YILDIZ TECHNICAL UNIVERSITY
FACULTY OF CHEMISTRY-METALLURGICAL
DEPARTMENT OF MATHEMATICAL ENGINEERING

GRADUATION THESIS

DESIGN AND DEVELOPMENT OF AN INTEGRATED CASE AND LAW FIRM MANAGEMENT SYSTEM

ADVISOR: Assoc. Prof. Birol ASLANYÜREK

Osman Bahadır AVCI, 20058015

Istanbul, 2026

CONTENTS

Page

FIGURE LIST

Figures 2.1, 2.2 and 3.6 were generated with the assistance of artifical intelligence tools

LIST OF ABBREVIATIONS

API	Application Programming Interface
CRUD	Create, Read, Update, Delete
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
JWT	JSON Web Token
MERNİS	Central Population Administration System (Turkey)
REST	Representational State Transfer
SSG	Static Site Generation
SSR	Server-Side Rendering
TAKBİS	Land Registry and Cadastre Information System (Turkey)
UYAP	National Judiciary Informatics System (Turkey)

PREFACE

This document is about a project named “Design and Development of an Integrated Case and Law Firm Management System”. This project has been created for my lecture named “Graduation Thesis”. Since the topic of legal technology and law firm management is interesting, I wanted to choose this subject. Also, this topic is related to mathematics, such as the application of scheduling algorithms, relational data modelling and user interface design. Since law firm management is a subject that contains many details, it has been treated generically in this project. The most fundamental modules — Client Management, Case Tracking and Hearing Calendar — have been created. I think that this study has made a significant contribution to my professional life.

I present my gratitudes to my thesis advisor Asst. Prof. Birol ASLANYÜREK for his advice and for being supportive throughout this study, to the lawyers and law firm employees who shared their daily workflow and pain points with me during the requirement gathering phase, to my colleagues who helped me test the application and provided valuable feedback, and lastly to my family for their continuous support.

Osman Bahadır AVCI, 2026

ABSTRACT

The goal of the project is the simplification of law firm operations, time saving for lawyers and reducing administrative overhead inside the office. Users are lawyers, paralegals and assistants of small or medium sized law firms who have basic level of technical knowledge about web applications. A typical situation in which the application will be used is when a lawyer wants to record a new client, follow the status of an ongoing case, or check the calendar for an upcoming hearing, all from a single integrated platform.

In this study, some brief information about “Design and Development of an Integrated Case and Law Firm Management System” is given. In the first chapter, there is an introduction to this topic. In the second chapter, there is an overview of “Next.js”, “Go” and “PostgreSQL”, which are used to build the user interface, the backend services and the database of the system. In the third chapter, details about the project and the system are explained. In addition, this chapter describes how some discussions with practising lawyers were carried out and how the application design and the user interface were created according to these discussions. In this chapter, the design of the user interface consists of five parts named “Database”, “Login and Authentication”, “Client Management”, “Case Tracking” and “Hearing Calendar”. All details about these five parts are given in this document. Finally, in the third chapter, some information about the “Implementation” of the system and a “Scheme” of the system are given for easy understanding of the system.

ÖZET

Bu projenin amacı, hukuk bürosu operasyonlarının basitleştirilmesi, avukatlar için zaman tasarrufu sağlanması ve büro içindeki idari iş yükünün azaltılmasıdır. Sistemin kullanıcıları, web uygulamaları hakkında temel düzeyde teknik bilgiye sahip olan küçük ve orta ölçekli hukuk bürolarındaki avukatlar, avukat yardımcıları ve asistanlardır. Uygulamanın kullanılacağı tipik bir durum, bir avukatın yeni bir müvekkil kaydetmek, devam eden bir davanın durumunu takip etmek veya yaklaşan bir duruşma için takvimi kontrol etmek istemesi ve tüm bunları tek bir entegre platform üzerinden gerçekleştirmesidir.

Bu çalışmada, "Entegre Dava ve Hukuk Bürosu Yönetim Sistemi Tasarımı ve Geliştirilmesi" hakkında özet bilgiler verilmiştir. Birinci bölümde konuya giriş yapılmıştır. İkinci bölümde, sistemin kullanıcı arayüzünü, arka uç (backend) servislerini ve veritabanını oluşturmak için kullanılan "Next.js", "Go" ve "PostgreSQL" teknolojilerine genel bir bakış sunulmuştur. Üçüncü bölümde, proje ve sistem hakkında ayrıntılar açıklanmıştır. Ayrıca bu bölümde, uygulamada çalışan avukatlarla yapılan görüşmelerin nasıl gerçekleştirildiği ve uygulama tasarımı ile kullanıcı arayüzünün bu görüşmeler doğrultusunda nasıl oluşturulduğu anlatılmıştır. Bu bölümde kullanıcı arayüzü tasarımı; "Veritabanı", "Giriş ve Kimlik Doğrulama", "Müvekkil Yönetimi", "Dava Takibi" ve "Duruşma Takvimi" olarak adlandırılan beş bölümden oluşmaktadır. Bu beş bölüme ilişkin tüm ayrıntılar bu dokümanda verilmiştir. Son olarak üçüncü bölümde, sistemin kolay anlaşılabilmesi için sistemin "Gerçekleştirimi (Implementation)" hakkında bazı bilgiler ve sistemin bir "Şeması" sunulmuştur.

1. INTRODUCTION

There are some key values for the project. Key values for the law firm are “lower administrative overhead”, “better organisation of cases and clients”, and “avoidance of missed hearings and deadlines”. Also there are some key values for the lawyers themselves, these are “time saving” and “simplification of daily workflow”. These words show some directions in the whole project.

1.1. Existing Solutions

In Turkey, the most widely known and the most widely used solution related to legal procedures is UYAP (the National Judiciary Informatics System, “Ulusal Yargı Ağı Bilişim Sistemi”). UYAP is an information system created by the Turkish Ministry of Justice for the purpose of carrying out judicial services in an electronic environment, and it has been gradually deployed across the country since the early 2000s [1]. Today UYAP is in operation in all judicial units of the Ministry of Justice, and judicial, administrative and audit activities are conducted electronically through this system [2]. Lawyers interact with UYAP mainly through a dedicated portal, the UYAP Lawyer Portal (“UYAP Avukat Portalı”), which can only be accessed by holders of a valid bar association certificate. Through this portal, lawyers can perform many operations such as filing civil, criminal and enforcement cases, submitting petitions, examining files in which they are a party, sending documents, paying court fees and expenses, and querying the status of cases [3]. A figure showing an example of an existing solution is given as below.

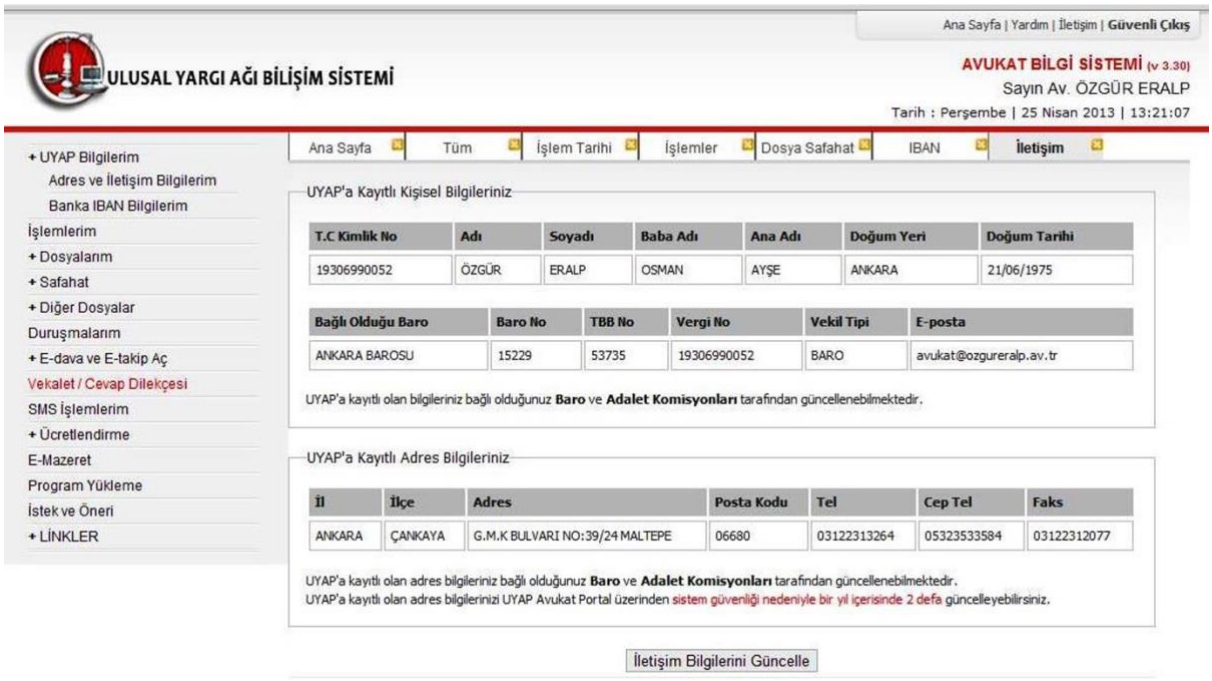

Figure 1.1 Existing Solution — UYAP Lawyer Portal (“UYAP Avukat Portalı”) [3]

Although UYAP is an indispensable tool for any practising lawyer in Turkey, when its scope is examined from the point of view of an individual law firm, several shortcomings appear. UYAP is, first of all, a court-side system: its main purpose is to digitise the workflow inside courts, prosecution offices and enforcement directorates, and to integrate them with external public institutions such as MERNİS, TAKBİS, POLNET and PTT [4]. It is not designed as an internal management tool of a private law firm. As a result, it does not offer features such as managing the firm's own client database with detailed contact information and notes, organising tasks among the lawyers and the assistants of the same office, or keeping a private internal calendar that combines hearings with non-judicial appointments.

Moreover, UYAP must be accessed with an electronic signature (e-imza) or a mobile signature, and the editor used for preparing documents (UYAP Editor) is a separate Java-based desktop application [5]. This means that, in daily practice, lawyers usually keep their own client list in a spreadsheet and their hearings in a personal calendar, while only court-related operations are performed through UYAP. The aim of the present project is therefore not to replace UYAP but to complement it with a private, firm-level management system that organises clients, cases and hearings in a single, modern web interface, while UYAP remains the official channel for the communication with the courts.

1.2. Study and Theory

For the project, "Next.js" (for the front-end), "Go" (for the backend services) and "PostgreSQL" (for the relational database) were used. Next.js was chosen because it is one of the most suitable frameworks for building modern, server-rendered React applications with built-in routing and good performance. Moreover, Go was chosen because it is a fast, statically typed language with excellent support for building scalable HTTP APIs. PostgreSQL was preferred over a NoSQL solution because the data of a law firm — clients, cases, hearings — has clear relationships, and a relational schema with foreign keys gives strong consistency guarantees, which is desirable when dealing with legal records. Some information about these technologies is given in the following.

Next.js: Next.js is an open-source React framework developed by Vercel. It provides features such as file-based routing, server-side rendering (SSR), static site generation (SSG), API routes, image optimisation, incremental static regeneration and built-in TypeScript support [7]. On top of React's component model, Next.js offers additional features that enhance productivity when building production-ready web applications, such as:

- A flexible file-based routing system using the App Router
- Server Components and client Components separation
- A unified environment for building both frontend pages and backend API endpoints
- Fast Refresh for instant feedback during development

- Built-in optimisation for images, fonts and scripts
- First-class TypeScript support

Go: Go (also known as Golang) is an open-source programming language designed at Google. It is well known for its simplicity, strong concurrency model based on goroutines, and very fast compilation times. For this project, Go is used to implement the REST API layer between the Next.js frontend and the PostgreSQL database, exposing endpoints for authentication, client management, case management and the hearing calendar [9].

PostgreSQL: PostgreSQL is a powerful, open-source object-relational database system. It supports advanced data types, indexing strategies and full ACID compliance. The data of the application — users, clients, cases and hearings — is stored in normalised tables connected to each other through foreign-key relationships, which makes it possible to enforce referential integrity at the database level [8].

2. AN OVERVIEW OF NEXT.JS, GO AND POSTGRESQL

For this application or system, basically three technologies were used. The first is “Next.js” for designing the user interface and the navigation logic of the application; the second is “Go” for the backend REST API that contains the business logic; the third is “PostgreSQL” for storing all data in a relational and consistent way. In the following parts “2.1”, “2.2” and “2.3”, some brief information is given about these three technologies.

2.1. Next.js

Next.js is the React framework that has been used for the front-end side of this project. It is built on top of React and provides production-grade features out of the box [7]. On top of React's powerful component model and developer tools, Next.js offers even more features that enhance productivity when building modern web apps, such as:

- A flexible file-system based routing using the App Router
- Server-Side Rendering and Static Site Generation
- API routes and middleware that run on the same project
- Fast Refresh to push changes to the running app instantly
- Image, font and script optimisation built-in
- Extensive support for TypeScript and ESLint
- Built-in support for environment variables and deployment to Vercel or any Node.js host [7]

2.1.1. Project Structure

Each project in Next.js contains one or more route segments organised under the “app” directory, together with shared resources. Types of folders include:

- Page and layout folders under app/
- Reusable components under components/
- Utility libraries under lib/

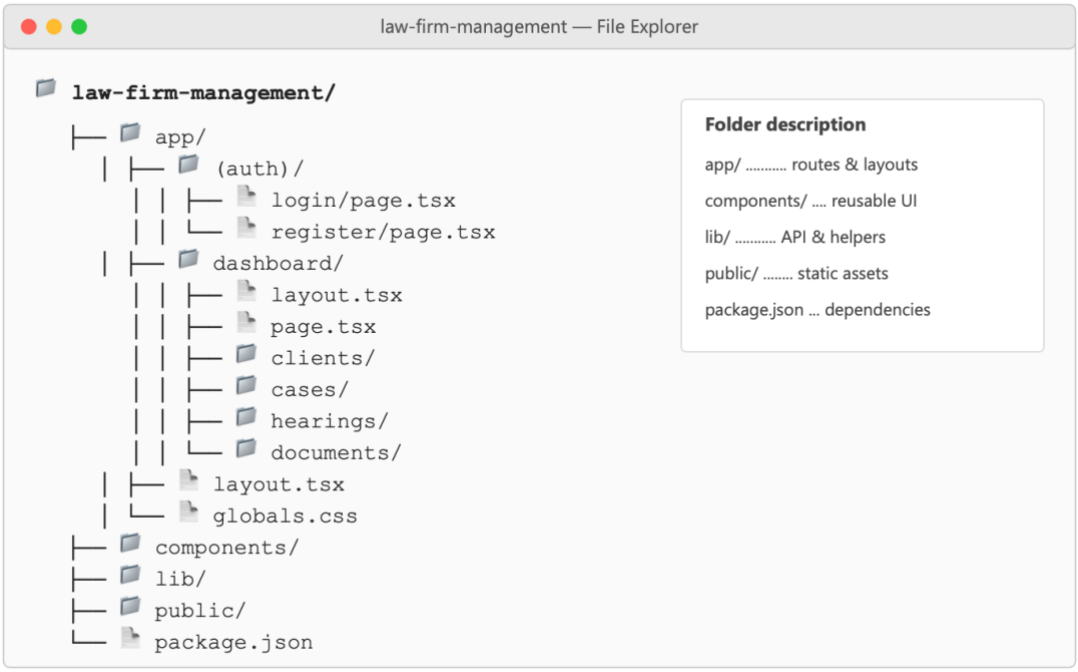

Figure 2.1 The Project Files in a Next.js Application

By default, the project is organised by routes to provide quick access to the application's key files. The App Router maps each folder under "app" to a URL segment, which makes the relationship between the directory tree and the live URLs easy to follow [9].

All build configuration files are visible at the top level under the project root and each Next.js project contains the following folders:

- app: Contains route segments, layouts, pages and route handlers.
- components: Contains reusable React components shared across pages.
- lib: Contains utility code such as API clients, validation schemas and authentication helpers.
- public: Contains static assets such as images, icons and fonts [9].

The Next.js project structure on disk reflects this layout directly. To see the actual file structure of the project, you can open the project folder in any IDE such as Visual Studio Code or WebStorm [7].

You can also customise the view of the project files to focus on specific aspects of your application development. For example, when an error occurs, the development server displays an error overlay containing links to the source files containing any recognised type or runtime errors, such as a missing prop in a component [7].

2.1.2. The User Interface

The Next.js development environment is made up of several logical areas identified in the following figure.

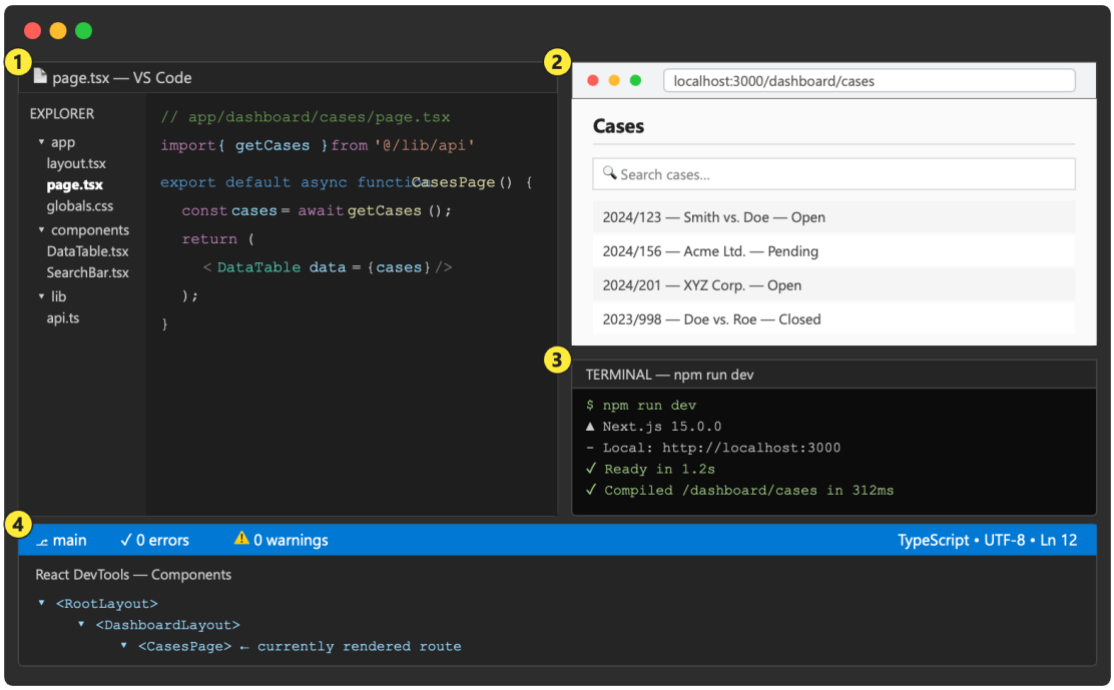

Figure 2.2 A Typical Next.js Development Window

1. The browser preview shows the rendered application, including hot module replacement when code is saved.
2. The IDE editor window is where you create and modify React components and TypeScript files.
3. The terminal runs the development server (npm run dev) and prints request logs.
4. The component tree (in React DevTools) gives access to props, state and rendered hierarchy.
5. The status bar of the IDE displays the status of the project and any warnings or messages [7].

You can organise the development environment to give yourself more screen space by hiding or moving panels. You can also use keyboard shortcuts to access most IDE features. At any time, you can search across your source code, components, route handlers and elements of the user interface, by using the global search shortcut of your editor. This can be very useful if, for example, you are trying to locate a particular component that you have forgotten the name of [7].

2.2. Go (Backend)

Go is the language that has been used to implement the backend of this project. The backend exposes a REST API that the Next.js frontend consumes via HTTP requests. All business rules — such as "a hearing cannot be scheduled in the past" or "only the assigned lawyer can edit a case" — live in the Go layer rather than in the frontend, which keeps the system secure and consistent.

2.2.1. REST API Architecture

The Go backend is structured into five clearly separated layers. The route layer defines all HTTP endpoints using Go's standard "net/http" package with "http.ServeMux". The handler layer parses JSON requests and formats responses. The service layer enforces all business rules. The repository layer defines interfaces for data access. The database layer provides PostgreSQL implementations of those interfaces using the "pgx/v5" driver and connection pooling via "pgxpool". HTTP requests flow as follows: Router → Middleware → Handler → Service → Repository → PostgreSQL. This separation improves code readability, isolates responsibilities, and makes each layer independently testable. API request and response models are defined in a dedicated "dto" package, keeping the domain models decoupled from the external API contract. The project directory structure reflects this layering directly: "internal/routes", "internal/handlers", "internal/services", "internal/repositories", "internal/database", and "internal/models" each map to a single architectural layer.

2.2.2. Authentication and JWT

Authentication in the system is based on JSON Web Tokens (JWT), implemented using the "github.com/golang-jwt/jwt/v5" library. When a user logs in successfully, the Go backend signs a token containing the following claims: "user_id", "office_id", "role", "sub", "iat", and "exp". For each subsequent request, the frontend attaches the token in the "Authorization: Bearer" header, and an auth middleware validates it before passing the request to the corresponding handler. The token expiry is configurable via the "ACCESS_TOKEN_TTL_SECONDS" environment variable. User passwords are never stored in plain text; they are hashed using the bcrypt algorithm via "golang.org/x/crypto/bcrypt" before being stored in the database, and the bcrypt comparison is performed at login time. The system also uses an office-based multi-tenancy model in which the "office_id" claim carried inside the token is used to scope all data queries to the user's active office.

2.3. PostgreSQL

PostgreSQL is a relational database system that supports multiple platforms and is widely used in production. All the data is stored in normalised tables and any change in the data is committed transactionally, which means that either all related changes are applied or none of them is. This allows building reliable, consistent applications easily with minimal effort [8].

2.3.1. Relational Schema

The application database consists of eight core tables. The “offices” table represents law firms or offices in the system. The “users” table stores all registered users. The “office_users” table models the many-to-many membership relationship between users and offices, storing the role (“owner”, “admin”, “lawyer”, “staff”) and the active/inactive status of each membership. The “office_invites” table stores time-limited, usage-counted invitation codes for joining an office. The “clients”, “cases”, “tasks”, and “events” tables hold the main domain entities, all of which are scoped to an office. Two PostgreSQL extensions are used: “pgcrypto” provides the “gen_random_uuid()” function for UUID generation, and “citext” enables case-insensitive comparison on email columns. All core tables contain a “deleted_at” column for soft-delete support; a record is considered active when “deleted_at IS NULL”. Partial indexes on this condition are defined in migration files to maintain query performance. Migration files are stored in the “migrations” directory and are run automatically on application startup using “golang-migrate/migrate”. Below is an example of the relational schema.

2.3.2. CRUD Operations

Before getting into the application code, some basic information about performing CRUD operations on the relational database should be known, because all these concepts have to be combined together to build a meaningful application with PostgreSQL as backend. In order to perform any operation on the database — whether it is read or write — you need to obtain a connection to the database first [8]. The following pseudo-code gives a reference to a PostgreSQL connection pool that can be reused across requests.

3. DESIGN AND DEVELOPMENT OF AN INTEGRATED CASE AND LAW FIRM MANAGEMENT SYSTEM

For this project, firstly, some discussions have been made with practising lawyers and law firm employees to understand the basic problems about case management, calendar handling and the user interfaces of legal software. Then, according to these discussions, some results appeared, and the user interfaces of this project have been created according to these results. These discussions and their results are given below.

3.1. Discussions with Lawyers

In these discussions about the web application, the design of the layout and some ideas have been examined. The main discussions were especially about the “Login and Authentication”, “Client Management”, “Case Tracking” and “Hearing Calendar” parts of the application. The reason is that, for the login part, the lawyer wants to log in or be added to the firm without confusion. For client management, the lawyer wants to record contact details, identification numbers and notes about the client in a single place. For case tracking, it must be easy to see the current status of every active case, the assigned lawyer and the next hearing date. For the calendar, missing a hearing date is one of the worst things that can happen in a law office, so the system must show the upcoming hearings of the next days very clearly.

In the application, there are three pages for the login part named “Login”, “Register” and “Forgot Password?”, and there are three main modules for the daily workflow named “Client Management”, “Case Tracking” and “Hearing Calendar”. Also, the lawyers want to find a specific client or case quickly, just by typing a few characters. That is why a global search bar has been added to the top navigation, present on every page.

3.2. Application Design

In the application, there are several pages. The first of them is the “Login” page. On this page, a user can log in or register to the system by providing their email and password. There is also a possibility for the user to set a new password if they have forgotten the old one. Moreover, the application can remember the user thanks to the JWT token stored in an HTTP-only cookie, which provides the user with the ability to be logged in automatically when they return to the application. The second page is the “Dashboard”, which gives a quick overview of the number of active cases, upcoming hearings of the week and recently added clients. The third page is “Client Management”, where the user can list, search, add, edit and archive clients of the firm. The next page is the “Case Tracking” page. This page lists all cases of the firm with their court, file number, status (open, pending, closed) and assigned lawyer; the user can click on a case to see its full detail page. Then there is the “Hearing Calendar” page, where the user can see all upcoming hearings in a monthly or weekly calendar view, add a new hearing for a specific case, or mark a hearing as completed.

3.3. Design of User Interface

According to the discussions with lawyers in part 3.1 and according to part 3.2, several user interfaces have been created. In this system, there are five parts for designing the user interface, namely “Database”, “Login and Authentication”, “Client Management”, “Case Tracking” and “Hearing Calendar”. Moreover, a fully custom server-side authentication process based on JWT has been developed for the web application.

3.3.1. Database (PostgreSQL)

To keep records of offices, users, clients, cases, tasks and events, the system uses a PostgreSQL relational database. PostgreSQL is a robust and ACID-compliant database well suited for an application that handles legally sensitive information. A brief explanation about PostgreSQL is given in part 2.3. The multi-tenancy isolation unit in the database is the “office_id” column. Every client, case, task and event record is linked to an office, and the repository layer filters all queries by the calling user’s active office. This design ensures that data belonging to one law firm is never visible to users of another firm. A soft-delete strategy is used throughout: rather than physically removing records, the application sets a “deleted_at” timestamp, and partial indexes on “deleted_at IS NULL” keep read queries efficient. Also, a figure about this part is given below.

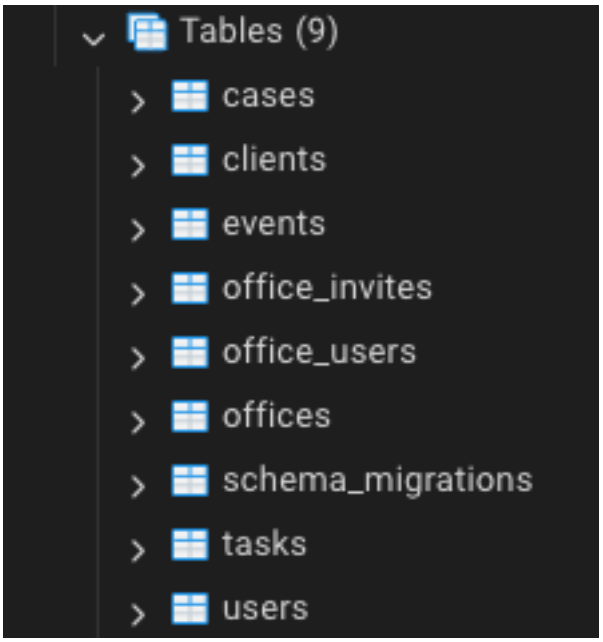

Figure 3.1 PostgreSQL Database for the Application

In the PostgreSQL database, each user can belong to multiple offices through the “office_users” join table, which stores the role and membership status. Each office can have multiple users. Clients, cases, tasks and events are all scoped to an office via a foreign key. A case can be linked to a client and assigned to a user who is an active member of the same office. Tasks can be linked to a case and carry a due date that is surfaced in the unified calendar view. Events are standalone calendar entries that may

optionally be associated with a case. Hearing dates are stored directly on case records, and all three sources — case hearing dates, task due dates, and calendar events — are merged by the calendar module into a single chronological view. Foreign keys at the database level guarantee referential integrity even if the application layer has a bug.

3.3.2. Login and Authentication

In the system, a user must complete the following steps before being able to access the main application modules:

- User registers by providing their name, email, password and phone number. The password is hashed with bcrypt before being stored.
- User completes onboarding by either creating a new office (becoming its "owner") or by joining an existing office using a time-limited invitation code (joining as "staff"). The onboarding operation runs inside a database transaction.
- User logs in with email and password. The backend verifies the bcrypt hash, checks active office membership, and issues a signed JWT access token.

For the above steps, three API endpoints have been implemented: "POST /auth/register", "POST /auth/onboarding", and "POST /auth/login". The onboarding endpoint accepts either an office creation payload or an invitation code, handling both flows. The "GET /auth/me" endpoint returns the currently authenticated user's information. A figure showing the login and authentication screens of the application is given below.

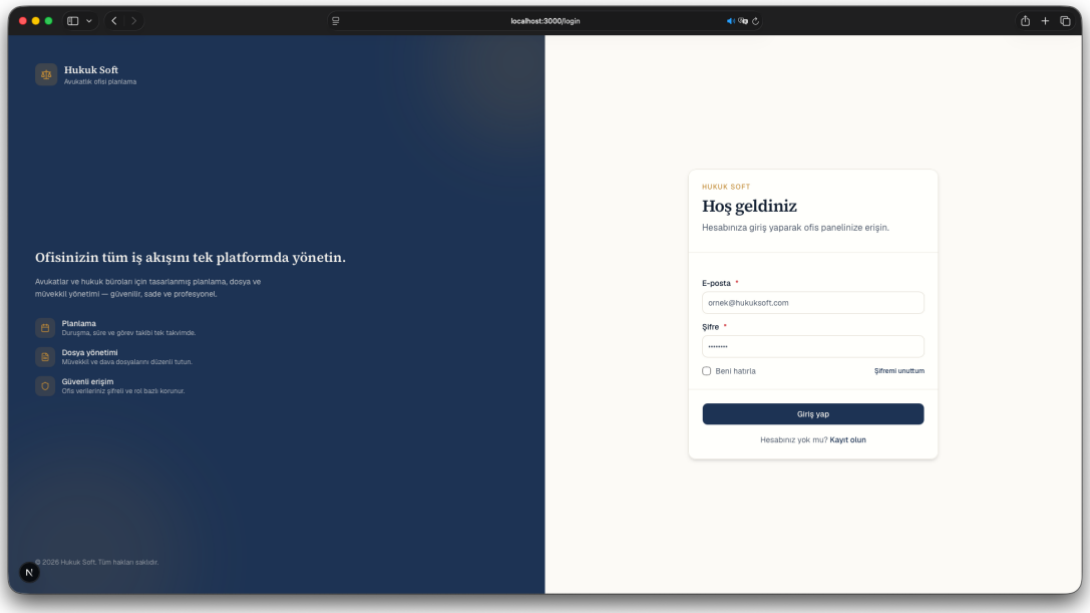

Figure 3.2 Login Part of the Application

3.3.3. Client Management

After login, the user is taken to the dashboard. From the side menu, the user can navigate to the “Client Management” module. On this page, the user can see a paginated list of all clients of the firm, search by name, national identifier or phone number, and click on a client to see their detail page. The detail page contains contact information, identification number, address, the list of cases that belong to the client, and a free-text notes area where the lawyer can keep observations about the client. There are buttons for “New Client”, “Edit Client” and “Archive Client”. The archive operation is preferred over delete, so that historical data is preserved for legal reasons. A figure about this part is given below.

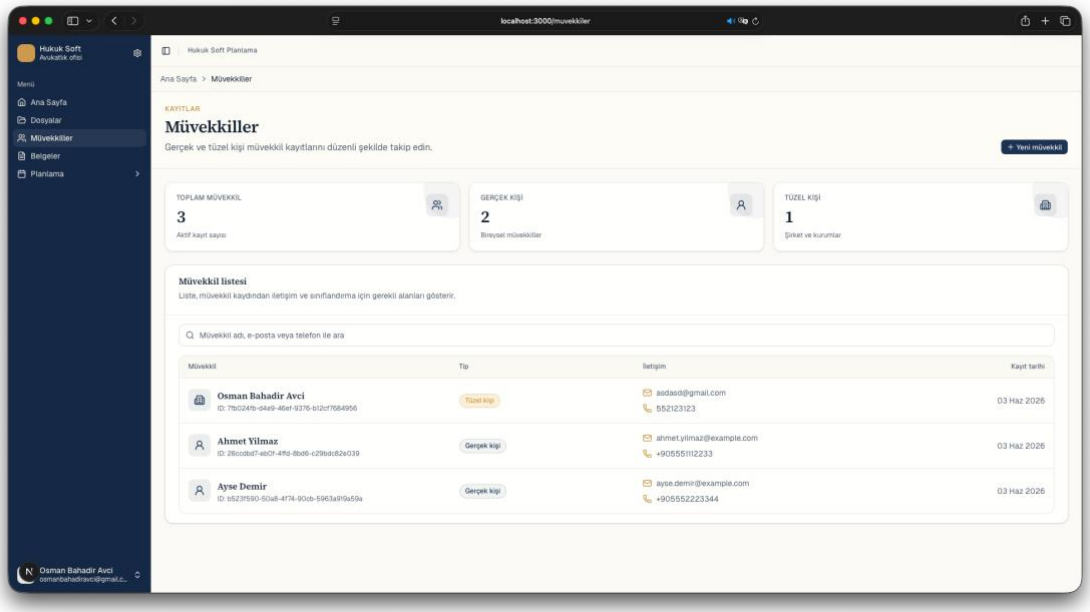

Figure 3.3 Client Management Part of the Application

3.3.4. Case Tracking

This part is one of the most important parts of this application (or system). This part consists of the case list and the case detail page. To allow lawyers to track their work, the case list shows the file number, the related court, the case type (such as “civil”, “criminal”, “administrative”, “enforcement”), the current status (open, pending, closed) and the assigned lawyer. The list can be filtered by status, court and assigned lawyer. The user clicks on a case to open its detail page. On the detail page, all information related to the case is shown: client, opposite party, court, file number, case type, status, assigned lawyer(s), upcoming hearings, attached documents and a chronological activity log. A figure about this part is given below.

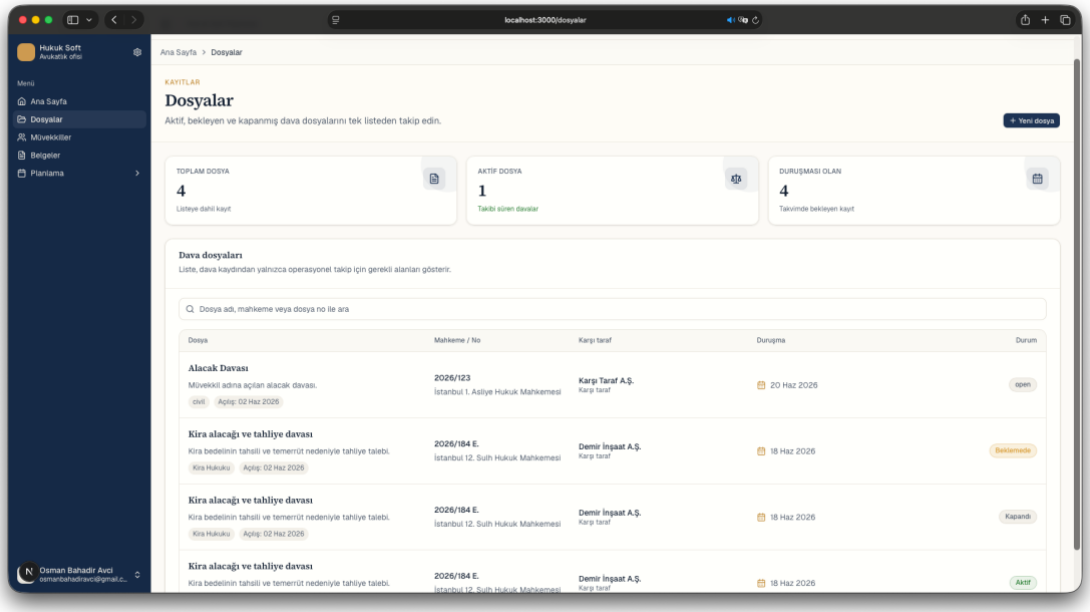

Figure 3.4 Case Tracking Part of the Application

After a case is opened, the user can navigate from the side menu to add a new hearing or change the status of the case to “closed” when the case is finalised.

3.3.5. Hearing Calendar

This part is the last part of the application. In the “Hearing Calendar”, the user can see their upcoming hearings in a monthly view, a weekly view or as a list grouped by day. Hearings of the current week are highlighted, and a hearing that is less than 24 hours away triggers a colour change in the badge so that the lawyer is reminded visually. Clicking on a hearing opens a small popup with the case file number, the court, the time, the courtroom and a free-text note that can be edited. There are also two buttons named “Mark as Completed” and “Postpone”. If the user presses “Postpone”, a small form opens where a new date and time can be chosen, and the system asks for confirmation before saving. After confirmation, the change is logged in the activity history of the related case so that there is a permanent record. A figure about this part is given below.

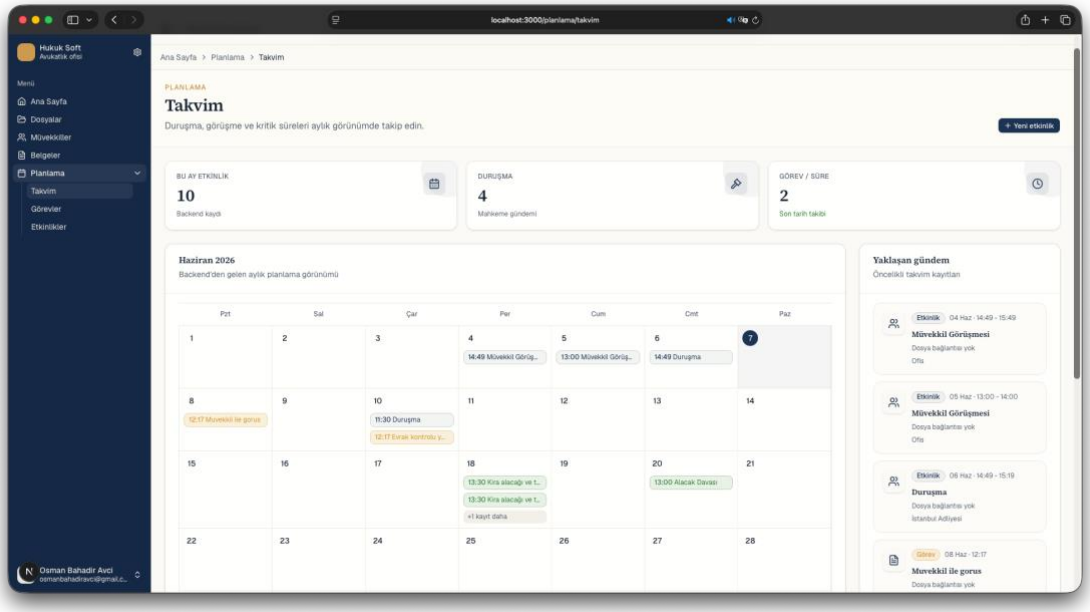

Figure 3.5 Hearing Calendar Part of the Application

3.4. Implementation

For the project, three primary technologies have been used for the main layers of the stack, complemented by several supporting libraries. These technologies and libraries are listed below.

- Next.js for the front-end web application (authentication flows, dashboard, client/case/task/event management modules)
- Go for the backend REST API, business logic, authentication, onboarding, and the unified calendar module
- PostgreSQL for the relational database, with “pgx/v5” as the driver, “pgxpool” for connection pooling, and “golang-migrate/migrate” for schema versioning

The backend also includes an auth middleware that validates JWT tokens and injects user context into requests, a CORS middleware configurable via environment variables, a request logging middleware that records method, path, status code, response size, remote address, user agent and duration, and a recovery middleware that catches panics and returns a “500 internal_error” response instead of crashing. Graceful shutdown is implemented by listening for “SIGINT” and “SIGTERM” signals. Configuration is loaded entirely from environment variables, with “.env” file support via “godotenv”. Structured logging is performed using Go’s built-in “log/slog” package. Also, an organisation scheme for this application is given on the next page.

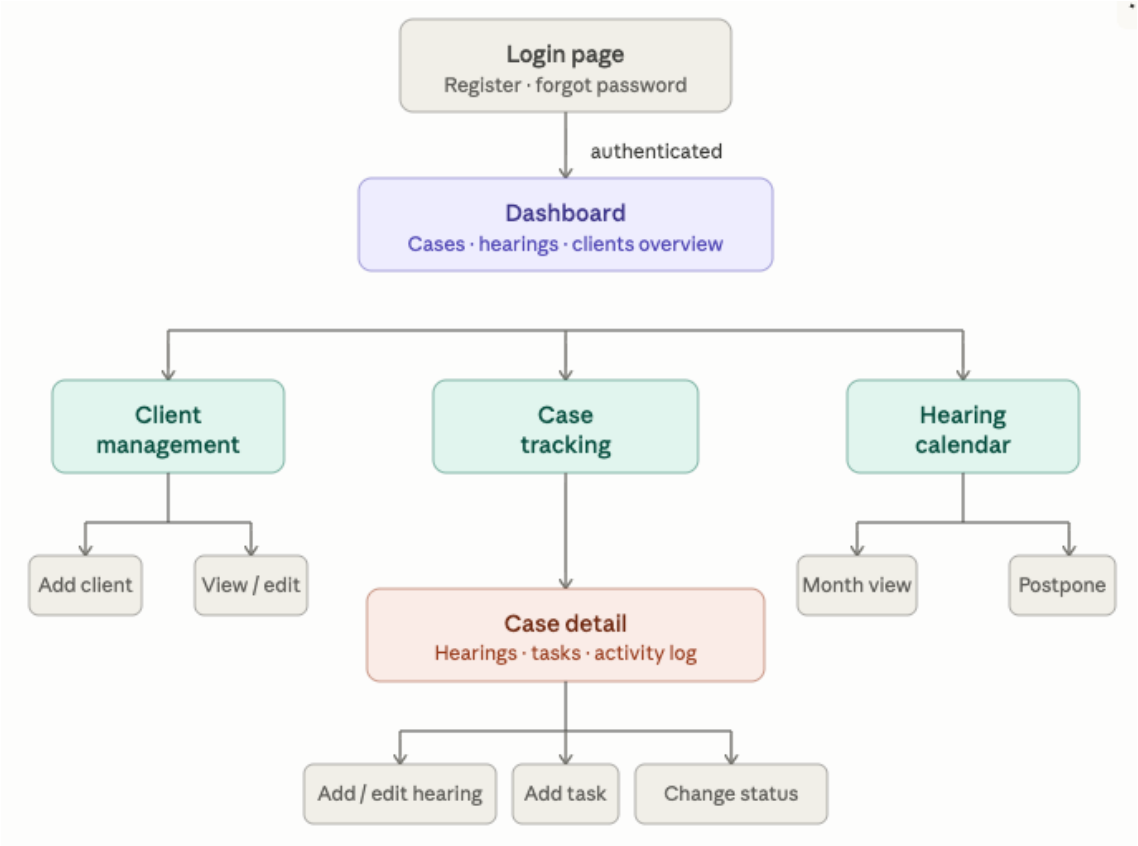

Figure 3.6 Organization Scheme of the Project

4. CONCLUSION

In this study, the design and development of a backend system for an integrated case and law firm management application is discussed. The system was implemented in Go with a PostgreSQL database, following a layered architecture and a multi-tenant design. Analysis of the implemented codebase yielded the following key results.

I. A layered backend architecture — route, handler, service, repository, and database layers — was successfully implemented, cleanly separating concerns and making the codebase maintainable.

II. JWT-based authentication with bcrypt password hashing and office-scoped multi-tenancy isolation were implemented as core security features of the system.

III. A unified calendar module was built that merges three domain sources — case hearing dates, task due dates, and standalone calendar events — into a single chronologically sorted view without requiring a dedicated calendar table.

IV. A soft-delete strategy with partial indexes was applied across all core tables, preserving historical data for legal and audit purposes while maintaining query performance.

V. Identified limitations include the absence of automated tests, incomplete role-based access control at the endpoint level, missing update/delete endpoints for several domain entities, and no Dockerfile or CI/CD pipeline.

VI. Security gaps such as the absence of rate limiting, brute-force protection on the login endpoint, and the lack of a refresh-token mechanism were identified as areas for future improvement.

VII. The project builds successfully with “go test ./...”, confirming that all packages compile without errors. However, the absence of test files means this result only validates compilation, not correctness.

In the project, the operational needs of law firms — managing clients, cases, tasks and calendar events from a single backend service — have been addressed through a Go and PostgreSQL stack built on layered architecture, JWT-based authentication, and office-scoped multi-tenancy. The most distinctive design element is the unified calendar module, which surfaces case hearing dates, task due dates, and standalone calendar events through a single “GET /calendar” endpoint without requiring a dedicated calendar table. The project is implemented using Next.js for the front-end, Go for the backend REST API, and PostgreSQL for the database. Future work may include completing role-based endpoint permissions for the “owner”, “admin”, “lawyer”, and “staff” roles, adding update and delete endpoints for all domain entities, building a unit and integration test suite with a containerised test database, adding Dockerfile and CI/CD pipeline support, introducing rate limiting and refresh-token mechanisms, and configuring HTTP server timeouts and request body size limits for production readiness.

REFERENCES

[1] Bilişim ve Hukuk, “Ulusal Yargı Ağı Projesi - UYAP - Nedir?”, https://www.bilisimvehukuk.net/icerik/ulusal-yargi-agi-projesi-uyap-nedir/ (10.04.2026)

[2] TRT Türk, “UYAP (Ulusal Yargı Ağı Bilişim Sistemi) nedir?”, https://www.trtturk.com.tr/faydali-bilgiler/uyap-ulusal-yargi-agi-bilisim-sistemi-nedir_70 (10.04.2026)

[3] Wikipedia, “UYAP”, https://tr.wikipedia.org/wiki/UYAP (12.04.2026)

[4] Eralp Avukatlık Bürosu, “Konu 02 – UYAP – Ulusal Yargı Ağı Bilişim Sistemi”, https://www.eralp.av.tr/ders-02-uyap-ulusal-yargi-agi-bilisim-sistemi/ (12.04.2026)

[5] T.C. Adalet Bakanlığı, UYAP Resmi Web Sitesi, https://www.uyap.gov.tr/ (15.04.2026)

[6] Ekotürk, “UYAP nedir ve nasıl çalışır? UYAP’ın temel özellikleri ve faydaları”, https://www.ekoturk.com/haberler/uyap-nedir-ve-nasil-calisir-uyapin-temel-ozellikleri-ve-faydalari/ (15.04.2026)

[7] Next.js Documentation, https://nextjs.org/docs (20.04.2026)

[8] PostgreSQL Documentation, https://www.postgresql.org/docs/ (22.04.2026)

[9] Go Programming Language Documentation, https://go.dev/doc/ (25.04.2026)