SHA256 signing for stateless authentication. Tokens are issued on login and must be included in the Authorization header for all protected endpoints.

Setting	Value
Algorithm	HMAC-SHA256 (HS256)
Token Expiry	60 minutes (configurable)
Clock Skew	Zero (strict expiry enforcement)
Claims Included	User ID (sub), Email, First Name, Last Name, Role
Header Format	Authorization: Bearer {token}

7.2 Password Security

- Passwords are hashed using BCrypt with a cost factor of 11
- BCrypt uses adaptive hashing - cost factor can be increased over time
- Plain text passwords are never stored or logged anywhere
- BCrypt.Verify() uses constant-time comparison to prevent timing attacks

7.3 Authorization Model

Role	Access Level	Endpoints
Anonymous	Public	/api/auth/register, /api/auth/login, /health
User	Authenticated	All job application and interview endpoints (own data only)
Admin	Elevated	All User access plus create/update/delete companies

7.4 Data Isolation

All job application and interview queries are automatically scoped to the authenticated user's ID extracted from the JWT token. It is architecturally impossible for one user to read or modify another user's data, as the UserId filter is applied at the repository level on every query.

8. Key Features & Business Logic

8.1 Auto Status Promotion

When a user adds an interview round to a job application that is still in 'Applied' status, the system automatically promotes the application status to 'Interviewing'. This reduces manual data entry and keeps the application status accurate.

8.2 PATCH Semantics

All update endpoints (PATCH) only modify the fields that are explicitly provided in the request body. Fields that are omitted (null) retain their existing values. This allows partial updates without requiring the client to send the full resource.

8.3 Pagination Response

All list endpoints return a structured pagination envelope with the following metadata:

Field	Type	Description
items	array	The current page of results
totalCount	int	Total number of matching records
page	int	Current page number
pageSize	int	Number of items per page
totalPages	int	Calculated total number of pages
hasPreviousPage	bool	Whether a previous page exists
hasNextPage	bool	Whether a next page exists

8.4 Global Exception Handling

A custom middleware intercepts all unhandled exceptions and returns structured JSON error responses with appropriate HTTP status codes. This prevents stack traces from leaking to clients in production.

Exception Type	HTTP Status	When It Occurs
KeyNotFoundException	404 Not Found	Requested resource does not exist
UnauthorizedAccessException	401 Unauthorized	Invalid credentials or token
InvalidOperationException	409 Conflict	Business rule violation (e.g. duplicate email)
ArgumentException	400 Bad Request	Invalid input arguments
Any other Exception	500 Internal Server Error	Unexpected system errors

9. NuGet Package Dependencies

Package	Version	Purpose
AutoMapper	12.0.1	Object-to-object mapping between entities and DTOs
AutoMapper.Extensions.Microsoft.DependencyInjection	12.0.1	AutoMapper DI integration for ASP.NET Core
BCrypt.Net-Next	4.0.3	Industry-standard password hashing
FluentValidation.AspNetCore	11.3.0	Advanced input validation framework
Microsoft.AspNetCore.Authentication.JwtBearer	8.0.0	JWT token validation middleware
Microsoft.AspNetCore.OpenApi	8.0.0	OpenAPI support for ASP.NET Core
Microsoft.EntityFrameworkCore	8.0.0	ORM core library
Microsoft.EntityFrameworkCore.Sqlite	8.0.0	SQLite database provider for EF Core
Microsoft.EntityFrameworkCore.Tools	8.0.0	EF Core CLI tools (migrations, scaffolding)
Microsoft.IdentityModel.Tokens	7.3.1	JWT token creation and validation primitives
Swashbuckle.AspNetCore	6.6.2	Swagger UI and OpenAPI document generation
System.IdentityModel.Tokens.Jwt	7.3.1	JWT token handler and serialization

10. Deployment

10.1 Docker Configuration

The application uses a multi-stage Docker build to minimize the final image size. The build stage compiles the application using the full .NET SDK, and the runtime stage uses only the smaller ASP.NET runtime image.

Stage	Base Image	Purpose
Build	mcr.microsoft.com/dotnet/sdk:8.0	Compile and publish the application
Runtime	mcr.microsoft.com/dotnet/aspnet:8.0	Run the compiled application

10.2 Render Cloud Deployment

Setting	Value
Platform	Render (render.com)
Runtime	Docker
Instance Type	Free Tier
Region	Singapore
Auto-Deploy	Yes - triggers on every GitHub push to main branch
Live URL	https://job-application-tracker-api-b9t2.onrender.com
Database	SQLite stored in /tmp/ directory

10.3 Environment Variables

Variable	Description
ASPNETCORE_ENVIRONMENT	Set to 'Production' on Render
JwtSettings__SecretKey	Secret key for signing JWT tokens (min 32 chars)
JwtSettings__Issuer	JWT issuer identifier
JwtSettings__Audience	JWT audience identifier

11. Testing

11.1 Unit Tests

The test project (JobTrackerAPI.Tests) contains unit tests for the AuthService using xUnit, Moq, and FluentAssertions.

Test Name	Scenario	Expected Result
RegisterAsync_WithValidData_ReturnsAuthResponse	Valid registration request	Returns JWT token and user details
RegisterAsync_WithDuplicateEmail_ThrowsInvalidOperationException	Email already exists in system	Throws InvalidOperationException
LoginAsync_WithInvalidPassword_ThrowsUnauthorizedAccessException	Wrong password provided	Throws UnauthorizedAccessException
LoginAsync_WithValidCredentials_ReturnsAuthResponse	Correct email and password	Returns valid JWT token

11.2 Testing Tools

Tool	Version	Purpose
xUnit	2.9.0	Test framework and test runner
Moq	4.20.72	Mocking framework for dependencies
FluentAssertions	6.12.1	Readable assertion syntax
Microsoft.NET.Test.Sdk	17.11.1	Test infrastructure for .NET

11.3 Running Tests

To run all unit tests, navigate to the JobTrackerAPI.Tests folder and execute:

```
dotnet test
```

12. Local Development Setup

12.1 Prerequisites

- .NET 8 SDK (https://dotnet.microsoft.com/download/dotnet/8.0)
- Visual Studio 2022, VS Code with C# extension, or JetBrains Rider
- Git for version control

12.2 Setup Steps

Step	Command / Action	Description
1	git clone <repository-url>	Clone the repository
2	cd JobTrackerAPI	Navigate to project folder
3	dotnet restore	Install all NuGet packages
4	dotnet ef database update	Create SQLite database and run migrations
5	$env:ASPNETCORE_ENVIRONMENT='Development'; dotnet run	Start API in development mode
6	Open http://localhost:5000	Access Swagger UI

12.3 Default Credentials

Account	Email	Password	Role
Admin	admin@jobtracker.com	Admin@12345	Admin
Test User	Register via /api/auth/register	Any (min 8 chars)	User

13. Challenges & Solutions

Challenge	Root Cause	Solution Applied
Build errors on test files	Test project included in main project compilation	Created separate JobTrackerAPI.Tests project; excluded Tests/ folder from main .csproj using <Compile Remove>
AutoMapper version mismatch warning	AutoMapper 13.x used with DI extension requiring 12.x	Downgraded AutoMapper to 12.0.1 to match extension package requirement
SQLite file not found on Render	Container filesystem is read-only by default in working directory	Changed connection string to use /tmp/JobTracker.db which is always writable
Seeded admin password invalid	BCrypt.HashPassword() called at EF model-building time produces inconsistent hashes	Replaced with a pre-computed BCrypt hash in the seed data
Swagger not showing in production	Swagger UI was gated behind IsDevelopment() check	Removed the environment check to enable Swagger in all environments
Migration conflicts	Hand-written migration file conflicted with auto-generated one	Deleted all migration files and regenerated using dotnet ef migrations add

14. Future Improvements

- Switch from SQLite to PostgreSQL or Azure SQL for persistent cloud storage
- Add refresh token support for longer user sessions
- Implement email notifications for interview reminders
- Add a statistics/dashboard endpoint (e.g. applications by status, success rate)
- Add rate limiting to prevent API abuse
- Implement soft deletes instead of hard deletes for data recovery
- Add integration tests using WebApplicationFactory
- Add GitHub Actions CI/CD pipeline for automated testing and deployment
- Add resume/document upload support per application
- Implement response caching for company listings

15. Summary

The Job Application Tracker API is a complete, production-deployed REST API that demonstrates professional software engineering practices. It was built from scratch using ASP.NET Core 8 with a clean layered architecture, secure JWT authentication, Entity Framework Core for data access, and a full Swagger UI for interactive documentation.

The project is live on Render and accessible publicly. It showcases real-world patterns including the Repository Pattern, Dependency Injection, DTO mapping with AutoMapper, global exception handling middleware, role-based authorization, and paginated/filtered query support.

Metric	Value
Total Files	48+ files across controllers, services, repositories, models, DTOs, and infrastructure
API Endpoints	17 RESTful endpoints across 5 controllers
Database Tables	4 tables with proper relationships and indexes
Unit Tests	4 unit tests covering core authentication scenarios
NuGet Packages	12 packages
Architecture	4-layer clean architecture with Repository Pattern
Deployment	Live on Render via Docker container
Authentication	JWT Bearer with BCrypt password hashing

Developed by Lokpavan P | February 2026