Norvia Backoffice

Phase 1 - Project Setup, Authentication & Application Foundation

Implementation-ready prompt for Codex / Claude Code | Client: Norvia Labs

1. Project Overview

Create a new ASP.NET Core .NET 10 web application named Norvia Backoffice in the currently selected folder. Norvia Labs operates multiple enterprise-grade, multi-tenant SaaS products for client organizations. Most products connect to a shared Azure SQL Database. The Backoffice will centralize management of Tenants, Tenant Administrators, SaaS Products, Product-specific Roles, and Backoffice Users / Administrators.

Backoffice roles: Super Admin, Admin, User. For this phase implement only the application foundation, authentication, dashboard, password management and profile functionality. Do not build the business management modules yet; structure the solution for their future addition.

2. Technology Stack

• ASP.NET Core MVC on .NET 10, C#, Razor Views, Bootstrap 5.

• Entity Framework Core 10 with existing Azure SQL Database.

• ASP.NET Core Cookie Authentication and PasswordHasher<TUser>.

• Gmail SMTP for transactional password reset email.

• Azure Virtual Machine running Ubuntu Linux as the future hosting target.

• VS Code development on Windows and macOS. Keep a simple layered MVC architecture; avoid unnecessary frameworks and dependencies.

3. Project Creation and Structure

Create the complete project in the selected folder. Organize controllers, entity and view models, EF Core DbContext, service interfaces/implementations, Razor views, shared layout, static assets, configuration, and README. Use dependency injection and async data access. Avoid microservices, CQRS, MediatR, React, Angular, or Blazor.

```
NorviaBackoffice/
  Controllers/ (Account, Home, Profile)
  Models/Entities/  Models/ViewModels/
  Data/NorviaDbContext.cs
  Services/Interfaces/  Services/Implementations/
  Views/Account/  Views/Home/  Views/Profile/  Views/Shared/
  wwwroot/css/  wwwroot/js/  wwwroot/lib/
  appsettings.json  appsettings.Development.json
  appsettings.Example.json  Program.cs
  NorviaBackoffice.csproj  README.md
```

4. Azure SQL Database and Configuration

Connect directly to an existing Azure SQL development database shared across local developer machines. Each developer authenticates with their own SQL username/password. Do not create a local database. Supply an example configuration with placeholders for Azure SQL server, database, SQL user/password, Gmail address/App Password, and application base URL.

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=tcp:<SERVER>.database.windows.net,1433;Initial Catalog=<DB>;User ID=<SQL_USER>;Password=<SQL
  },
  "EmailSettings": {
```

```
    "SmtpHost": "smtp.gmail.com",
    "SmtpPort": 587,
    "Username": "<GMAIL_ADDRESS>",
    "AppPassword": "<GMAIL_APP_PASSWORD>",
    "FromEmail": "<GMAIL_ADDRESS>",
    "FromName": "Norvia Backoffice"
  },
  "ApplicationSettings": {"BaseUrl": "https://<APPLICATION_URL>"}
}
```

• Commit placeholders only; never commit secrets or hardcode them in C# files.

• Use .NET User Secrets or environment variables for local developer credentials.

• Document configuration for each developer and report missing essential settings clearly.

5. Existing Database Schema

Use the existing dbo.NorviaUsers table: Id, Name, Email, PasswordHash, RoleId, IsActive, MustChangePassword, FailedLoginAttempts, LastLoginOn, CreatedOn, UpdatedOn. Use dbo.NorviaRoles: Id, Name, Description, IsActive. Roles: 1 = Super Admin, 2 = Admin, 3 = User.

• Map exact existing SQL table and column names using EF Core.

• Do not create or modify tables, run migrations, or insert sample/default users.

• Use the existing administrator account in NorviaUsers; use parameterized EF Core queries and AsNoTracking for read-only access.

6. Login Functionality

Build a modern responsive Bootstrap 5 login page with Norvia Backoffice branding, Email, Password, show/hide password, Remember Me, Login, and Forgot Password.

• Validate input; load user by email, require IsActive = 1 and an active role.

• Verify passwords with ASP.NET Core PasswordHasher<TUser>; do not compare password hashes as strings.

• Create secure cookie authentication and redirect to Dashboard. Include claims: User ID, Name, Email, Role ID, Role Name.

• Apply anti-forgery validation, secure/HttpOnly/SameSite cookies, production HTTPS, session expiration, and generic credential errors.

• Track FailedLoginAttempts; reset on successful login and update LastLoginOn.

• Implement configurable temporary lockout without changing the database schema; document limitations of the chosen server-side mechanism in multi-instance deployment.

• If MustChangePassword = 1, require password change before other protected pages are accessible.

7. Forgot Password

Create a Forgot Password page with email field, submit button and back-to-login link. If the email matches an active account, generate a cryptographically secure, single-use password reset token valid for 30 minutes. Email a reset link through Gmail SMTP (smtp.gmail.com, port 587, TLS, Gmail App Password read from configuration). Do not expose whether an email is registered.

Display: "If an account exists with this email address, you will receive password reset instructions shortly." Do not store reset tokens in plaintext. Use a server-side store supporting expiration and one-time consumption; do not use in-memory-only storage for production. Because database schema changes are forbidden, explain any required persistent-storage dependency and propose the smallest safe solution before implementing it.

8. Reset / Change Password

Provide a Reset Password page for valid emailed tokens and a Change Password page for signed-in users with Current Password, New Password, Confirm Password and Save. Validate current password for authenticated changes, confirm new password, hash with PasswordHasher<TUser>, update PasswordHash and UpdatedOn, and set MustChangePassword = 0. Enforce at least 12 characters, reject common passwords, and reject reuse of the current password. Invalidate prior sessions and require a fresh login after password changes.

9. Dashboard

After login, show a simple dashboard with "Welcome, [User Name]", the user role, and: "Welcome to Norvia Backoffice. Use the navigation menu to manage the Norvia Labs SaaS ecosystem." Do not add business charts or tenant statistics in this phase.

10. Shared Application Layout

Use a responsive Bootstrap 5 TOP navigation bar, not a sidebar. Left: Norvia Backoffice branding. Right: signed-in user name, role and user dropdown. Prepare menu labels: Dashboard, Tenants, Tenant Admins, Products, Product Roles, Backoffice Users. Only Dashboard is functional; label others as upcoming without fake working screens.

User dropdown: My Profile, Change Password, Logout. Footer: "Norvia Backoffice | Powered by Norvia Labs". Ensure responsive desktop, tablet, and mobile behavior.

11. My Profile

Display Name, Email, Role, Account Status and Last Login. Only Name is editable in Phase 1; Email and Role are read-only. Validate and save Name to NorviaUsers, update UpdatedOn, refresh the header display name and show a success message. Never allow self-service role or status changes.

12. Logout

Use an anti-forgery-protected POST logout endpoint. Sign out of cookie authentication, redirect to Login, and ensure protected pages require a new login.

13. Role-Based Authorization

Use NorviaRoles: Super Admin, Admin, User. All active roles may access Dashboard, view/update their own profile, change password and log out. Prepare policy/role authorization for future modules. Enforce authorization server-side, not only through hidden menu items.

14. UI Requirements

• Bootstrap 5 light theme; professional, modern enterprise appearance.

• Responsive top navigation, consistent typography and spacing, accessible forms, field validation, success/error notifications and appropriate loading indicators.

• Use Razor Views and Bootstrap components; avoid excessive animation and unnecessary JavaScript.

15. Azure VM Compatibility

Target later deployment to Azure Ubuntu Linux VM, but do not deploy yet. Use cross-platform .NET APIs, environment-based configuration, relative file paths, no IIS dependency, no hardcoded localhost URLs/IPs, and support HTTPS behind a reverse proxy.

16. Error Handling and Logging

Use centralized exception handling and ASP.NET Core built-in logging. Provide user-friendly errors without stack traces; do not log passwords or credentials. Handle database and SMTP errors gracefully. No external logging platform is required.

17. Testing and Verification

• Build and launch successfully; login page renders with Bootstrap styles.

• Connect to Azure SQL and read NorviaUsers / NorviaRoles without schema changes.

• Valid login succeeds; invalid or inactive accounts fail; claims, role display, Remember Me and protected-page authorization work.

• Forgot Password sends mail; reset tokens expire and cannot be reused; passwords are hashed.

• Change Password and profile update work; user cannot change own role; header refreshes name.

• Logout ends authentication and protected pages cannot be accessed without login.

18. Documentation

Create README.md with project overview, software prerequisites, Azure SQL configuration, Gmail App Password setup, local execution on Windows/macOS, DB connectivity verification, build/test commands, folder structure, and security notes.

19. Implementation Instructions

• Create the project in the selected folder using .NET 10 MVC and Bootstrap 5.

• Use the existing Azure SQL schema; never auto-create/alter tables or seed users.

• Implement login, forgot/reset password, change password, dashboard, profile and logout.

• Use secure configuration placeholders; ensure Ubuntu VM compatibility.

• Build and fix errors; add automated tests for critical authentication/password flows.

• If safe implementation requires additional persistent storage or a schema change, explain the smallest change and obtain direction before modifying the database.

20. Deliverables and Acceptance Criteria

Deliver a working .NET 10 MVC project with Login, Forgot Password, Reset Password, Dashboard, responsive top menu, My Profile, Change Password, Logout, Azure SQL integration, role-based authorization foundation, example configuration, README and automated tests. The application must run locally on Windows/macOS, use the existing Azure SQL dev database, display the logged-in name/role, protect authenticated pages, and build/test successfully without unintended database modifications.

Security Note

The development administrator password 123456 is weak and must be replaced before public deployment. Verify that any pre-generated initial password hash matches the intended password with ASP.NET Core PasswordHasher<TUser>.