# PROJECT REPORT

## (TestBankasi – Online Exam & Assessment System)

### TITLE PAGE

Student Name: **ABDOU VALERIO FOMA KENKACK**
Class : 2 year
Project Title: TestBankasi – Online Exam & Assessment System
Course Name : Database Systems / Web Development
Academic Year : 2025 – 2026

---

### ABSTRACT

This project presents the design and implementation of **TestBankasi**, a secure, full-stack online examination system. Unlike basic quiz apps, this system implements a hierarchical education structure (Institutions → Levels → Lessons → Topics). The system enables students to generate dynamic exams based on difficulty quotas, tracks performance in real-time, and enforces strict security via database triggers.

The system is developed using **React (Vite)** for the frontend and **ASP.NET Core Web API** for the backend, utilizing the **MVC (Model-View-Controller)** pattern. Data integrity is enforced strictly at the database level using **SQL Server**, **Stored Procedures**, and **Triggers**, ensuring that business logic remains consistent regardless of the application layer.

---

### TABLE OF CONTENTS

---

## 1. INTRODUCTION

Digital assessment systems often lack the depth to handle complex curriculum structures. **TestBankasi** was built to bridge this gap. It allows for granular testing—a student can take a specific exam on "Algebra" (Topic) within "Mathematics" (Lesson) for "10th Grade" (Level). The system uses a **Micro-ORM (Dapper)** for high-performance data retrieval and separates the User Interface (React) from the Data Logic (SQL Server) completely.

## 2. PROBLEM DEFINITION

Many existing quiz systems suffer from:

1. **Logical Inconsistencies:** Allowing users to mix unrelated topics (e.g., History questions in a Physics exam).
2. **Cheating Risks:** Students submitting answers after the time limit due to frontend-only validation.
3. **Performance Issues:** Slow generation of exams when the question bank grows to thousands of records.
4. **Rigid Structure:** Inability to handle different difficulty quotas (e.g., "Give me 2 Hard, 3 Medium, 5 Easy questions").

## 3. OBJECTIVES OF THE PROJECT

- To implement a **Restful API** using **ASP.NET Core** that serves JSON data to a React client.
- To use **Dapper** for efficient SQL execution instead of heavy ORMs like Entity Framework.
- To enforce **Referential Integrity** using Composite Keys and Unique Constraints.
- To prevent "Time-Hacking" using a database **Trigger (`trg_SinavZamanKilidi`)**.
- To visualize student performance using SQL Views (`View_DetayliPerformans`).

## 4. SCOPE OF THE PROJECT

- **Role-Based Access:** Admin (Teachers) vs. Standard Users (Students).
- **Dynamic Exam Generation:** Algorithmically selecting questions based on requested Topics and difficulty distribution. Ensuring a fair selection of questions among the different topics and different difficulty level
- **Exam Review:** Storing user choices vs. correct answers for historical review.

- **Soft Deletion:** Implementing `SilinmeTarihi` to preserve data history without cluttering active queries.

---

## 5. SYSTEM REQUIREMENTS

### 5.1 Hardware Requirements

- **Server:** Capable of running SQL Server 2019+ and .NET 8 Runtime.
- **Client:** Any modern web browser (Chrome, Edge, Firefox).

### 5.2 Software & Technology Stack

- **Frontend:** React.js (Vite Build Tool), CSS Modules, Axios (HTTP Client).
- **Backend:** ASP.NET Core 8.0 Web API.
- **Architecture Pattern:** MVC (Model-View-Controller) / Repository Pattern.
- **Database:** Microsoft SQL Server (T-SQL).
- **Data Access Layer:** Dapper (Micro-ORM).
- **API Testing:** Swagger / Postman.

---

## 6. SYSTEM ARCHITECTURE

The system follows a **Decoupled N-Tier Architecture**:

1. **Presentation Layer (React):**
    - Single Page Application (SPA).
    - Manages state using `useState` and `useEffect`.
    - Communicates via Async/Await HTTP requests.
2. **API Layer (ASP.NET Core Controllers):**
    - Acts as the traffic controller.
    - Endpoints: `AuthController`, `ExamController`, `QuestionController`.
    - Returns HTTP Status Codes (200 OK, 400 Bad Request, 401 Unauthorized).
3. **Data Access Layer (Repository):**
    - **Repository Pattern:** `AuthRepository.cs`, `ExamRepository.cs`.
    - Directly executes SQL Stored Procedures using `IDbConnection` (Dapper).
4. **Database Layer (SQL Server):**
    - Contains the "Brain" of the logic (Stored Procedures, Triggers, Views).

---

## 7. DATABASE DESIGN

The database is highly normalized (3NF) to prevent redundancy.

### 7.1 Key Tables

- **`Kullanici` (User):** Stores hashed passwords and links to `EgitimSeviye`.
- **`Soru` (Question):** The central table for question text.
- **`SoruSecenek` (Options):** Linked via `SoruID`.
- **`TestOturum` (Session):** Represents a single exam attempt (Start Time, End Time, Score).
- **`KullaniciTestSoru` (Junction Table):** Stores which question was asked in which session and what the user answered.

### 7.2 Critical Constraints (The "Difficult Concepts")

- **Data Integrity:** `UQ_Konu_Ders_Integrity` prevents orphaned logic (e.g., ensuring a Topic actually belongs to the selected Lesson).
- **Performance:** `IDX_Soru_Konu_Zorluk` (Composite Index) allows the `sp_BaslatSinav` procedure to find "Hard Math Questions" in milliseconds.

### 7.3 Advanced SQL Objects

- **Trigger (`trg_SinavZamanKilidi`):** A backend security lock. If a student tries to `INSERT` an answer after `BitirZaman` (End Time), the database rolls back the transaction and throws Error 51000.
- **Procedure (`sp_BaslatSinav`):** Accepts parameters for `EasyCount`, `MediumCount`, `HardCount` and builds a random exam dynamically. Thereby ensuring fair selection of questions across topics and difficulty levels.

---

## 8. IMPLEMENTATION DETAILS

### 8.1 Backend (Dapper & MVC)

We chose **Dapper** over Entity Framework for speed.

- **Code Example:** In `ExamRepository.cs`, we map SQL results directly to C# Objects:

  C#

  ```
  var parameters = new { KullaniciID = userId, DersID =
  lessonId ... };
  ```

```
var result = await connection.QueryAsync<ExamQuestionDTO>("sp_BaslatSinav", ...);
```

- **Authentication:** implemented using **JWT (JSON Web Tokens)**. The `AuthController` issues a token upon login, which the React frontend attaches to every subsequent request header (`Authorization: Bearer <token>`).

### 8.2 Frontend (React)

- **Routing:** `react-router-dom` handles navigation between `Login`, `Dashboard`, and `ExamRoom`.
- **State Management:**
  - `token`: Stored in `localStorage` to persist login.
  - `examState`: Tracks the current question index (0 to 10) during the quiz.

### 8.3 Security

- **Password Hashing:** Passwords are never stored in plain text. (Note: Handled in the API/Database layer).
- **SQL Injection Protection:** All inputs are parameterized via Dapper; no raw string concatenation is used.

---

## 9. KEY TECHNICAL CHALLENGES & SOLUTIONS

During development, I encountered specific challenges that required engineering solutions:

1. **Challenge:** Preventing students from modifying answers after the exam timer hits 00:00.
   - **Solution:** Frontend disabling is not enough. We added an `AFTER UPDATE` trigger (`trg_SinavZamanKilidi`) in SQL. Even if someone uses Postman to send an API request, the database rejects it.
2. **Challenge:** Generating an exam with a specific mix of difficulty levels (e.g., 50% Hard).
   - **Solution:** `sp_BaslatSinav` uses three separate `SELECT TOP (x) ... ORDER BY NEWID()` statements and combines them using `UNION ALL` to guarantee the exact distribution requested.
3. **Challenge:** Calculating analytics without slowing down the dashboard.
   - **Solution:** We created `View_DetayliPerformans` which pre-aggregates joins between 5 tables. The API simply selects from this view instead of running complex logic on every page load.

---

## 10. TESTING AND RESULTS

### Test Cases

| **Test ID** | **Description** | **Expected Outcome** | **Result** |
|---|---|---|---|
| **TC-01** | User logs in with wrong password | API returns 401 Unauthorized | **Pass** |
| **TC-02** | Student requests 10 questions | System returns 10 random unique questions | **Pass** |
| **TC-03** | **Time Lock:** Answer submitted after timeout | Database throws "Time Expired" error | **Pass** |
| **TC-04** | User views history | Shows correct calculation of Score | **Pass** |
| **TC-05** | Teacher's class statistics | Shows correct statistics and updates each time a student takes an exam | **Pass** |
| **TC-06** | Question Modification | Changes are correctly managed by the database | **Pass** |

---

## 11. LIMITATIONS

1. **The "Insufficient Questions" Edge Case:** If the database contains fewer questions than requested (e.g., asking for 5 Hard questions when only 2 exist), the system returns a smaller exam without warning.

---

## 12. FUTURE ENHANCEMENTS

1. **Improve the question selection algorithm:** If a user asks for questions in a particular difficulty level and there are not enough in the DB. The program will warn the user and select questions under different difficulty level.

2. **AI Question Generation:** Integration with OpenAI API to help the teachers create new questions.
3. **Database Enrichment:** Add more institutions universities, Colleges to allow for inter-university or inter-college competitions without needing the participants to be in the same place

---

## 13. CONCLUSION

The **TestBankasi** project demonstrates a robust implementation of modern Full-Stack development. By moving critical logic (constraints, time checks) to the Database layer and keeping the API layer lightweight using Dapper, we achieved a secure and fast application. It solves the core problem of "Rigid Assessment" by offering flexible, quota-based exam generation.

## 14. REFERENCES

• Microsoft. (2024). *ASP.NET Core Web API Documentation* with swagger/openAI
https://learn.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-8.0

• C# DAPPER
https://www.geeksforgeeks.org/c-sharp/c-dapper/

• React. (2024). *React.js Official Documentation*.
https://react.dev/

• Axios. (2024). *Axios HTTP Client Documentation*.
https://axios-http.com/

**Appendix**
Appendix A: ER Model Diagram
Appendix B: SQL Master Script
Appendix C: Backend API Source Code
Appendix D: Frontend Application Source Code