TED UNIVERSITY

Faculty of Engineering

Computer Engineering Department

CMPE_492 Senior Design Project II

FINAL 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

TABLE OF CONTENTS

1- Introduction

JustiWise is an AI-powered legal platform that uses smart automation, enhanced accessibility, and focused educational support to modernize and optimize the whole legal workflow. JustiWise can analyze statutes, case law, and client-provided documents quickly and accurately by integrating a Retrieval-Augmented Generation (RAG) pipeline with a fine-tuned large language model. An AI-driven question-answering interface that adjusts its answers based on the user's role, whether they are a client, lawyer, or law student automated legal document summary, and interactive virtual trial simulations for experiential learning are some of the key features. JustiWise provides a scalable, compliant, and user-centric experience that is suited to the changing demands of contemporary legal practice. It is built on a secure microservice architecture (FastAPI backend, PostgreSQL + pgvector for vector search, and React/MUI frontend) and is strengthened by stringent verification and privacy controls.

2- Architecture and Design

The final architecture and design of JustiWise consist of a secure, modular, and scalable microservice-based AI platform tailored to the needs of the Turkish legal ecosystem. The system is composed of three core subsystems:

1. Client Layer (Frontend)

The frontend is implemented using React.js, TypeScript, and deployed on Vercel. It provides a responsive and accessible user interface for three types of users: clients, lawyers, and law students. Key features include:

- AI-powered legal chat interface
- Virtual courtroom simulation
- Document upload, preview, and download
- Role-based navigation and protected routes
- Integration with backend and AI services via RESTful APIs

2. Server Layer (Backend)

The backend utilizes Supabase, an open-source backend-as-a-service built on PostgreSQL, to handle authentication, real-time database operations, and storage services. Key functionalities include:

- **User Authentication**: Managed via Supabase's built-in email/password auth and JWT-based session tokens.

- **Role-Based Access Control (RBAC)**: Each user (client, lawyer, student) is assigned a role at signup, stored securely in Supabase tables.

- **Case Management**: Case records, uploaded documents, and metadata (title, parties, timestamps) are managed through Supabase tables and API.

- **File Storage**: Legal documents (PDFs) are uploaded to Supabase Storage buckets; links to these files are stored in the database

- **API Communication**: The frontend communicates with Supabase services directly via Supabase client libraries.

- **Security & Compliance**: Supabase's row-level security (RLS) policies ensure data privacy and secure multi-tenant architecture, aligning with KVKK and GDPR requirements.

3. AI Layer (Intelligent Services)

The AI services run as independent FastAPI microservices, written in Python, and powered by a fine-tuned a model tailored for Turkish legal language. Core features include:

- Retrieval-Augmented Generation (RAG) using pgvector and Supabase/PostgreSQL for vector similarity search.

- Legal document summarization, translation, and legal Q&A tailored per user role.

- Real-time consultation and courtroom simulation.

- Citation verifier that cross-checks legal outputs with public APIs (e.g., UYAP, Resmî Gazete).

- Confidence-based response filtering and explainability outputs ("Why did JustiWise say this?").

Communication Between Subsystems

The frontend interacts directly with Supabase for most backend needs and separately with the FastAPI AI service for intelligent processing. All communication is secured through HTTPS and token-based access.

3- Impact of Engineering Solutions

The project addresses several key challenges:

Economic: Reduces legal processing costs through automation by minimizing the need for manual case handling, document analysis, and preliminary legal consulting. Through AI-driven summarization and Q&A features, JustiWise automates processes that typically require significant time and workforce, leading to lower operational costs for law firms and greater efficiency for independent lawyers.

Societal: Promotes legal literacy and training for students through interactive courtroom simulations and AI-guided case studies. Law students can participate in realistic, role-based scenarios that enhance their understanding of courtroom dynamics, improve argumentation skills, and prepare them for professional environments in an engaging, low-risk setting.

Environmental: Encourages paperless legal operations via digital document processing, reducing reliance on physical archives, printing, and manual filing. By storing, analyzing, and sharing legal documents electronically, JustiWise supports sustainable practices and reduces the carbon footprint associated with traditional legal documentation workflows.

4- Contemporary Issues

4.1 Algorithmic Bias & Fairness

Definition & Context

When an AI system's results consistently disadvantage people or case categories covered by the Turkish Constitution's Article 10 on equality before the law, this is known as algorithmic bias. The dangers of indirect prejudice in automated legal analytics are highlighted in recent scholarly research and TBB seminars.

Applicability to JustiWise

Our refined models rely significantly on criminal rulings from the Supreme Court (Yargıtay); a lack of family-law or administrative-law content could distort argument recommendations and perpetuate historical prejudices.

Impact & Risk

A biased recommendation that sways legal advice may be unconstitutional, subjecting practitioners to malpractice lawsuits or bar association disciplinary action.

Roadmap and Controls

Using stratified sampling, the administrative, civil, and criminal sub-domains were balanced.

False-negative audits and group-specific accuracy (scikit-fairness).

Bias-drift dashboard; results are attached to the Model Card.

2026 goal : Before every model update, counterfactual fairness tests are conducted.

4.2 Transparency & Explainability

Meaning and Background

According to Turkish procedural lawand the "gerekçeli karar" theory, court rulings and consequently the instruments that influence them must be comprehensible and subject to scrutiny.

Applicability to JustiWise

Attorneys must defend AI-assisted arguments to courts or opposing counsel while interacting with a chat front-end.

Impact & Risk

Opaque reasoning can damage client trust or make AI-generated submissions inadmissible.

Roadmap and Controls

Verification of citations is required; each antecedent ID is compared to the UYAP/Resmî Gazete. "JustiWise said this, but why?" For 30 days, the button saves the explanation JSON.

4.3 Reliability & Hallucination Risk

Definition

In AI systems, hallucination refers to outputs that are factually incorrect or completely made up. In the legal field, this can mean generating fake laws, incorrect article numbers, or non-existent court decisions—posing serious risks for both legal professionals and clients.

Applicability to JustiWise:

JustiWise uses Turkish laws and court rulings to produce legal recommendations and summaries. The model may mislead the user and undermine system trust if it "hallucinates" a fictitious court decision or misquotes a statute.

How hallucinations happen

Hallucinations frequently happen when:

- The system attempts to "guess" as it cannot locate pertinent information in its database.
- The format of a reference is misinterpreted or misinterpreted.
- The model produces responses that are excessively inventive or arbitrary.

Our technological methods

We use a number of security measures to lessen this risk:

- Legal Document Retrieval (RAG): JustiWise looks through official Turkish legal databases (such as UYAP and Mevzuat Bilgi Sistemi) before producing a response. Only the data the model discovers there may be used.

- Citation Verifier: After creating a legal reference, a verification tool uses public APIs to determine whether the law or ruling is real. The reference is eliminated or marked if the check is unsuccessful.

- Low-Confidence Warning: The user is presented with a warning message recommending manual inspection if the model appears uncertain (based on internal confidence scores).

- Content Filters: When the model is unable to locate any legitimate documents, it is told not to create new citations.

4.4 Privacy & Confidentiality (KVKK / Attorney–Client Privilege)

Meaning and Background

Legitimate basis, data minimization, and security are required under Law No. 6698 on the Protection of Personal Data (KVKK) (Articles 4–12).

Relevance to JustiWise: Users submit sensitive personal information, identity documents, and pleadings for retrieval and summarization.

Impact & Risk

Any data breach could result in KVKK penalties and invalidate privilege.

Roadmap and Controls

The incident response plan is in line with the KVKK breach notification regulations, and there is an annual penetration test. Aim for ISO/IEC 27701 certification by 2026.

5- New Tools and Technologies

In the JustiWise project, we adopted modern AI techniques and tools to ensure our legal assistant system is reliable, explainable, and well-aligned with the needs of Turkish legal professionals. One of the most important innovations we used is Retrieval-Augmented Generation (RAG).

5.1 Generation with Retrieval Augmentation (RAG)

RAG is a hybrid system made up of two potent parts:

- Retrieval: To locate pertinent papers, legislation, or court rulings, the system first looks through external legal sources (such UYAP and Mevzuat Bilgi Sistemi) rather than relying solely on the model's internal memory.
- Generation: The AI model then creates precise and contextually aware responses using the papers that were retrieved.

This method enhances trust, lowers the chance of hallucinations, and keeps our system current without requiring a complete model retraining.

All legal documents are pre-processed and incorporated into numerical vectors in JustiWise's vector database (PostgreSQL + pgvector), which handles retrieval. The same method is used to integrate user queries during inference, and the system uses similarity scores to obtain the most pertinent legal passages.

5.2 Model Frameworks and Fine-Tuning

We used basic model, which was further refined using Turkish legal documents, including legislation, anonymised case summaries, and Yargıtay rulings. The following tools were used for the deployment and training:

- HuggingFace Transformers: for pipeline fine-tuning and pre-trained model management
- FAISS/pgvector: for a search for similarity

The AI model will be wrapped as a RESTful microservice using FastAPI.

- For containerized deployment, use Docker.
- React is used to create user interfaces.

5.3 Layer of Verification

We included a Post-generation Verifier that verifies each created statute or decision against official government APIs in order to avoid phony citations or inaccurate legal references. Outputs that cannot be instantly validated are automatically blocked or flagged by this program.

6- Utilization of Library and Internet Resources

To guarantee effective implementation, dependable system behavior, and conformity with contemporary software standards, a variety of open-source libraries, online documentation, and developer tools were used during the JustiWise platform's development. Both the frontend testing procedures and the backend AI pipeline relied heavily on these resources.

6.1 Libraries and Frameworks Used:

Several key libraries and tools were used to accelerate development and ensure system reliability. These include:

- **LangChain** – Used for implementing the Retrieval-Augmented Generation (RAG) architecture, allowing the system to retrieve relevant Turkish legal documents before generating responses.

- **pgvector + PostgreSQL** – Enabled fast and efficient vector similarity search for document embeddings.

- **HuggingFace Transformers** – Used to load and fine-tune a language model on Turkish legal corpora.

- **FastAPI** – Provided a lightweight and performant backend framework for model inference and API communication.

- **React + Material UI (MUI)** – Used to develop the frontend interface with modern, accessible components.

- **Cypress** – Used for end-to-end UI testing to ensure functionality and responsiveness across different components.

- **Docker** – Used to containerize application components for easier deployment and testing across environments.

These tools were essential for building a modern, modular, and testable AI-based legal assistant.

6.2 Internet-Based Resources

Several high-quality web-based resources were consulted to guide the implementation of advanced components such as the Retrieval-Augmented Generation (RAG) pipeline, real-time vector search, and end-to-end testing:

Throughout the development process, official documents and internet tutorials were frequently examined. Among the examples are:

1.LangChain RAG Tutorial:

The official LangChain documentation played a central role in defining the core structure of our AI pipeline. This guide helped us:

- Understand how to combine embedding-based retrieval with a generative LLM,
- Decide on using a vector database (pgvector) with a similarity search mechanism,
- Implement context injection strategies for improved factual grounding.

2.LangGraph for LangChain Agents:

The LangGraph framework was investigated as a possible addition for upcoming versions, even though it hasn't been incorporated into our deployed system yet. For the purpose of recreating court-like dialogues or sequential legal reasoning flows, it presented a potent abstraction for creating stateful multi-step agents. The team gained a better understanding of agent memory, event handling, and control flow inside the LangChain ecosystem by evaluating LangGraph, which may have opened up new avenues for development in the following phase

3.Supabase Documentation:

Important information about handling vector embeddings, storage, and authentication in a scalable architecture was provided by the Supabase manual. Although we finally used pgvector to explicitly develop our vector search layer on a PostgreSQL instance, Supabase's demonstrations made best practices clear in:

- Security of APIs and role-based access management
- Managing the storage of metadata and massive file uploads,
- GraphQL and RESTful interface design with automated documentation.

4.Cypress Official Documentation:

In order to provide a reliable and repeatable testing environment for the JustiWise frontend, the Cypress testing guidelines were crucial. We used the official guidelines to solve numerous common Cypress issues, such as working with hidden items or asynchronous UI states, because our interface included dynamic chat views, file uploads, and modals with conditional visibility.

The guides were helpful to us:

- Create appropriate test selectors to prevent faulty user interface tests.
- Prior performing assertions, make sure the backend is ready by using cy.intercept() and cy.wait().
- Use ARIA-role validation and keyboard accessibility tests.

Because of Cypress's best practice principles, our testing structure remained stable and maintainable even when certain tests failed because to selector brittleness or backend limitations.

The development team made sure they were in accordance with the most recent best practices for frontend automation, web security, and AI pipeline architecture by consulting these resources.

Only content from peer-reviewed tutorials or officially maintained platforms was included, and these sources were used exclusively for instructional and implementation assistance purposes.

7- Testing Methodology

Using the Cypress test automation framework, we established a structured testing methodology based on component-driven end-to-end (E2E) testing to guarantee the JustiWise legal assistant platform's quality, functionality, and user experience.

Test Strategy

Our test approach was black-box, meaning that tests were executed without access to internal code logic. Instead, we interacted with the system exactly as a user would: through the user interface, input forms, buttons, and data entry flows.

Each test case was written using Cypress commands such as cy.get(), cy.type(), cy.click(), and cy.should(). In more advanced scenarios, we included file uploads (cy.selectFile()), visibility checks, and API interaction validations.

The test plan covered the following five major components of the system:

1. JustiWise Chat Interface
2. Courtroom Page UI
3. Document Summary Chat
4. Login Page
5. User Profile Management

Each component was tested for UI behavior, functionality, accessibility, responsiveness, and error handling under both normal and edge-case conditions.

Tools and Setup

- **Framework**: Cypress v12+
- **Environment**: Localhost with mock database instances
- **Test Style**: Behavior-driven with cy.get(), cy.type(), cy.should() patterns
- **Automation Type**: Black-box, no internal state manipulation
- **Focus Areas**:

a. Input/output integrity
b. DOM element visibility
c. Interaction reliability
d. Component loading

    e. Error messages
    f. Backend integration (in selected tests)

8- Test Results and Assessment

Overall Test Summary

The Cypress test run shows a significant number of failures across various components of the JustiWise application. While some core functionalities like login and basic file uploads are passing consistently, many critical UI interactions, display, and backend integration tests are failing.

- **Total Test Suites/Components:** 5
- **Total Tests Executed:** Approximately 35-40 (exact count varies by how sub-tests are counted)
- **Overall Pass Rate:** Low (Many red "X" icons indicating failures)
- **Overall Failure Rate:** High

Detailed Breakdown by Component

1. JustiWise Chat

- **Tests:** 4
- **Passed:** 1
- **Failed:** 3 (75% failure rate)

Key Issues:

- **cy.type() on multiple elements:** The test "Should allow user to type and send a message" failed because cy.type() was called on a selector that matched two elements instead of a single target input. This indicates a non-unique or overly broad selector.
- **Interaction with Hidden Elements:** Tests related to file attachment ("Should allow file to be attached and previewed", "Should send message with attached file...") failed because cy.selectFile() was attempted on a hidden input element (<input hidden="">). Cypress typically requires interaction with visible elements that trigger the file input (e.g., a button to open the file dialog).
- **Recommendation:** Refine selectors to be more specific. For file uploads, simulate the user clicking the visible element that opens the file input, then cy.selectFile() can be used on the hidden input.

2. CourtroomPage Bileşeni Testleri (Courtroom Page Component Tests)

- **Tests:** 8
- **Passed:** 1

- **Failed:** 7 (87.5% failure rate)

Key Issues:

- **Element Visibility (opacity: 0):** The initial UI display test failed because an input element had opacity: '0', making it invisible.
- **Element Not Found (Brittle Selectors):** Most failures here ("Expected to find element: div[style*='max-height: 300px'] p, but never found it.") indicate that the target elements either weren't present on the page, or the selectors used were too fragile (e.g., relying on inline styles [style*='...']).

3. Document Summary Chat Tests

- **Tests:** Approximately 17 (Total)
- **Passed:** Mixed, with notable failures in "Initial Page Load", "Chat Functionality", "User Interface and Styling", "Responsive Design", "Accessibility", and "Performance and Edge Cases".

Key Issues:

- **Initial Page Load:**
  - **Cypress Syntax Error:** TypeError: cy.get(...).should(...) _is not a function. This is a fundamental syntax error in the Cypress test code itself.
  - **Element Not Found:** The upload icon button was not found, possibly due to timing or an incorrect selector for an MUI component.
- **Chat Functionality:**
  - **Element Not Found (Brittle Selectors):** Many failures ("should send user message and receive AI response", "should handle multiple messages", "should handle long messages", "should scroll message history") are due to elements like MuiBox-root[style*='background-color: rgb(25, 118, 210)'] or MuiPaper root[style*='max-height'] not being found. These are highly susceptible to minor style changes.
- **User Interface and Styling:**
  - **Element Not Found (Brittle Selectors):** Similar to chat, tests rely on style-based selectors that lead to "element not found".
  - **Attribute Assertion Failure:** A textarea was expected to have the rows attribute but didn't, indicating a potential UI bug or a mismatch in how rows is handled by the component.
- **Responsive Design:**
  - **Element Not Found (Brittle Selectors):** One failure was due to a brittle selector for MuiBox-root[max-width='ind'].
- **Accessibility:**
  - **Cypress Syntax Errors:** Multiple accessibility tests (should have proper ARIA labels and roles, should be keyboard navigable) failed due to _is not a function or tab is not a function, indicating incorrect Cypress command usage

    (e.g., cy.tab() is not a standard Cypress command, cy.realPress('Tab') from cypress-real-events is usually used).

- **Performance and Edge Cases:**
    - **Interaction with Disabled/Covered Elements:** "should handle rapid button clicks gracefully" failed because the button was disabled or covered by another element, preventing interaction. This points to potential UI state issues or race conditions.

4. Login Page Test

- **Tests:** 3
- **Passed:** 3 (100% pass rate)

Key Issues:

- **None.** This suite is entirely passing, which is excellent. It suggests that basic login form rendering, successful login with valid credentials, and displaying toast messages for invalid credentials are all working as expected.
- **Recommendation:** Keep up the good work here.

5. Profile Component Tests

- **Tests:** Approximately 19 (Total)
- **Passed:** Mixed, with significant failures in "Profile Display", "Profile Edit Display", "Save Functionality", "User Interface Interactions", "Accessibility", "Error Handling", and "Performance".

Key Issues:

- **Profile Display:**
    - **CSS Assertion Failure:** Avatar width mismatch (120px expected, 28px actual). This is a visual/styling bug.
    - **Element Not Found (Generic Selector):** MuiTypography-root for join date was not found, possibly due to a very generic selector.
- **Profile Edit Display:**
    - **Interaction with Hidden Elements:** "should open edit dialog..." failed due to an input being not visible (opacity: 0).
    - **Element Not Found:** "should display current user data in edit form" failed because the input was not found.
- **Save Functionality:**
    - **CRITICAL Backend/Application Error:** "Error: The following error originated from your application code, not from Cypress. It was caused by an unhandled promise rejection -> new row for relation "users" violates check constraint users_role_check". This is a major functional bug in the application's backend or data model, not a Cypress issue. Cypress is correctly catching an unhandled promise rejection.

    - **Interaction with Disabled/Covered Elements:** Tests like "should save role changes successfully" and "should save both username and role changes" failed because elements were disabled or "cannot be interacted with".
    - **Cypress Syntax Error:** cy.get(...).should(...) _is not a function.
- **Image Upload:**
    - **All Passed:** Good. Similar to Document Summary, the upload mechanism itself seems robust.
- **User Interface Interactions:**
    - **Element Not Found / Class Missing:** "should handle form field focus state" failed because MuiOutlinedInput-root was expected to have Mui-focused class but didn't.
    - **Interaction with Disabled/Covered Elements:** "should maintain form state when switching between fields" failed due to disabled elements.
- **Accessibility:**
    - **Cypress Syntax Errors:** Multiple failures (should be keyboard navigable, should have proper form labels, should support screen readers) due to _is not a function or tab is not a function.
- **Error Handling:**
    - **CRITICAL Backend/Application Error:** "should handle network errors gracefully" again caught the "users_role_check" database constraint error.
    - **Test Data Setup Issue:** "should validate file upload size and type" failed because the specified test file ("C:\Users\Ahmet Ergin\Desktop\cmpe492\justiwise\This is not an image") was not found. This is an issue with the test environment or fixture setup.
- **Performance:**
    - **Interaction with Disabled/Covered Elements:** "should handle multiple rapid clicks gracefully" failed due to disabled or covered elements.

Common Problems Identified

1. **Fragile/Brittle Selectors:** The most prevalent issue is the use of selectors based on volatile attributes like inline styles ([style*='...']) or generic component roots (MuiBox-root, MuiTypography-root). These break easily with minor UI/styling changes.
2. **Element Visibility/Interaction Issues:** Many tests fail because Cypress cannot interact with elements that are hidden, have opacity: 0, are disabled, or are covered by another element. This suggests:
    - Timing/loading issues where elements aren't ready.
    - Incorrect UI state (e.g., buttons should be enabled but are disabled).
    - Z-index or layout problems where elements obscure each other.

3. **Cypress Test Code Syntax Errors:** Several tests fail due to _is not a function or tab is not a function. This indicates fundamental errors in how Cypress commands or assertions are being used.
4. **Critical Backend/Application Errors:** Unhandled promise rejections indicating database constraint violations (users_role_check) are a severe functional issue that needs immediate attention from the development team, as it's not a Cypress test bug but an application bug being exposed by the tests.
5. **Missing Test Data:** Some file upload validation tests fail because the specified test file cannot be found in the provided path.

Assessment and Suggestions

- **Positive:** The file upload and login processes are dependable and flawless.
- **Drawback:** Under test settings, important user processes like chat conversations, courtroom user interface, and profile management are presently erratic or non-functional.

Suggestions:

- Data-tested attributes should be used in place of brittle selectors.
- To guarantee that items are displayed prior to interaction, enhance loading state handling.
- Correct syntax mistakes in the Cypress test and use the appropriate accessibility testing instructions.
- Unhandled database exceptions must be addressed by the backend team.
- Verify that every test fixture file is accessible in the appropriate locations.

9- Conclusion

The goal of the JustiWise project was to create a platform for legal assistants driven by AI that would be suited to the requirements of attorneys, clients, and law students. The system provided legal document analysis, AI-driven Q&A, and interactive courtroom simulations by combining a Retrieval-Augmented Generation (RAG) pipeline, an optimized model, and a secure microservice-based architecture.

Technically, the project achieved its main objectives: a role-specific frontend interface, a testing deployment architecture, and a working backend with citation verification and RAG retrieval. Cypress testing showed a number of UI problems, primarily with brittle selections and visibility limitations, but the main features, such file upload and login, worked as intended.

Here is the test result :

Metric	Value
Total Test Suites	5
Total Tests Executed	~40
Overall Pass Rate	Low
Most Stable Component	Login Page (100% pass)
Most Problematic Components	Profile Page, Courtroom Page, Chat Interfaces

10- References

[1] https://www.acm.org/code-of-ethics/

[2] https://www.langchain.com/langgraph

[3] https://python.langchain.com/docs/tutorials/rag/

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