Software Design Document for project BankNumerator

Prepared by: Serhat Özdemir
Date:

August 26, 2025

Area / Feature	Previous Version	Current Version
Agent Tickets (UI & API)	Page included a *Release* button; lifecycle supported a Release action.	*Release* action removed; agents proceed with Accept / Reject / Route only.
Admin Dashboard (Analytics)	No charts/metrics dashboard.	Added analytics with `Chart.js`: per-service ticket counts; per-agent accepted/rejected/pending; overall totals (issued, pending, rejected).
Profile Page	Not available.	New Profile page added.
User Priority (Admin)	Admin could not assign a user/global priority.	Admin can assign/update a *PriorityScore* for users (impacts ticket ordering).
Service Priority	Services had no priority value.	Admin can assign per-service *priority* (used in sequencing and views).
Priority Aging (Pending)	Pending tickets had static priority.	Automatic *priority aging*: priority of pending tickets increases over time.

Table 1: Version Differences Summary

1 Introduction

1.1 Purpose

This Software Design Document (SDD) describes the architecture, components, and design decisions for the BankNumerator system, a queue management application built with Angular 20 and .NET 9. It is intended for developers, architects, QA engineers, and project stakeholders to ensure a shared understanding of the system design.

1.2 Scope

The BankNumerator application allows default users to obtain and cancel numbered tickets for various banking services, while admin users can manage services, view all tickets, enforce per-service limits, and cancel tickets. Agents with specialized skills handle ticket processing and can route tickets among themselves. Ticket issuance, cancellation, and all interactions are persisted in PostgreSQL.

1.3 Overview

This document is organized as follows:

- Section 2: System Overview – high-level functionality and context
- Section 3: System Architecture – decomposition of backend and frontend
- Section 4: Data Design – database schemas and data dictionary
- Section 5: Component Design – backend controllers and frontend components/services
- Section 6: Human Interface Design – user workflows and screen layouts
- Section 7: Requirements Matrix – mapping requirements to modules
- Section 8: Appendices – supplemental diagrams and references

1.4 Definitions and Acronyms

Term	**Definition**
SDD	Software Design Document
API	Application Programming Interface
JWT	JSON Web Token
SPA	Single Page Application
Admin	User role with full system privileges
Default User	End user with limited privileges
Agent	Specialized service representative (separate entity)
AgentSkill	Mapping between Agent and ServiceKey

Table 2: Definitions and Acronyms

2 System Overview

BankNumerator is a **three-tier** system:

- **Client Tier:** Angular 20 SPA (components, services, guards, interceptor)
- **Server Tier:** ASP.NET Core 9 Web API (controllers, business logic, data access)
- **Data Tier:** PostgreSQL database

3 System Architecture

3.1 Client Tier (Angular 20 SPA)

- **Components:**
  - LoginComponent (`src/app/components/login`)
  - SignupComponent (`src/app/components/signup`)
  - NumeratorComponent (`src/app/components/numerator`)
  - NavbarComponent (`src/app/components/navbar`)
  - AdminDashboardComponent (`src/app/components/admin-dashboard`)
  - AdminServiceManagementComponent (`src/app/components/admin-service-manageme`
  - AdminAgentsComponent (`src/app/components/admin-agents`)
  - AdminTicketsComponent (`src/app/components/admin-tickets`)
  - AdminSidebarComponent (`src/app/components/admin-sidebar`)
  - AdminSignupComponent (`src/app/components/admin-signup`)
  - AgentTicketsComponent (`src/app/components/agent-tickets`)
- **Services:**
  - AuthService (`auth.service.ts`)
  - AdminService (`admin.service.ts`)
  - AdminAgentsService (`admin-agents.service.ts`)
  - AgentService (`agent.service.ts`)
  - QueueService (`queue.service.ts`)
  - BankService (`bank.services.ts`)
- **Utilities:** authGuard/authMatchGuard, AuthInterceptor

3.2 Server Tier (ASP.NET Core 9 Web API)

- **Program.cs / Startup:** DI container, middleware, JWT setup
- **Controllers:**
  - AuthController (`/api/auth`)
  - ServicesController (`/api/services`)
  - NumeratorController (`/api/numerator`)
  - AdminController (`/api/admin`)
  - AgentTicketsController (`/api/agent/tickets`)
  - AdminAgentsController (`/api/admin/agents`)
- **Business Logic (Domain Services):**
  - AuthService (signup/login)

  - QueueService (issue/cancel tickets)
  - AdminService (service CRUD, ticket oversight)
  - AgentService (assign/release/route tickets)

- **Data Access:**
  - BankNumeratorContext (EF Core DbContext)
  - Entity models, migrations via postgreSQL

3.3 Data Tier (PostgreSQL)

- Tables: Users, Agents, Services, Tickets, AgentSkills, TicketAssignments
- All ORM interactions via Entity Framework Core

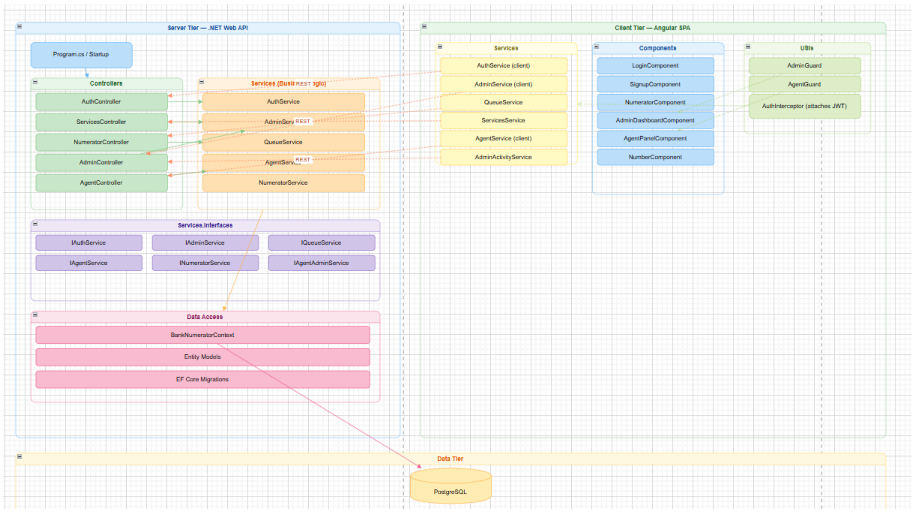

Figure 1: Tiered Architecture of BankNumerator

3.4 Backend API Controllers

Controller	**Route**	**Auth**	**Actions**
AuthController	/api/auth	Anonymous	POST /signup, POST /login
ServicesController	/api/services	Anonymous	GET /
NumeratorController	/api/numerator	`[Authorize]`	GET /next, DELETE /{ticketId}
AdminController	/api/admin	`[Authorize(Roles="Admin")]`	POST /services, PUT /services/{key}/deactivate, PUT /services/{key}/limit, GET /tickets, DELETE /tickets/{id}
AdminAgentsController	/api/admin/agents	`[Authorize(Roles="Admin")]`	GET /, POST /, DELETE /
AgentTicketsController	/api/agent/tickets	`[Authorize(Roles="Agent")]`	GET /, POST /accept, POST /reject, POST /release, POST /route, GET /route-candidates/{serviceKey}
TestController (Debug only)	/api/test	Anonymous	POST /reset

Table 3: Backend API Controllers

3.5 Angular Frontend Structure

Components

- `LoginComponent` (src/app/components/login)
- `SignupComponent` (src/app/components/signup)
- `NumeratorComponent` (src/app/components/numerator)

- `AdminDashboardComponent` (src/app/components/admin-dashboard)
- `AdminServiceManagementComponent` (src/app/components/admin-service-management)
- `AdminAgentsComponent` (src/app/components/admin-agents)
- `AdminTicketsComponent` (src/app/components/admin-tickets)
- `AgentTicketsComponent` (src/app/components/agent-tickets)
- `NavbarComponent` (src/app/components/navbar)

Services

- `AuthService` - signup, login, logout, token storage
- `ServicesService` - fetch active services
- `QueueService` - getNext, cancel ticket
- `AdminService` - manage services and tickets
- `AgentService` - fetch skills, assign/release/route tickets

Guards and Interceptors

- `authGuard` / `authMatchGuard` - protect routes
- `authInterceptor` - attach `Authorization` header

4 Data Design

4.1 Data Description

All domain entities are stored in PostgreSQL using the following tables:

- `Users` (Id, Username, Email, PasswordHash, PasswordSalt, Role, PriorityScore)
- `Agents` (Id, UserId, Skills, Assignments)
- `AgentSkills` (AgentId, ServiceKey)
- `ServiceItems` (Id, Key, Label, IsActive, MaxNumber)
- `ServiceCounters` (ServiceKey, CurrentNumber)
- `Tickets` (Id, Number, ServiceKey, UserId, AgentId?, Status, TakenAt, CancelledAt, PriorityAtIssue)
- `TicketAssignments` (TicketId, AgentId, AssignedAt, Status)

4.2 Data Dictionary

Entity	Field	Description
Users	Id (int, PK)	Unique user identifier
	Username (string)	Login/display name
	Email (string)	Unique email address
	PasswordHash (byte[])	Hashed password
	PasswordSalt (byte[])	Salt for hashing
	Role (enum)	{Default, Admin, Agent}
	PriorityScore (int)	Dynamic priority ranking
Agents	Id (int, PK)	Unique agent identifier
	UserId (int, FK)	References Users.Id
AgentSkills	AgentId (int, FK)	References Agents.Id
	ServiceKey (string, FK)	References ServiceItems.Key
ServiceItems	Id (int, PK)	Unique service identifier
	Key (string)	Service code (e.g., ”withdrawal”)
	Label (string)	Human-readable name
	IsActive (bool)	Service availability flag
	MaxNumber (int)	Daily ticket limit
ServiceCounters	ServiceKey (string, PK)	References ServiceItems.Key
	CurrentNumber (int)	Last issued ticket number
Tickets	Id (int, PK)	Unique ticket identifier
	Number (int)	Sequential ticket number
	ServiceKey (string, FK)	References ServiceItems.Key
	UserId (int, FK)	References Users.Id
	AgentId (int, FK, null)	References Agents.Id
	Status (enum)	{Issued, Cancelled, Completed}
	TakenAt (datetime)	Ticket creation time
	CancelledAt (datetime)	Cancellation time, if any
	PriorityAtIssue (int)	Priority score at time of issue
TicketAssignments	TicketId (int, FK)	References Tickets.Id
	AgentId (int, FK)	References Agents.Id
	AssignedAt (datetime)	Assignment timestamp
	Status (string)	{Pending, Accepted, Released}

Table 4: Data Dictionary

5 Component Design

5.1 Backend Controllers

AuthController

```
POST /signup:
  if missing fields -> BadRequest
  validate email format
  create password hash/salt
  save User (Role=Default)
  return { Id, Username, Email }

POST /login:
  find user by email
  verify password
  create JWT
  return { token }
```

NumeratorController

```
GET /next?service={key}:
  check ServiceItems.IsActive
  determine next Number based on priority
  insert Ticket with UserId, ServiceKey
  return { number }

DELETE /{ticketId}:
  authorize Default if owns ticket or Admin
  update Ticket.Status=Cancelled, CancelledAt=now
  return Ok
```

ServicesController

```
GET /api/services:
  return all active ServiceDto(Key, Label)
```

AdminController

```
POST /services: create new ServiceItem
PUT /services/{key}/deactivate: set IsActive=false
PUT /services/{key}/limit: set MaxNumber
GET /tickets: return filtered Ticket list
DELETE /tickets/{id}: cancel any ticket
```

AdminAgentsController

```
GET /admin/agents: list all agents
POST /admin/agents: create new agent
DELETE /admin/agents/{id}: remove agent
```

AgentTicketsController

```
GET /agent/tickets: list tickets assigned to agent
POST /{ticketId}/accept: set Status=Accepted
POST /{ticketId}/reject: reassign to another agent
POST /{ticketId}/release: remove assignment & ticket
POST /{ticketId}/route/{toAgentId}: assign to another agent
GET /route-candidates/{serviceKey}: list eligible agents
```

TestController (Debug only)

```
POST /reset: clear database for test automation
```

5.2 Frontend Components

LoginComponent

```
Inputs: email, password
On submit: AuthService.login()
On success: navigate to /numerator
```

SignupComponent

```
Inputs: username, email, password
On submit: AuthService.signup()
On success: navigate to /login
```

NumeratorComponent

```
ngOnInit(): fetch services via ServicesService
selectService(): set key, step++
assignNumber(): QueueService.getNext()
cancel(): QueueService.cancel()
```

AdminDashboardComponent

```
Contains sidebar and router-outlet for admin pages
```

AdminServiceManagementComponent

```
fetch services via AdminService
enable/disable service
update daily limits
```

AdminAgentsComponent

```
list agents via AdminAgentsService
add/remove agents
```

AdminTicketsComponent

```
fetch tickets via AdminService
cancel or manage tickets
```

AgentTicketsComponent

```
fetch assigned tickets via AgentService
accept/reject/release/route actions
```

NavbarComponent

```
logout: AuthService.logout()
theme switch
```

5.3 Services, Guards, Interceptor

AuthService : signup, login, logout, token storage

AdminService : CRUD for services and tickets

AdminAgentsService : CRUD for agents

AgentService : getSkills(), getTickets(), assign(), release(), route()

QueueService : getNext(), cancel(ticketId)

ServicesService : getAll()

authGuard / authMatchGuard : protect routes based on AuthService.isLoggedIn()

authInterceptor : attach `Authorization: Bearer <token>` header

6 Human Interface Design

6.1 User Workflows

1. **Default User**:
   (a) Signup
   (b) Login
   (c) Select service from available list
   (d) Receive a ticket number
   (e) Optionally cancel the ticket before it is processed
2. **Admin User**:
   (a) Login
   (b) Access Admin Dashboard
   (c) In *Service Management*:
      - Add or remove services
   (d) In *Agent Management*:
      - Create new Agent users
      - View all existing Agents
   (e) In *Admin Management*:

        - Create other Admin users

    (f) Take a ticket (same as Default user workflow)

    (g) In *Ticket Oversight*:
        - View all issued tickets
        - Cancel any ticket

3. **Agent User**:

    (a) Login

    (b) Access Agent Panel

    (c) View skills (service keys) assigned to this agent

    (d) See assigned tickets (ordered by priority score if applicable)

    (e) Actions on tickets:
        - Accept (Status = Accepted)
        - Reject (remove assignment, reassign to eligible agent)
        - Release (remove ticket entirely and decrement counter)
        - Route (reassign to another agent with the same skill)

6.2 Screen Mockups

Figure 2: Sample Admin Dashboard

6.3 Screen Objects and Actions

- Buttons: Assign, Assign,Save, Cancel
- Forms: Login, Signup, Service Limit
- Tables: Service list, Ticket list, Agent assignments

7 Requirements Matrix

Req. ID	**Requirement**	**Module**
R1	Ticket issuance and persistence	NumeratorController, QueueService
R2	Role-based access control	AuthController, authGuard
R3	Service CRUD (Admin)	AdminController, AdminService
R4	Ticket cancellation (Default/Admin)	NumeratorController
R5	Priority-based sequencing	NumeratorController
R6	Agent skill-based ticket assignment	AgentController, AgentService
R7	Agent ticket routing	AgentController, AgentService
R8	Admin dashboard with stats	AdminDashboardComponent
R9	Frontend guards and interceptor	authGuard, authInterceptor

Table 5: Traceability Matrix

8 Appendices

8.1 ER Diagram

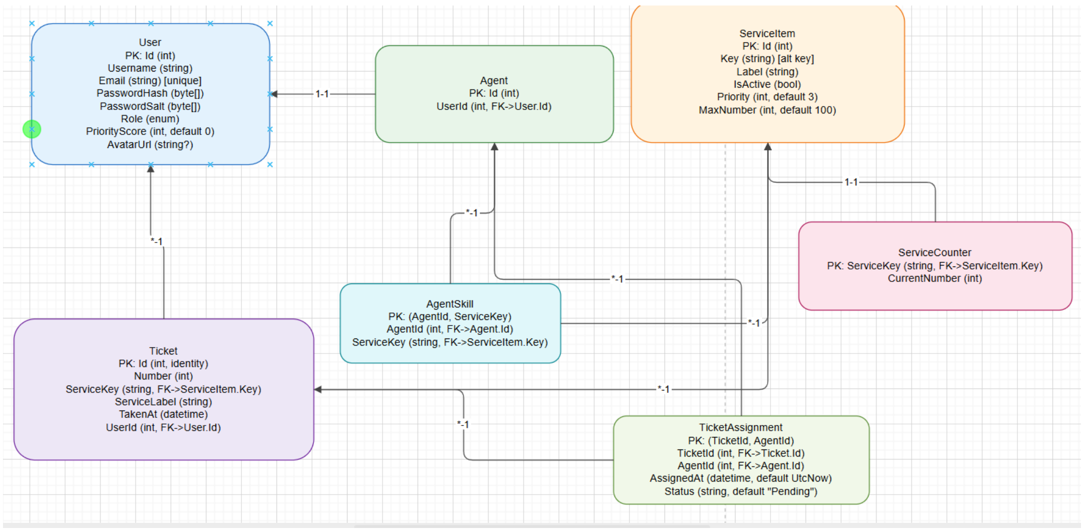
User
PK: Id (int)
Username (string)
Email (string) [unique]
PasswordHash (byte[])
PasswordSalt (byte[])
Role (enum)
PriorityScore (int, default 0)
AvatarUrl (string?)
Agent
PK: Id (int)
UserId (int, FK->User.Id)
ServiceItem
PK: Id (int)
Key (string) [alt key]
Label (string)
IsActive (bool)
Priority (int, default 3)
MaxNumber (int, default 100)
ServiceCounter
PK: ServiceKey (string, FK->ServiceItem.Key)
CurrentNumber (int)
AgentSkill
PK: (AgentId, ServiceKey)
AgentId (int, FK->Agent.Id)
ServiceKey (string, FK->ServiceItem.Key)
Ticket
PK: Id (int, identity)
Number (int)
ServiceKey (string, FK->ServiceItem.Key)
ServiceLabel (string)
TakenAt (datetime)
UserId (int, FK->User.Id)
TicketAssignment
PK: (TicketId, AgentId)
TicketId (int, FK->Ticket.Id)
AgentId (int, FK->Agent.Id)
AssignedAt (datetime, default UtcNow)
Status (string, default "Pending")
1-1
1-1
*-1
*-1
*-1
*-1
*-1
*-1

Figure 3: Entity–Relationship (ER) Diagram — BankNumerator

9 References

1. Angular Official Documentation (v20) `https://angular.dev`
2. JWT.IO — Introduction to JSON Web Tokens `https://jwt.io/introduction`
3. Playwright Testing Framework Documentation `https://playwright.dev`