TED UNIVERSITY

Faculty of Engineering

Computer Engineering Department

CMPE_492 Senior Design Project 2

LOW-LEVEL DESIGN REPORT

Name of the Project: JustiWise

Web Page URL: https://finalprojectjustiwise.github.io/CMPE_491/

Team Members:

Irmak Orhan

Ahmet Engin Büyükdığan

Aslı Algın

Ali Fuat Dündar

Supervisor: Tansel Dökeroğlu

Jury Members: Fırat Akba, Ayşe Yasemin Seydim

CONTENT

1. Introduction

Legal processes often suffer from inefficiencies due to outdated and manual workflows, leading to prolonged case resolution times and accessibility barriers. By offering automated document analysis, case evaluations, and interactive training for legal professionals and students, JustiWise is an AI-powered legal platform that aims to revolutionize the legal industry.

The platform ,which is accessible via a web-based interface, is made up of several subsystems that cooperate to provide a good experience for various user roles, such as clients, attorneys, and law students. JustiWise uses natural language processing (NLP) and artificial intelligence to:

- Automate repetitive processes like contract reviews, case summary, and legal research to streamline legal procedures.
- Enhance accessibility to legal services by providing AI-powered legal consultations that guide users through basic legal questions.
- Improve legal education and training by using interactive case simulations and mock trials to support in the development of practical skills in professionals and law students.
- Ensure compliance with legal standards such as GDPR and KVKK by implementing robust data protection measures and ethical AI practices.

1.1 Object design trade-offs

Several **trade-offs** were considered in the design process to balance performance

- **Security vs. Flexibility:** Protecting user data is a top priority, requiring encryption and strict access controls. However, excessive security measures can create usability challenges. JustiWise ensures a balance by implementing **role-based access control (RBAC)** and secure authentication mechanisms while maintaining ease of use.
- **Accuracy vs. Performance:** Response time may be impacted by the complex calculations needed for AI-powered legal analysis. In order to overcome this, JustiWise uses NLP models that have been tuned to process case documents quickly without sacrificing accuracy.
- **Scalability vs. Complexity:** The system needs to be responsive enough to accommodate an increasing user base. A microservices design was used to do this, enabling distinct subsystems to scale independently in response to demand.

1.2 Interface documentation guidelines

In this report, each class is described with its name, definition, attributes, and methods. Class interface documentation uses the table format shown below:

Class Name	Explanation of the class
Attributes	Type, name
Methods	Return type,MethodName(parameters)

Example : Handles user registration, authentication, and role management

Class Name	login page part
Attributes	String email - user's email String passwordHash - Encrypted password
Methods	Boolean CreateUser(String name,String email, String password) user getUser(String email)

1.3 Engineering standards (e.g., UML and IEEE)

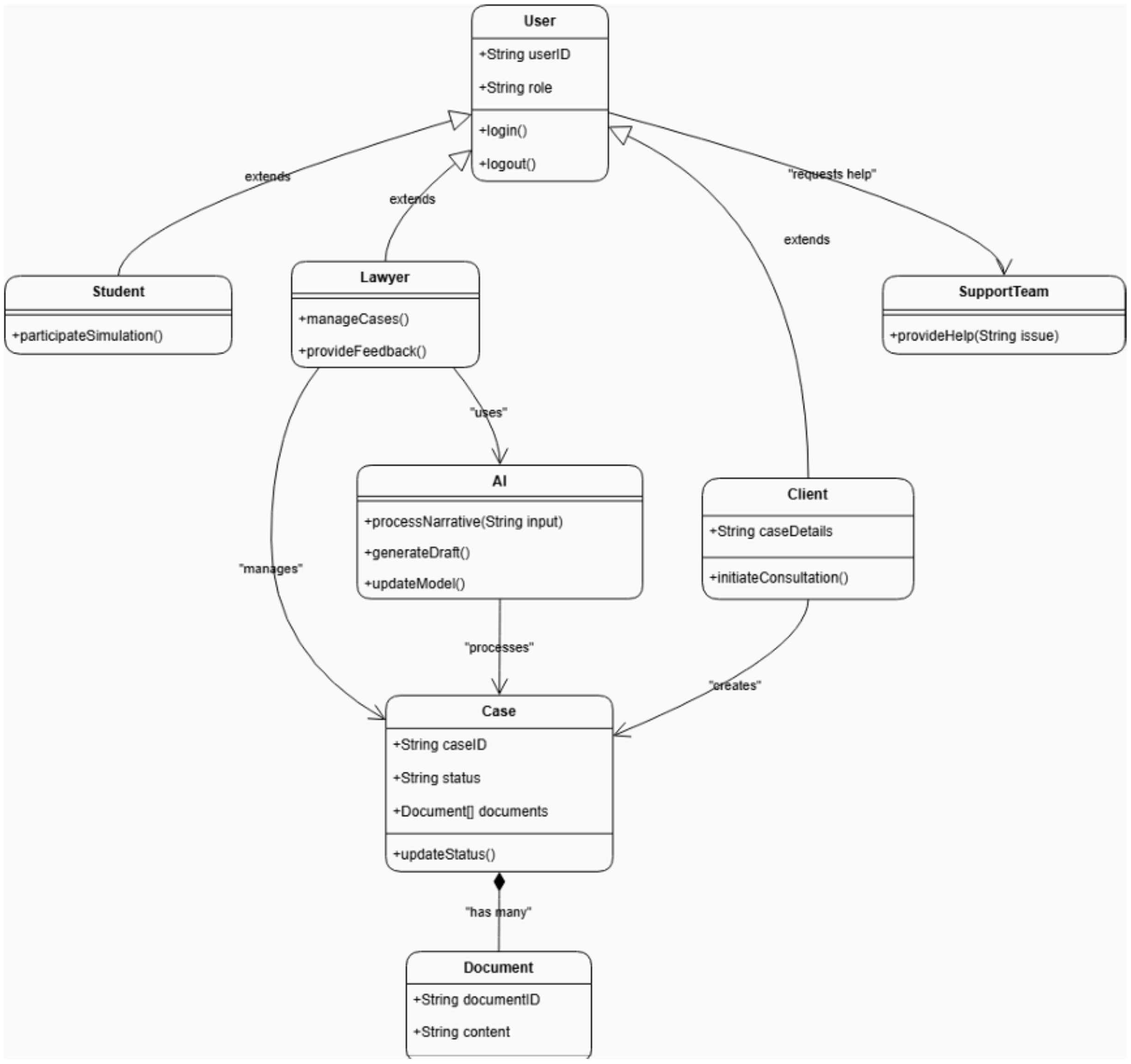
The development of JustiWise follows software engineering standards and architectural patterns to ensure scalability, maintainability, and security (trade offs).

- **Microservices Architecture:** By dividing the system into individual microservices, modular development and effective demand-driven scalability are made possible.
- **MVC (Model-View-Controller):** The **backend follows the MVC pattern**, ensuring clear separation between data management, business logic, and user interfaces.
- **RESTful API Design:** Communication between the frontend and backend is handled through REST APIs, enabling interoperability and structured data exchange.
- **Natural Language Processing (NLP):** AI-based document analysis relies on NLP techniques, using frameworks such as **spaCy** and **Transformers** for text processing.

Security and Compliance:

- User authentication is managed through **OAuth 2.0** and **JWT (JSON Web Tokens)** for secure session handling.
- Data privacy measures comply with **GDPR** and **KVKK** regulations, ensuring secure storage and processing of legal data.

Object and Class Model:



1.4 Definitions, acronyms, and abbreviations

This section defines key terms and abbreviations used throughout the report, with a focus on both general technical terms and project-specific terminology:

General Terms:

- **AI (Artificial Intelligence):** The ability of machines to simulate human intelligence.
- **NLP (Natural Language Processing):** AI techniques for understanding and processing human language.
- **RBAC (Role-Based Access Control):** Security framework that restricts system access based on user roles.
- **API (Application Programming Interface):** A set of functions enabling interaction between different software components.
- **JWT (JSON Web Token):** A compact, URL-safe means of representing claims to be transferred between two parties.
- **GDPR (General Data Protection Regulation):** European data privacy law.
- **KVKK:** Turkish Personal Data Protection Law.

Project-Specific Terms:

- AI-Powered Legal Consultation: A module that uses natural language processing (NLP) and legal data processing to automatically advise users on legal matters.
- Legal Case Analyzer: An AI-powered program that examines court records and extracts important details about a case.
- Virtual trial simulation: interactive learning environments that allow professionals and law students to participate in simulated court cases.

Roles for Users:

- Client: People looking for legal advice via the AI-powered consultation services offered by JustiWise.
- Lawyer: A legal expert who evaluates AI-generated case evaluations and offers knowledgeable legal counsel.
- Law Students: Users taking part in educational legal training and case simulations.

2. Packages

The system is structured into three primary packages:

1. Client Package (React.js, TypeScript, Vercel)

- **Components:** UI Screens, API Clients, User Authentication
- **Features:**

    - Interactive legal simulation environment
    - AI-assisted virtual lawyer interface
    - User authentication and role-based access (clients, lawyers, students)
    - Integration with backend services via REST API

2. Backend Package (Java Spring Boot, PostgreSQL, Docker)

- **Services:**
    - User Management (Clients, Lawyers, Students)
    - Case Management and Legal Document Storage
    - AI Integration for Legal NLP Processing
- **Layers:**
    - **Controller:** Handles API requests
    - **Service:** Business logic for user cases, document processing
    - **Repository:** Database interactions using JPA & PostgreSQL
    - **Model:** Data structures and entities
    - **Security:** Authentication & Authorization (Spring Security, JWT)
- **Infrastructure:**
    - RESTful APIs for communication with the frontend and AI package
    - Dockerized microservices for scalability

3. AI Package (Python, Llama 3, FastAPI)

- **Components:**
    - AI-powered **legal document analysis** (contract summarization, legal language translation)
    - AI-assisted **case simulation** for student training
    - **NLP models** for legal guidance and automation
- **API Interface:**
    - FastAPI-based service to interact with backend
    - Real-time AI processing for legal scenarios

3. Class Interfaces

3.1 Backend Class Documentation

ClassName	**Explanation of the Class**
User	Represents a user of the system, including lawyers, clients, and students.
Role	Defines the different roles available in the system (client, lawyer, student).
UserDTO	Data Transfer Object for User, used to transfer data between layers.
UserRepository	Data Access Layer for User, manages database interactions.
UserService	Interface defining user-related business logic.
UserServiceImpl	Implementation of UserService, handles user registration and retrieval.
UserController	API Controller for user authentication and registration.
UserMapper	Converts between User entity and UserDTO using MapStruct.
SecurityConfig	Configures Spring Security, JWT authentication, and role-based access.
JwtUtil	Manages JWT token creation, validation, and extraction.
AuthController	Handles user login and JWT token generation.
AIController	Analyze the cases.

3.1.1 Backend Class Details

Class Name	**User**
Attributes	id, email, password, role (Enum: clıent, lawyer, student)
Methods	getId(), setId(Long id), getEmail(), setEmail(String email), getPassword(), setPassword(String password), getRole(), setRole(Role role)

Class Name	**UserDTO((Data Transfer Object)**
Attributes	İd, email, password, role
Methods	getId(), setId(Long id), getEmail(), setEmail(String email), getPassword(), setPassword(String password), getRole(), setRole(Role role)
Class Name	**UserRepository (Data Access Layer)**
Methods	findByEmail(String email)

Class Name	**UserService (Service Interface)**
Methods	registerUser(UserDTO userDto), getAllUsers()
Class Name	**UserServiceImpl (Service Implementation) - Implements UserService**

Methods	registerUser(UserDTO userDto), getAllUsers()

Class Name	**UserController (REST API)**
Methods	registerUser(UserDTO userDTO), getAllUsers()

Class Name	**UserMapper (Entity-DTO Mapper)**
Methods	toDto(User user), toEntity(UserDTO userDTO)

Class Name	**SecurityConfig (Spring Security Configuration)**
Methods	registerUser(UserDTO userDTO), passwordEncoder(), authenticationManager(AuthenticationConfiguration authConfig), securityFilterChain(HttpSecurity http)

Class Name	**JwtUtil (JWT Token Manager)**
Methods	generateToken(String username), extractUsername(String token), validateToken(String token, String username)

Class Name	**AuthController (Authentication Controller)**
Methods	authenticateUser(UserDTO userDTO)

Class Name	**AIController (AI Integration)**
Methods	analyzeCase(String text)

3.2 Frontend Class Documentation

PackageName	**Explanation of the Package**
Components	The Component package contains reusable and modular parts. This avoids code duplication, makes the ui more modular and more manageable.
Service	Service pack data is more manageable and is used to separate business logic. Manages API requests. It allows components to be independent of the data source, so that components only deal with visual operations.
Pages	The Pages package is used to organize screens/pages. Each page is usually a combination of several components.
Router	The router package manages transitions between pages, screens, components and prevents ineffective access.
Helper	Helper package contains functions to help other packages to be used. For example for error handling.
Model	The model package is used to bind the information in API requests or responses to the object. This makes data management and access easier.

3.2.1 Backend Class Details

Role.ts (Model)

Role(Type)

Attributes	**Type & Name**
id?	**number**
rolename	**string**

Case.ts (Model)

Case(Type)

Attributes	**Type & Name**
id	number
title	string
description	string
clientId	clientId
lawyerId?	number
documents	string[]
createdAt	Date
updatedAt	Date

FilteredCase= Partial<Case>

UpdateCase=Partial<Omit<Case,"id">>

NewCase=Omit<Case,”id”|” createdAt “| “updatedAt “>

User.ts(Model)

User(Type)

Attributes	**Type & Name**
id	number
name	string
roles	string[]
username	string
surname	string
email	string

LoginDTO(Type)

Attributes	Type & Name
username	string
password	string

LoginResponse(Interface)

Attributes	**Type & Name**
token	string

NewUser(Type)

Attributes	**Type & Name**
id	number
name	string

roles	number[]
username	string
surname	string
email?	string
password	string

UpdateUser =Partial<Omit<User,"id">>

FilteredUser = Partial<Omit<newUser,"id">>

authservice.ts(Service)

Attributes	**Type & Name**
api	string
token	string | null

Methods	**Return Type & Method Signature**
login(loginDTO: LoginDTO)	Promise<LoginResponse> login(loginDTO: LoginDTO)
register(createdNewUser: newUser)	Promise<User> createUser(createdNewUser: newUser)

userservice.ts(Service)

Attributes	**Type & Name**
api	string
token	string | null

Methods	Return Type & Method Signature
updateUser(updateUser: UpdateUser, id: number)	Promise<User> updateUser(updateUser: UpdateUser, id: number)
filterUser(FilteredUser: FilteredUser)	Promise<User[]> filterUser(FilteredUser: FilteredUser)
deleteUser(id: number)	Promise<User> deleteUser(id: number)
getAllUser()	Promise<User[]> getAllUser()

CaseService.ts(Service)

Attributes	Type & Name
api	string
token	string | null

Methods	Return Type & Method Signature
updateCase (updateCase: UpdateCase, id: number)	Promise<Case> updateCase (updateCase: UpdateCase, id: number)
filterCase(FilteredUser: FilteredUser)	Promise<Case[]>filterCase (filteredCase: FilteredCase)
deleteCase(id: number)	Promise<Case> deleteCase (id: number)
getAllCases()	Promise<Case[]> getAllCases()
getCaseById(id:number)	Promise<Case> getCaseById (id:number)

ErrorHandler.tsx(Helper)

Methods	Return Type & Method Signature
handleError ()	void handleError (error:any)
onSuccesfulCreate (entity:string)	void onSuccesfulCreate (entity string)
onSuccesfulEdit (entity:string)	void onSuccesfulEdit (entity string)
onFailedEdit (entity:string)	void onFailedEdit (entity string)
onFailedCreate (entity:string)	void onFailedCreate (entity string)

TokenPayload (Interface)

Attributes	Type & Name
exp	number
iss	string
aud	string
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/na	string
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/	string
http://schemas.microsoft.com/ws/2008/06/identity	string[]

Props(Type)

Attributes	Type & Name
children	ReactNode
roles	String[]

ProtectedRoute.tsx(Router)

Attributes	Type & Name
{children,roles}	Props

Methods	Return Type & Method Signature
render()	TSX.Element

Router.tsx(Router)

Methods	Return Type & Method Signature
createBrowserRouter()	void

Frontend Technologies

Technology	Purpose
Typescript	Frontend development language
React	Framework
Axios	Axios is the library used to make HTTP requests to 3rd party APIs. In the project, it will be used to make API calls from front-end to backend created with React.

React Router Dom	React router dom library is used to navigate between pages. In the project, for example, it will be used to navigate from the home page to the menu.
Materiul UI	Material UI is a pre-designed ui library. It contains many pre-designed components
Toastify	It is a component library for pop-up popup messages that provide feedback for what users have done. It contains information and alerts such as success, error, warnings or updates depending on the action taken.
Jwt-decode	Browser library to help decode Base64url encoded tokens.

3.3 AI Class Documentation

1. **ai.core Package:**

   Purpose: Contains core AI model interfaces and abstract classes, foundational data structures for AI processing, and base agent classes.

   Components: AIModel, Agent, LegalDataProcessor, TextSummarizer, LegalTermConverter, CaseAnalyzer.

2. **ai.consultation Package:**

   Purpose: Houses components for the AI-Powered Legal Consultation module, including the AI Avatar and related NLP models.

   Components: AIAvatarAgent, ClientNarrativeProcessor, LegalTermExtractor, FAQAnswerer.

3. **ai.document_processing Package:**

   Purpose: Contains components for Legal Document Processing functionalities like summarization and analysis.

   Components: DocumentSummarizationAgent, LegalDocumentAnalyzer, ContractClauseExtractor, JudgmentSummarizer.

4. **ai.research Package:**

   Purpose: Includes components for the Legal Research Agent, facilitating automated legal research tasks.

Components: LegalResearchAgent, QueryFormulator, DatabaseSearcher, ResearchResultSummarizer, SourceVerifier.

3.3.1 ai.core Package - Core AI Components

AIModel Interface

Definition: Abstract interface for all AI models used in the system, enforcing a standard process method.

Methods	**Return Type, Method Name(Parameters)**
	String, process(String input)
	void, loadModel()
	void, unloadModel()

Agent Abstract Class

Definition: Base abstract class for all AI Agents, defining core agent functionalities like task management and communication with AI models.

Attributes	**Type, Name**
	String, agentName
	AIModel, model
Methods	Return Type, Method Name(Parameters)
	void, performTask()
	void, communicate(String message)

LegalDataProcessor Interface

Definition: Interface for components that process legal data (text documents, client narratives), providing a standard processLegalText method.

Methods	**Return Type, Method Name(Parameters)**
	String, processLegalText(String legalText)

3.3.2 ai.consultation Package - AI Consultation Components

AIAvatarAgent Class

Definition: AI Agent responsible for managing the client legal consultation process, interacting with the ClientNarrativeProcessor and other relevant models.

Attributes	**Type, Name**
	String, agentName
	ClientNarrativeProcessor, narrativeProcessor
	LegalTermExtractor, termExtractor
	FAQAnswerer, faqAnswerer
Methods	**Return Type, Method Name(Parameters)**
	String, startConsultation(String clientInput)
	String, processClientInput(String input)
	String, getLegalTerminologySummary()
	String, answerFAQ(String question)

ClientNarrativeProcessor Class

Definition: Component responsible for processing the client's narrative input using NLP techniques (e.g., using a smaller, fine-tuned conversational Transformer model).

Attributes	Type, Name
	AIModel, nlpModel (e.g., fine-tuned DistilBERT for NLU)
Methods	**Return Type, Method Name(Parameters)**
	String, extractKeyInformation(String clientNarrative)

LegalTermExtractor Class

Definition: Component to extract legal terms and entities from processed client narratives using a fine-tuned NER model.

Attributes	Type, Name
	AIModel, nerModel (e.g., fine-tuned MobileBERT for Legal NER)
Methods	**Return Type, Method Name(Parameters)**
	List<String>, extractLegalTerms(String processedText)

FAQAnswerer Class

Definition: Component to answer frequently asked legal questions, potentially using a RAG approach with a smaller model and a legal FAQ knowledge base.

Attributes	Type, Name
	AIModel, qaModel (e.g., smaller QA Transformer model with RAG)
	LegalFAQDatabase, faqDB
Methods	**Return Type, Method Name(Parameters)**
	String, answerQuestion(String question)

3.3.3 ai.document_processing Package - Document Processing Components

DocumentSummarizationAgent Class

Definition: AI Agent managing legal document summarization tasks, using LegalDocumentAnalyzer and JudgmentSummarizer.

Attributes	**Type, Name**
	String, agentName
	LegalDocumentAnalyzer, docAnalyzer
	JudgmentSummarizer, judgmentSummarizer
Methods	**Return Type, Method Name(Parameters)**
	String, summarizeDocument(String documentText, String documentType)

LegalDocumentAnalyzer Class

Definition: Component for general legal document analysis, including clause extraction and entity recognition (using a fine-tuned model).

Attributes	**Type, Name**
	AIModel, analysisModel (e.g., fine-tuned DistilBERT for Legal Analysis)
Methods	**Return Type, Method Name(Parameters)**
	String, analyzeLegalDocument(String documentText)
	List<String>, extractClauses(String documentText)
	List<String>, recognizeLegalEntities(String documentText)

JudgmentSummarizer Class

Definition: Specialized component for summarizing court judgments, potentially using a model specifically fine-tuned for judgment summarization.

Attributes	Type, Name
	AIModel, summaryModel (e.g., fine-tuned MobileBERT for Legal Summarization)
Methods	**Return Type, Method Name(Parameters)**
	String, summarizeJudgment(String judgmentText)

3.3.4 ai.research Package - Legal Research Components

LegalResearchAgent Class

Definition: AI Agent that automates legal research, interacting with QueryFormulator, DatabaseSearcher, and ResearchResultSummarizer.

Attributes	Type, Name
	String, agentName
	QueryFormulator, queryFormulator
	DatabaseSearcher, dbSearcher
	ResearchResultSummarizer, resultSummarizer
	SourceVerifier, sourceVerifier
Methods	**Return Type, Method Name(Parameters)**
	List<String>, performLegalResearch(String researchQuery)
	String, refineResearchQuery(String currentQuery, String feedback)
	String, summarizeResearchResults(List<String> results)
	Boolean, verifySourceCredibility(String source)

QueryFormulator Class

Definition: Component that formulates effective search queries for legal databases based on user input (potentially using a smaller model for query understanding and refinement).

Attributes	**Type, Name**
	AIModel, queryModel (e.g., smaller model for query understanding)
Methods	**Return Type, Method Name(Parameters)**
	String, formulateQuery(String userInput)
	String, refineQuery(String currentQuery, String feedback)

LegalDatabaseSearcher Class

Definition: Component to directly interact with legal databases for searching, executing queries and retrieving results.

Attributes	**Type, Name**
	LegalDatabaseConnection, dbConnection (e.g., for Yargıtay DB)
Methods	**Return Type, Method Name(Parameters)**
	List<LegalDocument>, searchDatabase(String query)
	boolean, connectToDatabase()
	boolean, disconnectFromDatabase()

ResearchResultSummarizer Class

Definition: Component to summarize research results, making them more digestible for the user.

Attributes	**Type, Name**
	AIModel, summaryModel (e.g., fine-tuned MobileBERT for Legal Summarization)
Methods	**Return Type, Method Name(Parameters)**
	String, summarizeResults(List<LegalDocument> researchResults)
	String, formatSummaryForUser(String summaryText)

SourceVerifierService Class

Definition: Provides source verification as a service component, checking the credibility and authority of legal sources.

Attributes	**Type, Name**
	List<String>, trustedLegalDomains (e.g., list of official legal websites)
Methods	Return Type, Method Name(Parameters)
	Boolean, verifyCredibility(LegalDocument source)
	String, getAuthorityScore(LegalDocument source)

4. Glossary

Token: A digital security credential used for authentication and authorization, often in the form of a JWT (JSON Web Token).

API (Application Programming Interface): An interface that allows communication between two software components.

Authentication: The process of verifying a user's identity.

Authorization: The process of determining whether a user has permission to access specific resources.

Backend: The server-side part of an application that handles data processing and business logic.

Frontend: The client-side part of an application responsible for the user interface.

Database: A system used to store and manage structured data (e.g., PostgreSQL, MongoDB, MySQL).

Encryption: The process of converting data into a secure format to prevent unauthorized access.

Middleware: Software that acts as a bridge between different systems, handling requests and responses.

OAuth: A widely used standard for authentication and authorization.

AI Agent: In the context of software systems and Artificial Intelligence, an AI Agent is a more autonomous and proactive software entity than a simple component. An agent is typically designed to make decision, take actions and being proactive.

Fine-tuning: A process in machine learning where a pre-trained model is further trained on a smaller, task-specific dataset to improve its performance on a particular task or domain (in JustiWise's case, the legal domain).

NER (Named Entity Recognition): A Natural Language Processing (NLP) task that involves identifying and classifying named entities in text, such as people, organizations, locations, dates, and in the legal context, legal entities like case names, statutes, etc.

NLP (Natural Language Processing): A field of computer science and linguistics concerned with enabling computers to understand, interpret, and generate human language.

RAG (Retrieval-Augmented Generation): An AI architecture that combines the strengths of pre-trained language models with information retrieval. It enhances the model's ability to generate accurate and contextually relevant responses by retrieving information from an external knowledge source (like a database) and using it to inform the generation process.

Transformer Model: A type of neural network architecture that has revolutionized Natural Language Processing. Models like BERT, DistilBERT, MobileBERT, and the Llama family are based on the Transformer architecture, known for their ability to process sequential data like text effectively.

5. References

[1] PERSONAL DATA PROTECTION INSTITUTION | KVKK | Personal Data Protection Authority. (2024). Kvkk.gov.tr. https://kvkk.gov.tr/

[2] The Code affirms an obligation of computing professionals to use their skills for the benefit of society. (n.d.). Www.acm.org. https://www.acm.org/code-of-ethics/

[3] IEEE - IEEE Code of Ethics. (n.d.). https://www.ieee.org/about/corporate/governance/p7-8.html

[4] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0

[5] Spring Framework Documentation, https://spring.io/projects/spring-framework

[6] React Documentation, https://react.dev/

[7] Typescript Documentation,https://www.typescriptlang.org/docs/