PROCESS RAPPORT- DIXEN

Event Management, Discovery & Ticketing System

18-02-2026

Dikshya Singh Shah

CLASS:1205hf26016per

TECHNICAL EDUCATION COPENHAGEN

Guldblommevej 5, Ballerup

Dikshya Singh Shah
Class: 1205hf26016per

Project title: Process Report of Dixen

(Event Management, Discovery & Ticketing System)

Project Title: Dixen

Name: Dikshya Singh Shah

Project period: 19-01-2026 – 18-02-2026

School: TEC Ballerup

Class: 1205hf26016per

Word count: 60.874 characters with space

Total Figures: 14

Supervisor: Flemming Sørensen

Signature fields

Signatures:

Student: _____ **Date:** _____

DIKSHYA SINGH SHAH

Dikshya Singh Shah
Class: 1205hf26016per

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Full Form
2FA	Two-Factor Authentication
API	Application Programming Interface
ASP.NET	Active Server Pages.NET
CRUD	Create, Read, Update, Delete
CORS	Cross-origin Resource Sharing
CSS	Cascading Style Sheets
DI	Dependancy Injection
DTO	Data Transfer Object
EF Core	Entity Framework Core
HTML	Hyper Text Markup Language
HTTP/ HTTPS	Hypertext Transfer Protocol/ Hyper Text Transfer Protocol Secure
IDE	Integrated Development Environment
ER	Entity-Relationship Diagram
JWT	JSON Web Token
LINQ	Language integrated Query
ORM	Object Relational Mapping
OTP /TOTP	One Time Password/ Time-Based One-Time Password
QR Code	Quick Responsive Code
REST	Representational State Transfer
SPA	Single-Page Application
SQL	Structured Query Language
SMTP	Simple Mail Transfer Protocol
URL	Uniform Resource Locator
UX /UI	User Experience/ User Interface
PBKDF2	Password-Based Key Derivation Function 2
SDK	Software Development Kit
EN	English (for multi-language support)
DA	Danish language code (for multi-language support)

1. READING GUIDE

This process report describes the planning, development process, technology choices, and reflections behind the Dixen web application.

The report is as follows:

- Sections 2–5 Introduce the background, purpose, and problem formulation.
- Section 6 explains project planning and management, including schedule and methodology
- Section 7 describes the development process step by step.
- Section 8 presents system functionality and technical implementation.
- Sections 9–11 cover architecture, tools, lessons learned, and future improvements.
- Appendices contain detailed documentation such as mockups, diagrams, logbook, source code, and test reports.

This document complements the Product Report, which focuses on system functionality and technical specifications. Full source code is available on GitHub (see Appendix 8)

2. PREFACE

This process report describes the development journey of Dixen, a web-based event management system. It focuses on the planning, methods, technologies, development steps, iterations, challenges, and lessons learned throughout the project. The report also reflects on the learning process involved in planning, building, and delivering a real-world solution. I am grateful to everyone who supported me during this process.

3. WHO AM I?

My name is Dikshya Singh Shah, and I am a 35-year-old mother of two currently studying IT at TEC Ballerup. I am completing a school-based internship in Hvidovre, where I work with web and mobile application development technologies including HTML, Tailwind CSS, .NET, C#, API, Java, SQL, bootstrap and Entity Framework.

Before entering IT, I worked for three years as a District Project Coordinator with Ipas Nepal and CARE Nepal, managing health projects and stakeholder coordination. I hold a bachelor's degree in public health and transitioned into IT because I find programming more engaging and rewarding.

4. INTRODUCTION AND CASE DESCRIPTION

This report provides an overview of how Dixen was planned, developed, and completed, from the first idea to the finished Process. It covers the challenges I faced, the decisions I made, and the technologies I used, along with reflections on the learning process.

Dixen is a modern, web-based event management, discovery, and ticketing platform designed to simplify the event lifecycle for users, Host (organizers), and administrators. It provides secure event registration, event discovery, proposal submission and approval workflows, ticketing, social sharing, and analytics. The system addresses common challenges in event management platforms, including security, usability, and scalability *for more details, see the Requirements specification section 4.3).*

4.1 PURPOSE

The main goal of Dixen was to create a system that simplifies event management and improves user engagement. The system allows users to easily discover events, register, purchase tickets, and share events on social media. Hosts (Organizers) can efficiently submit and manage events, from proposal and approval to execution and ticket management, while administrators maintain centralized control, ensure security, and monitor overall platform activity.

Dixen was developed using ASP.NET Core 8 for the backend, Angular v20.3.16 for the frontend, and Microsoft SQL Server for the database, with the aim of delivering a secure, responsive, and user-friendly platform that integrates multiple roles and real-world workflows.

5. PROBLEM FORMULATION

Digital event platforms are increasingly important for how users discover, participate in, and organize events. However, many existing solutions have weaknesses in security, workflow clarity, and role management, which reduce usability, trust, and engagement. Insufficient authentication, unclear event proposal processes, poorly synchronized frontend and backend systems and inconsistent language create friction for participants, organizers, and administrators.

From a participant perspective, event discovery, registration, and sharing should be simple, intuitive, and secure. Weak login mechanisms without proper email verification or two-factor authentication (2FA) expose platforms to security risks. Clear and consistent language is also essential so that participants can navigate the platform and understand event information easily.

From an organizer perspective, the absence of structured workflows for submitting, approving, and managing events makes it difficult to plan and execute events efficiently. Administrators similarly require clear tools for role management, approval, and system oversight to ensure stable platform operation.

The core problem addressed in this project is therefore:

How can a secure, role-based, and user-friendly event management system be designed to support event discovery, proposal workflows, administration, and clear communication, while ensuring data integrity and frontend–backend synchronization?

To address this problem, the project focuses on the following development-oriented questions:

- How can secure authentication be implemented using modern standards (e.g., JWT, email confirmation, and 2FA) without reducing usability?
- How can role-based access control clearly separate the responsibilities of participants, organizers, and administrators?
- How can frontend and backend systems be designed to remain synchronized and consistent during event creation, approval, and booking workflows?
- How can user engagement and clear communication be supported, for example through social sharing and consistent language?

Dixen aimed to create a system that solves these issues and offers smooth experience for everyone involved.

6. PROJECT PLANNING AND MANAGEMENT

6.1 PROJECT PLANNING

The foundation of this project was laid out in the process report under "Requirements Specification," where I clearly defined the functional and non-functional requirements. These requirements served as a detailed roadmap, guiding the development process from start to finish.

To better visualize the user experience and interface, I created several hand-drawn mockups illustrating key pages and workflows. These sketches helped me refine the layout and navigation early on. Digital versions of all mockups are included in "*Appendix 2 – Mockups*"

I adopted an agile development approach, allowing me to build the system incrementally while maintaining flexibility to adapt to new insights or challenges throughout the project.

6.2 PROJECT MANAGEMENT

Since the project was developed individually, a formal Kanban board was implemented but not strictly followed. Instead, project management was primarily handled through a detailed schedule that broke work into smaller, manageable tasks with clear deadlines. This hybrid approach effectively tracked progress, maintained momentum, and eliminated unnecessary overhead while ensuring timely completion.

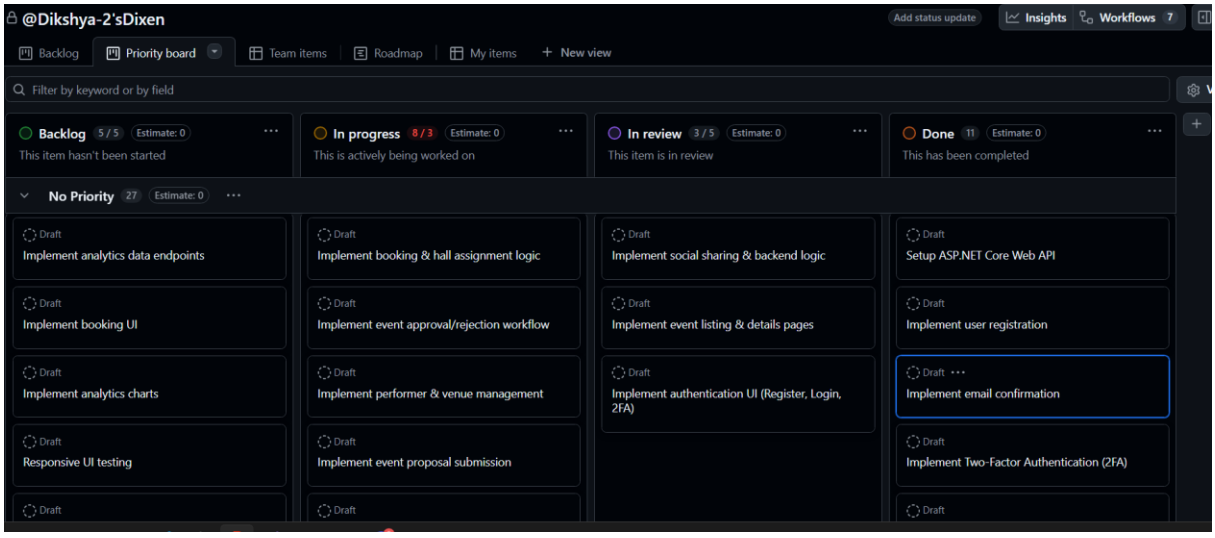


Figure 1 : Kanban board and task scheduling (https://github.com/users/Dikshya-2/projects/8)

6.3 DIVISION OF LABOR

Since I was unable to join a group, I chose to work independently on the project. As a result, I took full responsibility for the entire project, both frontend and backend development. I selected technologies and tools I'm already familiar with to work efficiently and avoid unnecessary delays.

Working alone gave me full control over all aspects of the project, allowing me to make quick decisions and continuously adapt the work to meet my own standards and goals. This approach also helped me develop a deeper understanding of the entire development process, improve my problem-solving skills, and strengthen my ability to manage time and responsibilities effectively. Taking full ownership of the project has been both a challenge and a valuable learning experience.

6.4 ESTIMATED SCHEDULE AND PROJECT TIMELINE

During the requirements specification phase, I created an estimated schedule and project timeline using Excel (*see appendix 4 workplan*). This timeline outlined the main phases and tasks of the project, including backend setup, frontend development, authentication features, event modules, and final testing and documentation.

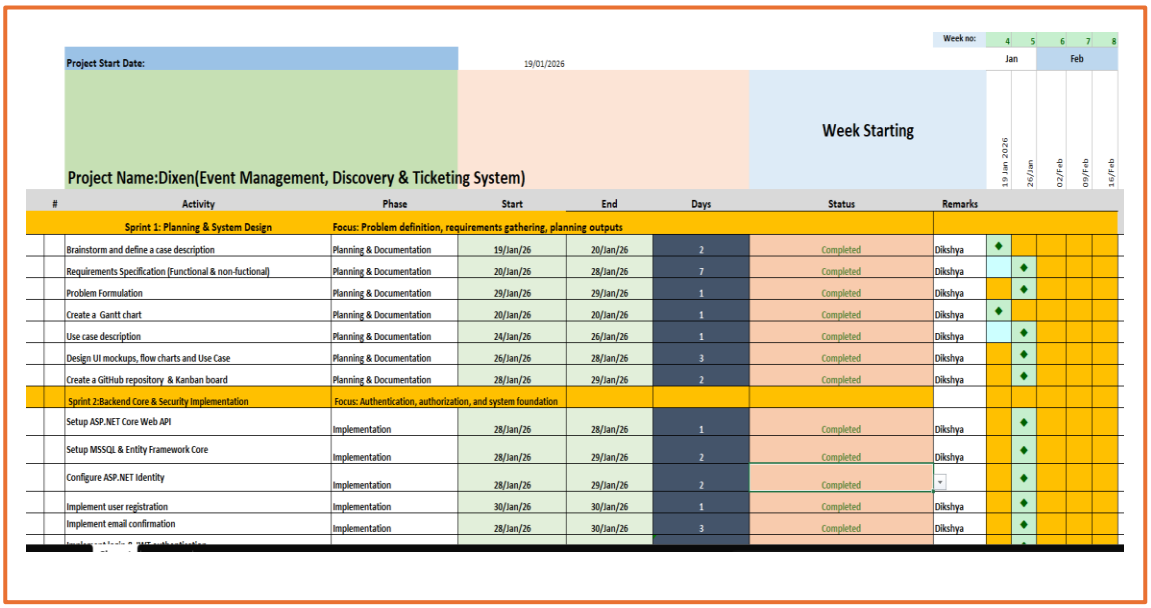
The schedule was designed to provide a clear overview of when each milestone should be reached and to help manage my time effectively throughout the project.

My initial estimate was that the entire project-including both development and report writing-would take approximately five weeks. Each phase was given roughly equal time in the schedule, reflecting a balanced approach to the different areas of work. The schedule was structured into five weekly sprints, each focusing on a defined set of features and goals, in line with the Agile methodology described earlier.

It is important to note that this timeline was an estimate created at the start of the project. As development progressed, I used the schedule as a guideline, adjusted when necessary. For example, the phase involving analytics, event submission and event create, and Update phase took longer than planned due to unexpected technical challenges, while the event list UI was completed ahead of schedule thanks to efficient use of Tailwind CSS.

Overall, having an estimated schedule from the requirement specification phase helped me stay organized, track my progress, and ensure that I met my project deadlines. The experience also taught me the importance of flexibility, as real-world development often requires adapting the plan to address new challenges or opportunities.

A detailed daily log of tasks, challenges, and progress throughout the development process is documented in Appendix 5.



Project Start Date: 19/01/2026

Project Name:Dixen(Event Management, Discovery & Ticketing System)

Week no: 4 5 6 7 8 — Jan — Feb

Week Starting: 19 Jan 2026 | 26/Jan | 02/Feb | 09/Feb | 16/Feb

#	Activity	Phase	Start	End	Days	Status	Remarks
	Sprint 1: Planning & System Design	Focus: Problem definition, requirements gathering, planning outputs					
	Brainstorm and define a case description	Planning & Documentation	19/Jan/26	20/Jan/26	2	Completed	Dikshya
	Requirements Specification (Functional & non-fuctional)	Planning & Documentation	20/Jan/26	28/Jan/26	7	Completed	Dikshya
	Problem Formulation	Planning & Documentation	29/Jan/26	29/Jan/26	1	Completed	Dikshya
	Create a Gantt chart	Planning & Documentation	20/Jan/26	20/Jan/26	1	Completed	Dikshya
	Use case description	Planning & Documentation	24/Jan/26	26/Jan/26	1	Completed	Dikshya
	Design UI mockups, flow charts and Use Case	Planning & Documentation	26/Jan/26	28/Jan/26	3	Completed	Dikshya
	Create a GitHub repository & Kanban board	Planning & Documentation	28/Jan/26	29/Jan/26	2	Completed	Dikshya
	Sprint 2:Backend Core & Security Implementation	Focus: Authentication, authorization, and system foundation					
	Setup ASP.NET Core Web API	Implementation	28/Jan/26	28/Jan/26	1	Completed	Dikshya
	Setup MSSQL & Entity Framework Core	Implementation	28/Jan/26	29/Jan/26	2	Completed	Dikshya
	Configure ASP.NET Identity	Implementation	28/Jan/26	29/Jan/26	2	Completed	
	Implement user registration	Implementation	30/Jan/26	30/Jan/26	1	Completed	Dikshya
	Implement email confirmation	Implementation	28/Jan/26	30/Jan/26	3	Completed	Dikshya

Figure 2: Project workplan and Gantt chart

7. THE DEVELOPMENT PROCESS

7.1 OVERVIEW

The development of Dixen began even before the official project period. The initial idea came during a personal experience: I was struggling to find a suitable hall to Host (Organizers) my son's weaning ceremony. This challenge sparked the thought of creating a platform where people could easily discover, book, and manage events. Drawing on my prior experience in web development, I began imagining a web application that could help users like me connect effortlessly with venues and event Host (Organizers).

I also realized that promoting the platform through social media would be essential, not only to help users discover events but also to increase awareness about the website itself. This inspired me to include features such as social sharing, which allows users to share events across multiple platforms and enhances user engagement.

During the early brainstorming phase, my supervisor, Flemming Sørensen, encouraged me to "look bigger" and design a project with multiple features and opportunities - like a cinema system, which offers various user interactions and functionalities. His advice motivated me to expand the scope of Dixen, aiming not only for a functional system but also for a project that would challenge me and provide comprehensive learning about web application development.

I also wanted to integrate my diverse background into this project. My bachelor's degree in public health gave me knowledge in biostatistics and epidemiology, but I had previously struggled to find ways to apply it in programming. While I briefly considered developing a health-related application, I realized that combining my understanding of data analysis, user behavior, and system design could be applied effectively in an event management platform. This allowed me to explore data insights, analytics, and user interactions in a meaningful context.

Another important consideration that came from my personal experience as an immigrant in Denmark was language accessibility. Many users may have difficulty navigating Danish-only platforms. This inspired me to plan for multi-language support, ensuring that users from diverse backgrounds can use Dixen comfortably. This feature not only improves usability but also broadens the reach of the platform, making it inclusive for a larger audience.

For the planning process, I followed PRINCE2 principles, which emphasize structured project planning, risk management, and efficient use of resources. Using this framework, I carefully defined the project's scope, identified functional and non-functional requirements, and selected technologies I was already familiar with-ASP.NET Core for the backend and Angular for the frontend. This allowed me to focus on delivering the features rather than spending excessive time learning new tools.

For the development process, I applied Agile principles, dividing the work into five weekly sprints with clear goals and deliverables. This incremental approach allowed me to develop the system step by step, continuously testing, refining, and improving both backend and frontend components. Features such as role-based access control, two-factor authentication, event discovery, social sharing, and multi-language support were built iteratively, allowing me to adapt quickly based on testing outcomes and usability feedback.

Overall, Dixen's development process reflects a combination of personal motivation, prior experience, mentorship guidance, structured planning (PRINCE2), and iterative development (Agile). This approach allowed me to grow technically and creatively while producing a practical, user-focused web application. *See Appendix-5* for the full timeline

7.2 PROCESS STEPS

7.2.1 PLANNING AND SYSTEM DESIGN

- Created an estimated schedule and timeline during the requirements specification phase, outlining key milestones and deliverables (*see appendix 4 workplan*).
- Defined functional and non-functional requirements, flow-chart, use-case descriptions & diagrams and prepared mockups for the main user interfaces using Draw.io, Lucid chart and figma (*appendix 1 & 2*).
- Choose ASP.NET Core and Angular for their scalability, modern development experience, and community support.

7.2.2 BACKEND CORE AND SECURITY IMPLEMENTATION

- Set up the ASP.NET Core Web API project and configured the database using Microsoft SQL Server and Entity Framework Core & connected to the database.

Server=(localdb)\\MSSQLLocalDB;Database=Dixen;Trusted_Connection=True;TrustServerCertificate=True

- Implemented user authentication and authorization with ASP.NET Core Identity, JWT tokens for secure API communication, and role-based access control.
- Integrated email confirmation via SMTP and enabled two-factor authentication (2FA) using Google Authenticator, including QR code generation for OTP setup.
- Generated keys with SymmetricSecurityKey and secured tokens using HmacSha256.
- Configured JWT authentication:

```
var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
```

7.2.3 EVENT & BUSINESS LOGIC DEVELOPMENT

- Developed core business logic for the Dixen event management system, including event creation, registration, categorization, booking workflows, and administrative approval processes.
- Implemented CRUD APIs using ASP.NET Core Web API and Entity Framework Core to support event, performers, venue, and booking management, along with backend logic for social sharing and analytics.
- Registered services in the DI container, including AddIdentity, AddJwtBearer, AddDbContext, IEmailSender for email confirmation, TwoFAService for two-factor authentication, JWTService for token handling, and core business/repository services such as EventService and BookingService. Configured CORS to allow frontend integration and configure openApi with security for API testing.

7.2.4 FRONTEND AND INTEGRATION

- Built the frontend using Angular 20.3.16, TypeScript 5.9.3, and tailwind css 4.1.18, jwt-decode 4.0.0, focusing on a responsive and user-friendly interface (see appendix 3, 6, 8 & 9).

- Connected the frontend to the backend API, handling authentication flows (login, registration, 2FA) and secure token storage.
- Developed UI components for event discovery, registration, ticket booking, Host (Organizers) proposal submission, multi-language support, and admin dashboards. Integrated all components with backend APIs, handling authentication flows (login, registration, 2FA) and secure token storage.

7.2.5 TESTING, DOCUMENTATION AND FINAL DELIVERY

- Conducted unit testing on backend APIs using xUnit (Inmemory and Moq), covering scenarios such as successful responses, error handling, and exception flows (see appendix 7).
- Wrote test cases for controller methods (e.g., Get, Create, Update, Delete) with mocked repositories to ensure data isolation.
- Performed manual testing of all features, including authentication, event workflows, and error handling.
- Debugged frontend issues using browser dev tools and Angular CLI.
- Continuously improved both backend and frontend based on test outcomes and usability feedback.
- Iteratively improved both frontend and backend based on test results and usability feedback.
- Fixed bugs and optimized performance where necessary.
- Used Swagger UI for API testing.
- Performed frontend debugging using browser dev tools and Angular CLI test tools.
- Iterated based on usability feedback and resolved bugs related to token expiration and UI responsiveness.
- Added role-based access control to ensure that organizers(host), users, and admins could only access relevant features.
- Writing the final report with screenshots (in progress)
- Preparing submission materials (documentation + code)

8. SYSTEM DESCRIPTION

Dixen is a modern single-page web application that enables users to browse upcoming events, register securely with email confirmation and two-factor authentication (2FA), and book tickets. Host (Organizers) can submit proposals for new events, assign performers, and track event status, while administrators review, approve, and manage all listings.

All data is securely stored in a Microsoft SQL Server database and accessed via a clean and scalable ASP.NET Core Web API. The platform supports multi-language functionality and ticketing for different types of events, offering a responsive and user-friendly experience

8.1 DIXEN WEB-APPLICATION FUNKTIONALITET

The functionality of the web application is based on the Use-Case descriptions in "*Appendix-1 Use Case Descriptions*". Dixen web application functions are divided into some central sections that are easily accessible to users. These sections can be easily navigated via the menu at the top of the app in the navigation bar (navbar). *See in appendix-6 section navigation Bar more details.*

8.2 USER REGISTRATION AND LOGIN:

The Dixen platform implements secure user registration and login, including email confirmation and two-factor authentication (2FA) using Google Authenticator. Below are code examples from both the backend (ASP.NET Core) and frontend (Angular) that illustrate the main logic for these features.

8.2.1 REGISTRATION AND LOGIN FLOW

The registration process on the Dixen platform is implemented securely, including email confirmation and two-factor authentication (2FA) using Google Authenticator.

Steps:

- User fills out registration form on Angular frontend.
- Frontend sends registration data to backend API /api/auth/register.
- Backend validates users, creates accounts, assigns role, and sends email confirmation via SMTP.
- User clicks confirmation link.
- Backend enables 2FA - generates QR code for Google Authenticator.
- User scans QR code to configure 2FA.

- Login: backend checks credentials, email confirmation, and 2FA before issuing JWT/session token.

Figure 3: User Registration, 2FA and Authentication flow (https://github.com/Dikshya-2/Dixen/wiki)

8.2.2 BACKEND IMPLEMENTATION

The backend implementation of the Dixen platform is divided into three key functional areas. The Registration Endpoint handles user creation, role assignments, and generates email confirmation tokens to ensure that only verified users can access the platform. The Email Service is responsible for sending HTML emails via SMTP, including the confirmation link, while securely managing SMTP configurations.

Finally, the QR Code Image function generates a Base64-encoded QR code containing the 2FA secret key, allowing users to set up Google Authenticator for two-factor authentication without exposing the secret directly. Together, these three components ensure secure registration, email verification, and 2FA setup.

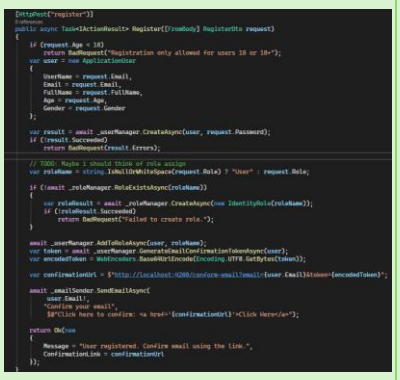
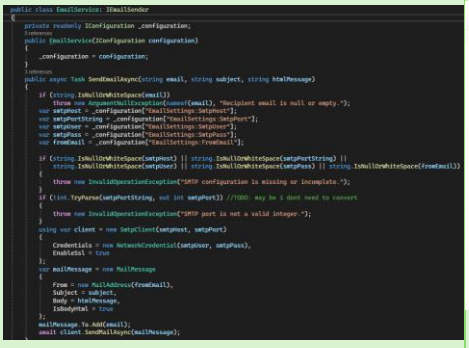
Registration Endpoint	Email Service	QR code Image
		
Code shows flow of registration	*SMPT Email sending diagram*	*QR code generation for Google Authenticator 2FA*

Figure 4: Backend registration components overview (for more details see appendix-8)

8.2.3 FRONTEND IMPLEMENTATION (ANGULAR)

The Angular frontend handles user registration and login using reactive forms to ensure proper validation and smooth user experience.

The registration form collects user details such as email, password, full name, age, and gender, submits the data via AuthService.register(), and displays success or error messages dynamically.

Login & Registration Component

Figure 5: Screenshots of angular registration & login

The login form collects email and password, submits them via AuthService.login(), and shows an error if authentication fails. Together, these forms provide a secure and user-friendly interface for account creation and login.

8.3 EVENT SEARCH AND FILTERING

8.3.1 USER EXPERIENCE:

Dixen offers an advanced event search interface where users enter event title (prefix search), event date (exact day), or venue city (exact match). A dedicated Angular search component uses two-way binding to capture inputs, shows a loading indicator during API calls, displays "No events found" for empty results, and renders matches in a responsive Tailwind grid with title, formatted date, city/address, image, and category names.

8.3.2 TECHNICAL IMPLEMENTATION:

The frontend sends a POST request with EventSearchFilterDto (title, eventDate, venueCity) to */api/Event/search* via a GenericService. The EventController delegates to EventService. SearchEventsAsync, which builds an EF query on Evnt entities (including Categories and Halls, Venue), applies filters dynamically, orders by title, and maps to EventResponseDto (using first hall's venue for location).

8.3.2.1 BACKEND FILTERS PRECISELY:

Title: StartsWith (case-insensitive prefix).

EventDate: Events starting on that UTC day only.

VenueCity: Exact match on any hall's venue city.

Additional backend filters (Keyword contains in title/description, OrganizerId, CategoryIds) exist but aren't used in this UI.

8.3.2.2 CODE EXAMPLE:

This example shows how user input is bound to a filter DTO and sent to the backend when a search is performed.

The backend returns matching events, which are then displayed dynamically in the user interface.

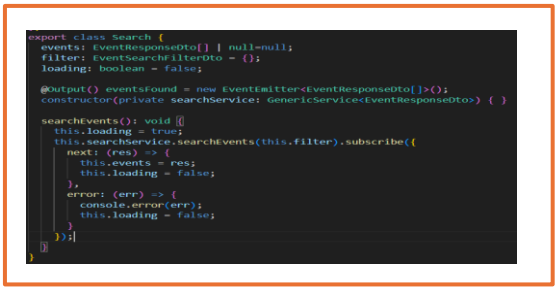


Figure 6: Dynamic event filtering code

8.3.3 USER INTERFACE AND FLOW:

The event search has a simple and responsive layout with fields for title, date, and city, along with a search button. While searching, a loading indicator is shown. The results are displayed as event cards in a responsive grid, and a “No events found” message appears if there are no results.

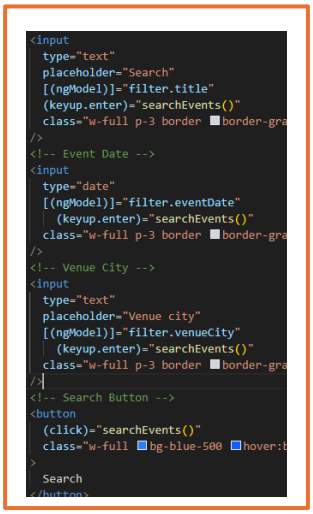


Figure 7: Event search UI flow

See Appendix 6, section- error handling – Event search and Filtering, for a visual example of this functionality.

ERROR HANDLING AND VALIDATION:

Frontend logs errors via console.error and resets loading. Backend returns empty list with no matches (no explicit validation shown). UI shows user-friendly "No events found" message.

8.3.4 OUTCOME:

The implemented search and filtering functionality allows users to efficiently locate relevant events using multiple criteria, enhancing usability and flexibility. By performing filtering on the server side, the system remains scalable and avoids unnecessary client-side processing, ensuring consistent performance as the number of events grows.

8.4 SOCIAL SHARING

8.4.1 USER EXPERIENCE:

Users can share events directly on social media platforms such as Twitter, Facebook, and WhatsApp. This feature increases event visibility, promotes user engagement, and helps events reach a broader audience. Users simply click the "Share" button associated with an event, choose a platform, and share the event link along with a brief description.

A share count badge is displayed for each event, allowing users to see how often an event has been shared.

8.4.2 TECHNICAL IMPLEMENTATION

The Social Sharing functionality is implemented across both the frontend (Angular) and the backend (.NET Web API):

8.4.2.1 FRONTEND (ANGULAR):

- The share() function handles the logic for opening the appropriate share URL (based on platform) and calls the backend to record the share.
- The getShareCount() function fetches how many times each event has been shared.
- toggleShare() toggles visibility of the social sharing buttons per event.
- Share counts are dynamically displayed beside each event.

8.4.2.2 BACKEND (ASP.NET CORE API):

- The SocialShareController manages all operations related to social shares.
- Create: Logs a share action with the selected platform, user ID, and event ID.
- GetAll / GetById / GetShareCount: Retrieves shared records and counts.
- Each social share is stored in a SocialShares table in the database and linked via foreign keys to users and events *(*see Appendix 8- sourceCode*)*.

8.4.2.3 CODE EXAMPLE (FRONTEND – ANGULAR & BACKEND – ASP.NET CORE)

```
share(platform: string, event: Evnt): void {
const userEmail = getUserEmailFromToken();
const pageUrl = encodeURIComponent(
  `${window.location.origin}/event/${event.id}`
);
const text = encodeURIComponent(`Check out this event: ${event.title}`);

const shareUrlMap: Record<string, string> = {
  facebook: `https://www.facebook.com/sharer/sharer.php?u=${pageUrl}`,
  twitter: `https://twitter.com/intent/tweet?text=${text}&url=${pageUrl}`,
  whatsapp: `https://wa.me/?text=${text}%20${pageUrl}`,
};

window.open(shareUrlMap[platform], '_blank');

const payload: SocialShare = {
  eventId: event.id,
  platform,
  userEmail: userEmail ?? undefined,
};

this.socialShareService.shareEvent(payload).subscribe({
  next: () => {
    this.shareCounts[event.id] = (this.shareCounts[event.id] || 0) + 1;
  },
  error: (err) => console.error('Failed to save share', err),
});


getShareCount(eventId: number): void {
  this.socialShareService.getShareCount(eventId).subscribe({
    next: (count) => (this.shareCounts[eventId] = count),
    error: (err) => console.error(err),
  });
}
toggleShare(eventId: number): void {
  this.shareVisible[eventId] = !this.shareVisible[eventId];
```

```
[HttpPost]
0 references
public async Task<ActionResult<SocialShareDto>> Create([FromBody] SocialShareDto socialShareDto)
{
    string? userId = null;

    if (!string.IsNullOrWhiteSpace(socialShareDto.UserEmail))
    {
        var user = await _userManager.FindByEmailAsync(socialShareDto.UserEmail);
        if (user == null)
        {
            return BadRequest("Specified user does not exist.");
        }
        userId = user.Id; // Use the user's ID for FK
    }

    var eventExists = await _eventRepo.Exists(socialShareDto.EventId);
    if (!eventExists)
    {
        return NotFound("Event not found.");
    }

    var socialShare = new SocialShare
    {
        Platform = socialShareDto.Platform,
        SharedAt = DateTime.UtcNow,
        EventId = socialShareDto.EventId,
        UserId = userId // Store the user's ID, not email
    };

    await _socialShareRepo.Create(socialShare);

    // Prepare response DTO
    var responseDto = new SocialShareDto
    {
        Id = socialShare.Id,
        Platform = socialShare.Platform,
        SharedAt = socialShare.SharedAt,
        EventId = socialShare.EventId,
        UserEmail = socialShareDto.UserEmail // Return the email for reference
    };

    return CreatedAtAction(nameof(GetById), new { id = responseDto.Id }, responseDto);
}
```

Figure 8: social share controller and angular logic

8.4.3 USER INTERFACE AND FLOW

- The user clicks “Share Event” on the event card.
- A list of available platforms (Twitter, Facebook, WhatsApp) appears.
- The user selects a platform.
- A new browser tab opens with a prefilled share message.
- The backend logs the share action.
- The share count is updated in real-time.
- Event Social Share Buttons and Counts *See Appendix 6 social Sharing functionality:*

8.4.4 ERROR HANDLING

- If the user is not authenticated, a warning is logged.
- If the selected platform is not supported, no action is taken.
- If the share logging fails (e.g., backend issue), a console error is recorded.
- If the user or event does not exist in the database, an error response is returned.

8.4.5 OUTCOME

Users can promote events across popular social platforms with one click. The share count feature also adds transparency and social proof, showing users which events are popular or trending. Host (Organizers) benefit from increased visibility and reach, enhancing the event's success.

8.5 EVENT REVIEWS AND RATINGS

8.5.1 USER EXPERIENCE:

Users rate events using a 1-5 start directly on category pages. Stars light up yellow on click. Rating displays live next to each event *see appendix-6 for details*.

8.5.2 TECHNICAL IMPLEMENTATION

8.5.2.1 FRONTEND (ANGULAR CATEGORYDETAIL):

When users click a star to rate an event, the Angular frontend calls the rateEvent function with the event ID and selected rating. This function sends the rating to the backend via POST to the EventReview API endpoint and immediately fetches the updated average rating using GET to api/EventReview/average/{eventId}.

8.5.2.2 BACKEND

Backend saves each individual rating to the EventReview table with the eventId, userId, and rating value. The average endpoint queries the database to calculate the average rating for that specific event using SQL AVG (rating) grouped by even

8.5.2.3 ERROR HANDLING

Ensure that if users aren't logged in, they see a "Login to rate" message. Backend failures cache the rating locally, so it retries on next page load.

8.5.3 OUTCOME

Live ratings give users instant social proof - events showing (3) appear more trustworthy than unrated events.

8.6 MULTI-LANGUAGE SUPPORT

8.6.1 USER EXPERIENCE

- All UI text translate Full ngx-translate integration with EN/DA switching. Translation files (`en.json`, `da.json`) in `src/assets/i18n/` contain all UI text (navbar, footer, auth messages). Dynamic switching without page reloading serves international audience.

8.6.2 IMPLEMENTATION

en.json: Navbar: Sign In, Footer: About Us and so on.

da.json: Navbar: Log Ind, Footer: Om Os and so on.

8.6.3 OUTCOME

Tourists and locals are both comfortable.

8.7 ANALYTICS DASHBOARD

8.7.1 USER EXPERIENCE

Admins and Host (Organizers) access real-time event analytics through a professional Tailwind-styled dashboard featuring agCharts bar charts (shares vs tickets per event) and donut charts (social shares by platform), plus an upcoming events list. The responsive design ensures smooth scaling from mobile to desktop devices.

8.7.2 TECHNICAL IMPLEMENTATION

8.7.2.1 BACKEND (ANALYTICSCONTROLLER AND EVENTANALYSISSERVICE):

The AnalyticsController has one endpoint called GET api/reporting/events-summary.

It calls EventAnalysisService which:

- Counts total tickets sold by adding up all Tickets from all Bookings
- Counts social shares by checking the SocialShare table
- Loads data fast by using Include() to get Categories, Bookings, Tickets in one query
- Does math in memory (after ToList()) so database isn't slow

8.7.2.2 ERROR HANDLING:

- Returns HTTP 500 Problem() details with "REAL ERROR" title

8.7.2.3 FRONTEND (ANGULAR ANALYSIS COMPONENT):

The Angular Analysis components work together to show complete event analytics. app-analysis creates a bar chart comparing shares vs tickets (from /api/reporting/events-summary), while app-analysis2 shows a donut chart of social shares by platform (Facebook, Twitter, etc from Social-Share platforms). Both use beautiful Tailwind styling with loading spinners and display in Admin/Host dashboards, giving Host (Organizers) clear visual insights into event performance and sharing patterns.

8.7.2.4 USER INTERFACE AND FLOW

Dashboard loads: Both charts show blue loading spinners

App-analysis fetches event data: Bar chart (shares vs tickets) appears

App-analysis2 fetches platform data: Donut chart (Facebook, Twitter, etc) appears

8.7.2.5 ERROR HANDLING

API fail: console log error, chart shows loading state

Empty data: spinner persists, table shows empty

8.7.3 OUTCOME

Host (Organizers) instantly see event performance. Relationships between shares and tickets are visually represented in pie chart. Professional dashboard builds trust in Dixen analytics reliability.

8.8 EVENT DETAIL AND TICKET BOOKING

Users can view detailed information about each event, including date, location, venue, and available halls. From the event detail page, users select a hall based on real-time availability, choose a ticket type (e.g., General, VIP, Student), and specify the quantity of tickets.

The system calculates the total price dynamically based on ticket type and quantity. Hall capacity and available seats are retrieved from the backend to prevent overbooking. After the booking is confirmed, tickets are generated and seat availability is refreshed automatically.

The payment step is implemented as a simulated transaction to demonstrate the complete booking workflow without integrating a real payment gateway *see appendix 6 event detail and ticketing interface.*

Figure 9: Event detail and ticket booking flow (https://github.com/Dikshya-2/Dixen/wiki)

8.9 GOOGLE MAPS INTEGRATION

Dixen provides users with the ability to locate event venues directly on Google Maps, improving usability and helping participants plan their visits efficiently. Each event detail page features a "Get Directions" button that opens Google Maps in a new tab, displaying the venue location.

USER EXPERIENCE:

Users click the "Get Directions" button to open Google Maps with the event's exact address. The button is visually prominent with an icon and interactive styling to enhance engagement. Users can view the venue location, explore nearby areas, and get route directions seamlessly.

TECHNICAL IMPLEMENTATION:

A TypeScript function getGoogleMapsLink() creates a Google Maps link using the event's name, address, and city. If any information is missing, it defaults to a safe link. This link is attached to a button in the Angular template, which opens Google Maps in a new tab. The button is styled with Tailwind CSS for a modern and responsive look.

OUTCOME:

- Users can quickly navigate to event venues without leaving the platform.
- Enhances overall user experience by providing a direct, reliable, and familiar mapping service.
- Contributes to Dixen's goal of making event discovery and planning intuitive and user-friendly.

```
getGoogleMapsLink(): string {
  if (!this.event) return '#';
  const venueName = this.event.name?.trim() || '';
  const address = this.event.address?.trim() || '';
  const city = this.event.city?.trim() || '';
  const query = [venueName, address, city]
    .filter((part) => part.length > 0)
    .join(', ');
  return `https://www.google.com/maps/search/?api=1&query=${encodeURIComponent(query)}`;
}
```

Figure 10: Google maps link Generation (refer appendix 6 for UI design)

8.10 ROLE-BASED EVENT MANAGEMENT WITH FULL CRUD (ADMIN ACCESS ONLY)

In this system, full CRUD functionality (Create, Read, Update, Delete) has been implemented for all major entities such as Event, Category, Hall, Performer, Organizer and Booking. Only Admin users are authorized to perform these operations, following a strict role-based access control model.

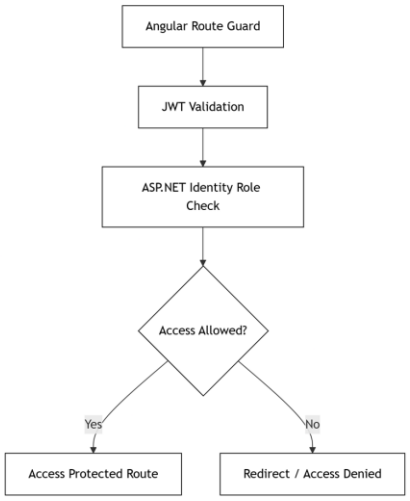


Figure 11: Route Guard & Role-Based Access Flow (https://github.com/Dikshya-2/Dixen/wiki)

8.10.1 ACTOR: ADMIN

Although Host (Organizers) can submit event proposals, all final CRUD operations (Create, Read, Update, Delete) are restricted to Admin users to ensure centralized control and data integrity.

8.10.2 ROLE-BASED ACCESS CONTROL

- Access to each CRUD route is protected via Angular route guards (canActivate: [authGuard]).
- Backend access is secured via role-based authorization using ASP.NET Identity.
- Admin users are assigned a special Admin role, verified before permitting any operation on sensitive endpoints.

8.10.3 ADMIN ROUTING & BACKEND API ROUTES EXAMPLE

All components are structured under the admin module/folder for better separation of concerns & You exposed RESTful endpoints for every entity. For examples

Event Entity CRUD:

- GET /api/Evnt – List all events
- GET /api/Evnt/{id} – View a specific event
- POST /api/Evnt – Create new event
- PUT /api/Evnt/{id} – Edit an event
- DELETE /api/Evnt/{id} – Delete an event
- POST /api/Evnt/Search

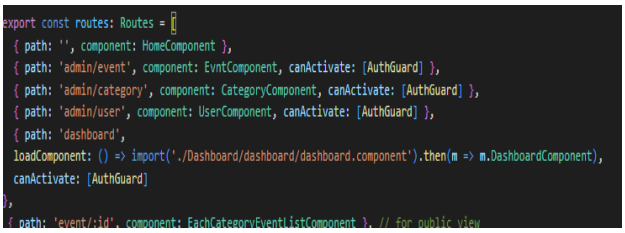


Figure 12: Routing

This pattern is repeated consistently across entities like Category, Hall, Performer, etc. *(see Appendix 8 source code)*

8.10.4 ERROR HANDLING

- The system handles errors such as:
- Invalid input (missing title, date conflict)
- Unauthorized access (non-admin trying to access CRUD routes)
- Backend validation failures (e.g., hall already booked)

8.10.5 SECURITY NOTES

- All API requests check the user's JWT token and role.
- Frontend routes are protected using Angular guards.
- Unauthorized users are redirected and blocked from restricted areas.

9. SYSTEM OVERVIEW

To provide a comprehensive understanding of the Dixen system, I have prepared a set of documents that describe its key aspects and functionalities. The Use Case Descriptions and Use Case Diagram offer a clear overview of the main actions users can perform within Dixen, including use cases tailored for both administrators and regular users.

Flowcharts have been created to visualize user interactions and illustrate how different parts of the system are interconnected. These diagrams highlight the navigation and dynamic behavior of the application, with specific flows for each user role.

During the design phase, I developed mockups to illustrate the overall layout and user interface of the web application, as well as the interaction between various sections. The Sitemap

provides a visual map of user navigation paths, helping to clarify how users move through the site. Additionally, the Database Diagram details the organization and relationships between tables in the system, supporting a clear understanding of the data structure.

Together, these documents strengthen the understanding of Dixen's structure and functionality for all user roles. The related documents can be found in the appendices:

- Use Case Diagram and Descriptions: Appendix 1 – section 12.2
- Flowcharts: Appendix 1 – section 12.3, https://github.com/Dikshya-2/Dixen/wiki
- Mockups: Appendix 2 – Mockups
- Database Diagram: Product report - Database Design

These materials serve as both technical and user documentation, ensuring that stakeholders, developers, and end users can fully understand and effectively interact with the Dixen system.

10. METHOD AND TECHNOLOGY CHOICES

I followed an agile-inspired approach and worked in small, focused steps. The development process was divided into phases, allowing me to deliver and test features progressively. I began by sketching initial mockups on paper to visualize the core user interfaces and then translated them into digital versions using Draw.io. This helped guide both design and functionality throughout development

10.1. TOOLS AND RESOURCES

The tools for Dixen were selected based on scalability, security, and ease of integration:

Frontend: Angular 20.3.16 to ensure a responsive, single-page application (SPA) interface with dynamic interactions.

Backend: ASP.NET Core 8 Web API, delivering a secure and high-performance API layer, supported by Entity Framework for database access.

Database: Microsoft SQL Server, ensuring reliable and scalable relational data storage.

Authentication: ASP.NET Core Identity combined with JWT tokens for secure, role-based access control.

Security Enhancements: Integration of two-factor authentication (2FA) using Google Authenticator and email confirmation via an SMTP server to enhance account security.

Development Tools: Visual Studio, VS Code, GitHub, Postman, Swagger, Draw.io

10.1.1 FRONTEND (CLIENT SIDE)

10.1.1.1 ANGULAR

I used Angular as the frontend framework because of its modular structure, built-in support for TypeScript, and its capability to build responsive single-page applications. Angular helped me organize components for process listings, the login system, and the admin panel efficiently.

10.1.1.2 STANDALONE COMPONENTS & TAILWIND CSS:

Standalone components (noNgModules) and tailwind css for responsive and modern UI.

10.1.1.3 NGX-TRANSLATE (MULTI-LANGUAGE)

Multi-language support with ngx-translate. Dynamic language switching (EN/DA) without page reload.

10.1.1.4 ROUTER GUARD

I implemented an Angular Route Guard (using CanActivateFn) to manage access based on authentication status. The guard checks if the user is logged in via the authService, allowing or redirecting as needed to protect restricted routes.

10.1.1.5 AGCHARTS

gCharts Community for pie charts and donut charts in analytics dashboard.

10.1.1.6 ANGULAR ENVIRONMENT AND PROXY CONFIGURATION

Angular environment.ts and proxy.conf.json enables API URL switching without code changes.
environment.ts (Development):

10.1.2 BACKEND (SERVER SIDE)

10.1.2.1 ASP.NET CORE WEB API (C#)

I developed the backend using ASP.NET Core Web API, which provided a robust and scalable environment for building RESTful services. It allowed me to handle all CRUD operations, manage authentication, and communicate efficiently with the frontend.

10.1.2.2 ENTITY FRAMEWORK CORE (EF CORE)

I implemented EF Core using the Code-First approach. This allowed me to define my models in code and automatically generate the SQL Server database structure. I used EF Core to manage data related to users, events, categories, booking.

10.1.3 AUTHENTICATION & SECURITY

10.1.3.1 ASP.NET CORE IDENTITY

I used ASP.NET Core Identity to handle user registration, login, roles, and secure password storage. Identity automatically hashes and salts passwords using secure algorithms (PBKDF2 by default), so I didn't need to implement custom hashing. It also made it easier to manage features like email confirmation, and two-factor authentication (2FA).

10.1.3.2 JWT AUTHENTICATION

I implemented stateless, secure authentication using JSON Web Tokens (JWT). After a successful login or two-factor authentication (2FA), a token is generated by JWTService and stored in localStorage on the frontend. This token is read by Angular route guards and jwtUtils helper for protecting admin routes and enabling authenticated features like social sharing and event ratings. JWTs provide scalable session management without server-side storage while enabling role-based access control that distinguishes between regular users and administrators, ensuring sensitive operations remain restricted to authorized roles.

10.1.3.3 ROLE-BASED ACCESS CONTROL

I used Identity's role management to assign different roles (e.g., Admin, User and host) and protect specific endpoints based on authorization policies. Angular authguard enforces route protection based on token role.

10.1.3.4 TWO-FACTOR AUTHENTICATION (2FA)

I implemented two-factor authentication using Google Authenticator via QR code and OTP. Upon email confirmation, users are prompted to scan a QR code and enter a time-based token, which is verified on login to add an extra layer of security.

10.1.3.5 HTTP SECURITY

Backend APIs and Swagger run on HTTPS (TLS 1.2+) at https://localhost:7204.

All API calls between Angular frontend and ASP.NET backend are encrypted.

EMAIL VERIFICATION

I used IEmailSender to send email confirmation links. The user must confirm their email before being allowed to log in, ensuring only valid accounts can access the system.

10.1.4 DATABASE & ORM

Entity Framework Core (EF Core): Handles all database operations with LINQ queries and automatic migrations from C# models to SQL Server (localdb\MSSQLLocalDB).

SQL Server LocalDB: Stores users, events, bookings, categories and so on. EF Core keeps database schema in sync with code changes automatically.

10.1.5 OTHER TOOLS & LIBRARIES

10.1.5.1 QR CODE GENERATOR (QRCODER)

Used to dynamically generate QR codes for 2FA setup, encoded as Base64 PNG images to be rendered on the frontend.

10.1.5.2 ZXING & URLENCODER

Used for generating and encoding TOTP URIs in a format compatible with Google Authenticator apps.

10.1.5.3 SWAGGER / OPENAPI

Integrated Swagger UI for easy API testing and documentation during development.

10.1.5.4 CORS POLICY

Configured to allow frontend (e.g., Angular) to make secure cross-origin API calls.

10.1.6 DEPENDENCIES

Microsoft.NET.Sdk (8.0.20): Standard .NET class library SDK for data access layer.

Microsoft.EntityFrameworkCore.SqlServer (8.0.20): SQL Server database provider for Entity Framework - connects Dixen to Microsoft SQL database.

Microsoft.AspNetCore.Identity.EntityFrameworkCore (8.0.20): User authentication system (login/register/roles) integrated with EF Core database.

Microsoft.EntityFrameworkCore.Tools (8.0.20): Database migration tools - creates/updates Dixen database schema from C# models.

Microsoft.EntityFrameworkCore.Design (8.0.20): Runtime support for EF tools and design-time operations.

QRCoder (1.6.0): QR code generator for 2FA setup - creates Google Authenticator QR codes as base64 images.

Microsoft.NET.Sdk.Web (8.0): ASP.NET Core Web API SDK -builds REST endpoints for Dixen frontend.

Microsoft.AspNetCore.Authentication.JwtBearer (8.0.1): JWT token validation middleware - secures API endpoints with your JWT tokens.

Swashbuckle.AspNetCore (6.6.2): Swagger/OpenAPI documentation - generates interactive API docs at /swagger.

ProjectReference Dixen.Repo: Links to data layer - API controllers use Repo **services/repositories.**

Database: ddDbContext<DatabaseContext>()- Connects Dixen to SQL Server (LocalDB)

Repositories: AddScoped<IGRepo<>, GRepo<>>() - Generic CRUD for Events/SocialShares

IEmailSender: Sends registration/booking confirmation emails

IJWTService: Generates JWT tokens after login/2FA

TwoFAService: Google Authenticator QR code & OTP validation

EventService/BookingService/TicketService: Core Dixen business logic

CORS- AllowAnyOrigin(): Allows Angular frontend (localhost:4200) to call API

Swagger: Auto-generates interactive API docs at swagger for testing

Identity & JWT: AddIdentity - User registration & roles (Admin/User)

AddJwtBearer: Validates JWT tokens on every API request

RequireConfirmedEmail: true - Email verification before login

Middleware Pipeline: Https - CORS - Auth - Authorization - Controllers

10.1.7 FRONTEND DEPENDENCIES

angular/core: Core Angular framework - builds/runs SPA components

angular/common: Common directives/pipes (ngIf, date formatting)

angular/forms: Reactive forms & validation for login/register

angular/router: Navigation & route guards (auth protection)

ngx-translate/core: Multi-language switching (EN/DA)

ngx-translate/http-loader: Loads translation JSON files

ag-charts-community: Analytics dashboard pie charts & stats

jwt-decode: Reads JWT token roles/claims for auth guards

tailwindcss: Responsive UI styling (mobile-first design)

fortawesome/angular-fontawesome: Professional icons (social/share)

11. JUSTIFICATION FOR CHOICE OF METHOD AND TECHNOLOGY

The selection of methods and technologies for Dixen was guided by several key factors:

11.1 SCALABILITY AND SECURITY

ASP.NET Core 8 provides a robust, scalable, and secure API framework, while Angular 20.3.16 and tailwind css ensures a responsive, modern single-page application experience. Microsoft SQL Server offers reliable, high-performance data storage.

11.2 INTEGRATION AND FAMILIARITY

As a solo developer, I prioritized technologies I was already proficient with. This allowed me to work efficiently, avoid unnecessary delays, and focus on delivering a functional and secure application.

11.3 EFFICIENT ITERATION

Familiarity with the stack enabled rapid iteration between frontend and backend, ensuring tight integration and quick resolution of issues. This was especially important while working alone, as it allowed me to maintain momentum and adapt quickly to new requirements.

11.4 SECURITY ENHANCEMENTS

ASP.NET Core Identity, JWT tokens, and two-factor authentication (2FA) via Google Authenticator were chosen to address modern security requirements and protect user data. Email verification further ensures only legitimate users gain access.

11.5 COMMUNITY AND SUPPORT

Both ASP.NET Core and Angular have strong community support and extensive documentation, reducing risk and enabling best practices.

11.6 PROJECT MANAGEMENT

Working solo requires making pragmatic decisions that balanced feature set, security, and delivery timeline. By choosing a stack I knew well, I could better manage time and ensure the project stayed on track.

This approach allowed me to focus on delivering a functional, secure, and user-friendly event management platform within the project's time constraints

12. DEVELOPMENT TOOLS / SOFTWARE DESIGNER (VERSION)

12.1 MICROSOFT VISUAL STUDIO COMMUNITY 2022

I used Visual Studio for backend development and database management because of its strong support for ASP.NET Core and integrated tools for debugging and testing.

12.2 VISUAL STUDIO CODE

I used VS Code for frontend development. It provided a lightweight and efficient environment for writing Angular code and managing frontend assets.

12.3 SWAGGER

I used Swagger for documenting and testing my API endpoints. It helped me confirm that all endpoints worked as expected before connecting them to the frontend.

12.4 DRAW.IO AND LUCID CHART

I used Draw.io to create system architecture diagrams and ER diagrams to visually represent the structure and relationships of my application.

12.5 GITHUB DESKTOP

I used GitHub for version control to track my changes and maintain backups. It also helped me manage different development stages through commits and branches.

12.6 MSSQL SERVER MANAGEMENT STUDIO

It helps me check entity Framework database table and test data directly.

12.7 ONEDRIVE

Kept all project files safe with automatic version backup

13. ARCHITECTURE AND STRUCTURE

- I followed a layered architecture in the backend, separating Controllers, Services, and Repositories for better maintainability and testability.
- I used DTOs (Data Transfer Objects) to control the data flow between the backend and frontend, improving performance and security.
- Angular's HttpClient was used to handle communication between the frontend and backend via HTTP requests.

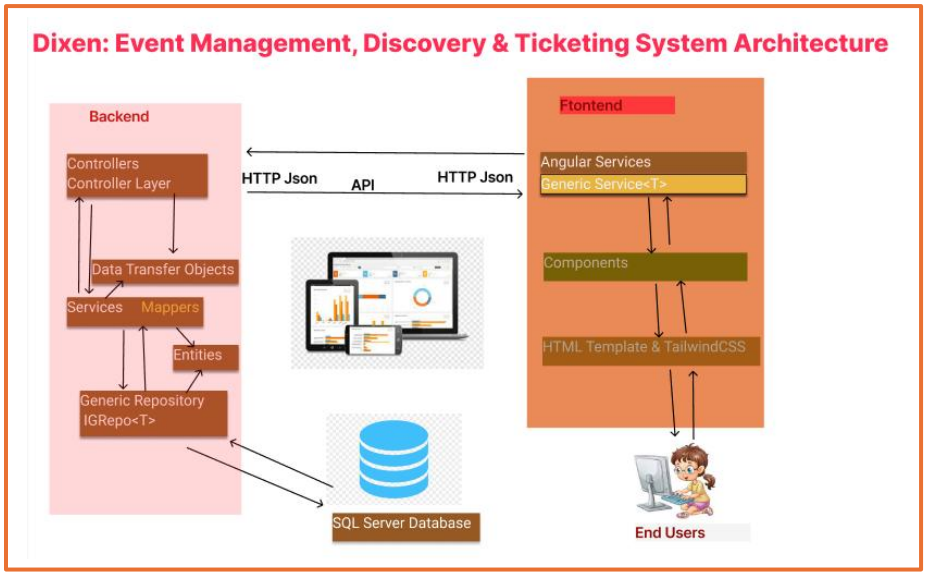


Figure 13: Dixen system Architecture Overview (https://github.com/Dikshya-2/Dixen/wiki)

13.1 ROUTE GUARD IMPLEMENTATION:

To manage route access based on the user's authentication status, I implemented an Angular Route Guard using CanActivateFn. This guard checks whether a user is logged in before allowing access to certain routes.

AuthGuard:

- Custom route guard function implemented using CanActivateFn.
- It uses AuthService to check if the user is logged in (authService.isLoggedIn()).
- If the user is logged in, the guard allows the route to be activated (return true).
- If the user is not logged in, the guard redirects them to the login page using router.navigate(['/login']) and prevents access to the requested route (return false).

This implementation enhances frontend security and aligns with best practices for Angular applications.

14. THE FUTURE OF THE PROGRAM

Progressive Web App: Offline booking, push notifications, Apple Pay/Google Pay, & Installable on iOS/Android home screens.

Advanced Analytics: Predictive ticket sales forecasting and user heatmaps.

Payment Integration: PayPal real payments.

Event Streaming: Virtual attendance, live chat, QR code venue entry.

15. LESSONS LEARNED

Throughout the process, I encountered challenges such as Core Policy blocking my AnalyticsController and difficulties integrating AG Charts in Angular. The CORS issue was eventually resolved by modifying the AnalyticsController route and retesting the Angular integration, as documented in my project logbook (10-02-2026). I addressed these issues through research, step-by-step testing, and adjustments to my project timeline.

While developing the Category-Detail page, I initially implemented the rating specifically for events. Later, I realized that building it as a separate, reusable component would have allowed me to easily apply it to upcoming and recommended movies as well. This experience highlighted the importance of reusable UI design and forward-thinking architecture.

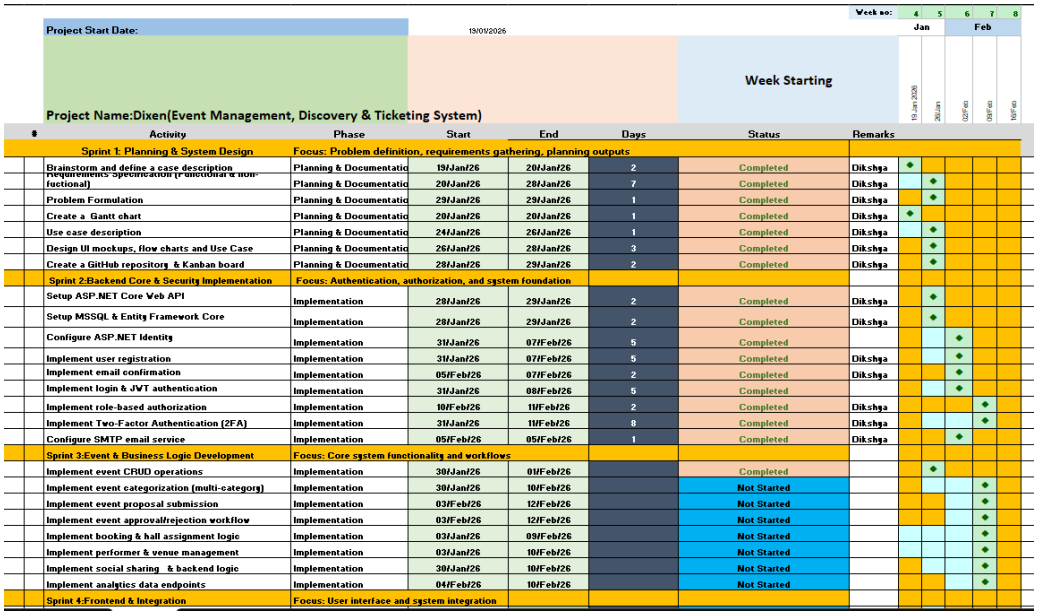
Overall, working independently required careful time management and quick decision-making, which strengthened both my coding and project planning skills.

16. LOGBOOK

A detailed logbook documenting the progress of this project is included. Since I worked independently, the logbook reflects all tasks, decisions, and developments I handled throughout the project. The full logbook can be found in *Appendix 5 – Logbook*.

16.1 REALIZED SCHEDULE

I planned five equal sprints, but reality shifted: the authentication sprint took two extra days because configuring Google Authenticator was trickier than expected. Conversely, the event list UI went faster than estimated because tailwind-CSS saved styling time *(see Appendix 4 workplan)*.



Project Start Date:			19/01/2026				Week no:	4 5 6 7 8
						Week Starting	Jan / Feb	19 Jan 2026 / 26Jan / 02Feb / 09Feb / 16Feb
Project Name:Dixen(Event Management, Discovery & Ticketing System)								
#	Activity	Phase	Start	End	Days	Status	Remarks	
	Sprint 1: Planning & System Design	Focus: Problem definition, requirements gathering, planning outputs						
	Brainstorm and define a case description	Planning & Documentatio	19/Jan/26	20/Jan/26	2	Completed	Dikshya	
	Requirements Specification (Functional & non-fuctional)	Planning & Documentatio	20/Jan/26	28/Jan/26	7	Completed	Dikshya	
	Problem Formulation	Planning & Documentatio	29/Jan/26	29/Jan/26	1	Completed	Dikshya	
	Create a Gantt chart	Planning & Documentatio	20/Jan/26	20/Jan/26	1	Completed	Dikshya	
	Use case description	Planning & Documentatio	24/Jan/26	26/Jan/26	1	Completed	Dikshya	
	Design UI mockups, flow charts and Use Case	Planning & Documentatio	26/Jan/26	28/Jan/26	3	Completed	Dikshya	
	Create a GitHub repository & Kanban board	Planning & Documentatio	28/Jan/26	29/Jan/26	2	Completed	Dikshya	
	Sprint 2:Backend Core & Security Implementation	Focus: Authentication, authorization, and system foundation						
	Setup ASP.NET Core Web API	Implementation	28/Jan/26	29/Jan/26	2	Completed	Dikshya	
	Setup MSSQL & Entity Framework Core	Implementation	28/Jan/26	29/Jan/26	2	Completed	Dikshya	
	Configure ASP.NET Identity	Implementation	31/Jan/26	07/Feb/26	5	Completed		
	Implement user registration	Implementation	31/Jan/26	07/Feb/26	5	Completed	Dikshya	
	Implement email confirmation	Implementation	05/Feb/26	07/Feb/26	2	Completed	Dikshya	
	Implement login & JWT authentication	Implementation	31/Jan/26	08/Feb/26	5	Completed		
	Implement role-based authorization	Implementation	10/Feb/26	11/Feb/26	2	Completed	Dikshya	
	Implement Two-Factor Authentication (2FA)	Implementation	31/Jan/26	11/Feb/26	8	Completed	Dikshya	
	Configure SMTP email service	Implementation	05/Feb/26	05/Feb/26	1	Completed	Dikshya	
	Sprint 3:Event & Business Logic Development	Focus: Core system functionality and workflows						
	Implement event CRUD operations	Implementation	30/Jan/26	01/Feb/26		Completed		
	Implement event categorization (multi-category)	Implementation	30/Jan/26	10/Feb/26		Not Started		
	Implement event proposal submission	Implementation	03/Feb/26	12/Feb/26		Not Started		
	Implement event approval/rejection workflow	Implementation	03/Feb/26	12/Feb/26		Not Started		
	Implement booking & hall assignment logic	Implementation	03/Jan/26	09/Feb/26		Not Started		
	Implement performer & venue management	Implementation	03/Jan/26	10/Feb/26		Not Started		
	Implement social sharing & backend logic	Implementation	30/Jan/26	10/Feb/26		Not Started		
	Implement analytics data endpoints	Implementation	04/Feb/26	10/Feb/26		Not Started		
	Sprint 4:Frontend & Integration	Focus: User interface and system integration						

Figure 14: Realized vs planned Schedule

16.2 SOCIETAL TRENDS

Dixen supports cybersecurity with 2FA and JWT authentication, protecting user data from hackers. Digital ticketing reduces paper waste, promoting sustainability. Multi-language support (EN/DA) improves accessibility for Denmark's immigrant population. These features make Dixen relevant to modern societal needs.

17. CONCLUSION

The development of Dixen has successfully resulted in a secure, scalable, and user-friendly event management platform that directly addresses the challenges defined in the problem formulation. By combining an Agile development approach with structured project management principles from PRINCE2, the project-maintained flexibility while ensuring clear milestones, risk management, and steady progress throughout each development phase.

The selected technology stack-Angular, ASP.NET Core, Microsoft SQL Server, and JWT were carefully chosen to ensure performance, security, and scalability. This architectural foundation allowed Dixen to deliver a robust full-stack solution capable of supporting both current requirements and future expansion.

The problem formulation has been thoroughly addressed.

How can secure authentication be implemented using modern standards (e.g., JWT, email confirmation, and 2FA) without reducing usability?

Secure authentication was implemented using ASP.NET Core Identity combined with JWT-based authentication for secure and stateless session management. Email confirmation via SMTP ensures verified user registration, while two-factor authentication (2FA) using QR-code integration with Google Authenticator adds an extra security layer

Even though these security features are included, the login process remains simple and user-friendly. The system provides clear instructions and only adds extra steps

How can role-based access control clearly separate the responsibilities of participants, organizers, and administrators?

Role-based access control (RBAC) was implemented directly within the backend using ASP.NET Core Identity roles. Users are assigned specific roles- Participant, Organizer (Host), or Administrator- which determine their permissions and accessible features. Route guards on the frontend and authorization policies on the backend ensure that each role only accesses relevant functionality. This separation improves security, prevents unauthorized actions, and maintains a clear distribution of responsibilities within the system.

How can frontend and backend systems be designed to remain synchronized and consistent during event creation, approval, and booking workflows?

Synchronization between frontend and backend was achieved through a structured RESTful API architecture built with ASP.NET Core and consumed by Angular. Data Transfer Objects (DTOs) standardize communication and ensure consistent data structures across the application. Validation is handled both client-side and server-side to prevent inconsistencies. This design ensures that event creation, approval processes, and booking workflows remain accurate, reliable, and fully synchronized in real time.

How can user engagement and clear communication be supported, for example through social sharing and consistent language?

User engagement was enhanced by integrating social sharing functionality, allowing users to easily promote events across external platforms. Additionally, consistent terminology and structured UI design were applied throughout the application to maintain clarity in communication. Clear status indicators (e.g., pending, approved, booked) and structured messaging improve transparency for both organizers and participants. This approach strengthens user engagement while ensuring effective and professional communication across the platform.

In conclusion, Dixen successfully fulfills its original objectives by delivering a comprehensive event management solution that balances security, usability, performance, and maintainability. The combination of modern technologies, structured development methodology, and user-centered design has resulted in a reliable platform capable of supporting both organizers and participants in a secure and efficient manner.

18. APPENDICES

Appendix 1: Requirements Specification

Appendix 2: Mockup

Appendix 3: Commands and installation.

Appendix 4: Work-Plan

Appendix 5: Logbook

Appendix 6: UI Screenshots

Appendix 7: UnitTestReport

Appendix 8: Source Code (GitHub link)

19. REFERENCE

- Microsoft. (n.d.). SQL Server Documentation. Retrieved from https://learn.microsoft.com/en-us/sql/sql-server
- Angular Team. (n.d.). Angular Documentation. Retrieved from https://angular.io/docs
- Microsoft. (n.d.). Entity Framework Core Documentation. Retrieved from https://learn.microsoft.com/en-us/ef/core/
- Microsoft. (n.d.). ASP.NET Core Identity Documentation. Retrieved from https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity
- Microsoft. (n.d.). Two-factor authentication (2FA) in ASP.NET Core Identity. Retrieved from https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity-enable-qrcodes
- Auth0. (n.d.). JSON Web Tokens Introduction. Retrieved from https://auth0.com/learn/json-web-tokens
- Microsoft. (n.d.). Using JWT authentication in ASP.NET Core. Retrieved from https://learn.microsoft.com/en-us/aspnet/core/security/authentication/jwtbearer
- Microsoft. (n.d.). Sending Emails in ASP.NET Core with SMTP. Retrieved from https://learn.microsoft.com/en-us/aspnet/core/fundamentals/email
- Raffaele Rialdi. (n.d.). QRCoder Library Documentation. Retrieved from https://github.com/codebude/QRCoder
- Node.js Foundation. (n.d.). Node.js Documentation. Retrieved from https://nodejs.org/en/docs
- Tailwind css
- Microsoft. (n.d.). Code First Approach in Entity Framework Core. Retrieved from https://learn.microsoft.com/en-us/ef/core/modeling/
- Microsoft. (n.d.). Visual Studio IDE Documentation. Retrieved from https://learn.microsoft.com/en-us/visualstudio/
- Microsoft. (n.d.). Visual Studio Code Documentation. Retrieved from https://code.visualstudio.com/docs
- Microsoft. (n.d.). How to create and validate JWTs in ASP.NET Core.
  Retrieved from:

https://learn.microsoft.com/en-us/aspnet/core/security/authentication/jwt

- Microsoft. (n.d.). Use JWT bearer authentication in ASP.NET Core. https://learn.microsoft.com/en-us/aspnet/core/security/authentication/jwt
- Auth0. (n.d.). JWT.IO Introduction. https://jwt.io/introduction/
- https://tailwindcss.com/docs/installation/using-vite