POWER GRID CORPORATION OF INDIA

INTERNSHIP REPORT

Ujjwal Godara | 9654671068 | ujjwal.godara9@gmail.com

DEC 2023 - JAN 2024

Guide : Ms. Shambhavi Pandey

Summary

During my internship at Power Grid Corporation of India, I undertook the development of a circular management system, leveraging the capabilities of ASP.NET Core and SQL Server. This report provides a comprehensive overview of how these technologies were harnessed to create an efficient and robust system.. Also implemented redis caching in this to make the application a lot faster and user friendly.

Technology Used

- ASP.NET Core
- SQL Server
- Redis

ASP.NET Core MVC Framework

The foundation of the circular management system was laid using the ASP.NET Core MVC framework. This modern web application framework follows the Model-View-Controller architectural pattern, providing a modular and organized structure for development. The MVC pattern allowed me to separate concerns and maintain a clean codebase.

Database Modeling with SQL Server

To persistently store and manage data, I utilized SQL Server, a powerful relational database management system. The database was designed using the database-first approach, where the models were generated based on the existing database schema. This approach ensured that the application's data model remained in sync with the database structure, fostering consistency and reducing development time.

Model Entities:

1. HrPolicy Model:
- Represented the core entity for the circular management system.
- Included properties such as Id, Guid, TopicId, PolicyRefNo, PolicyName, SortOrder, Remarks, Status, and timestamps for insertion, update, and archiving.
- Established relationships with other entities, such as PolicyDocument and PolicyTopic, creating a well-connected data model.

2. MasterRole Model:
- Defined the roles within the system.
- Included properties like RoleId, RoleName, Status, and a collection of Users associated with each role.

3. PolicyDocument Model:
- Represented individual documents related to HR policies.
- Incorporated properties such as Id, Guid, TopicId, PolicyId, DocumentCaption, DocumentDate, KeywordsString, DocumentTypeId, Remarks, SortOrder, Extension, ContentType, FileSize, FileName, FileHash, Status, and timestamps.
- Established relationships with other entities, including HrPolicy and PolicyTopic.

4. PolicyTopic Model:
- Represented different topics or categories of HR policies.
- Included properties like Id, Guid, Name, SortOrder, Status, Remarks, and timestamps.
- Formed relationships with HrPolicy and PolicyDocument entities.

5. User Model:
- Represented users in the system.
- Included properties like Id, EmpId, RoleId, Status, UpdatedBy, UpdatedOn, and Email.
- Formed a relationship with the MasterRole entity.

Views and Controller:

The ASP.NET Core MVC framework facilitated the creation of various views and controllers to enable seamless interaction between users and the system. Views were designed using HTML, CSS, and JavaScript to provide an intuitive and visually appealing user interface. Controllers handled user inputs, processed data, and interacted with the database through Entity Framework.

Search Functionality

Implemented a robust search functionality allowing users to search for documents based on policy topics, sub-policy topics, or specific keywords within the documents. The search results were efficiently displayed in the 'SearchDocument.cshtml' view, providing users with quick and accurate access to relevant information.

Document Management

Developed an 'Edit Book' view to centralize document management. This view allowed users to edit, delete, or view details of individual documents, providing a comprehensive solution for document-related operations.

Enhancement with Redis Caching

In addition to the solid foundation laid with ASP.NET Core and SQL Server, the circular management system was further optimized through the implementation of Redis caching. Redis caching played a pivotal role in significantly improving the application's speed and overall performance.

Redis Caching Implementation

To enhance the system's efficiency, I introduced Redis caching as a caching mechanism. Redis, known for its lightning-fast data retrieval, was employed to store and retrieve frequently accessed data. The caching strategy involved populating the cache with data from SQL Server upon application startup, minimizing the need for repeated database queries during user interactions.

Improving Application Speed

The incorporation of Redis caching led to a remarkable improvement in the application's speed. By storing essential data in Redis, subsequent user requests were served directly from the cache, reducing the latency associated with repeated database queries. This optimization significantly enhanced the user experience by delivering quicker response times and smoother interactions.

Redis and Windows-Linux Integration

One notable challenge in implementing Redis caching was the need to bridge the compatibility gap between Windows-based Visual Studio and Linux-based Redis. This was addressed by running the Redis server on a virtual machine (Linux) while hosting the Visual Studio development environment on a Windows machine. This approach ensured seamless communication between the two components, enabling the benefits of Redis caching in a Windows-centric development environment.

Enhanced User Experience

With Redis caching in place, the circular management system provided users with a more responsive and fluid experience. The reduction in data retrieval times not only improved the system's overall speed but also contributed to a more seamless navigation experience for users interacting with various views and functionalities.

Role-Based Access Control and Security

The Redis caching implementation was fine-tuned to exclude sensitive user details, such as those stored in the User model, ensuring that security measures remained intact. Role-based access control was maintained, allowing admins access to all functionalities while restricting employees to document searches. This careful consideration of security aspects alongside performance optimization demonstrated a balanced approach in system design.

Conclusion

The integration of ASP.NET Core and SQL Server proved to be a powerful combination for the development of the circular management system. The MVC architecture facilitated the creation of a well-structured and maintainable codebase, while SQL Server provided a robust and scalable database solution. The use of the database-first approach ensured alignment between the application's data model and the underlying database schema, enhancing consistency and development efficiency. Overall, the implementation of ASP.NET Core and SQL Server successfully laid the groundwork for a feature-rich and efficient circular management system.

The integration of Redis caching into the circular management system served as a pivotal enhancement, further elevating the application's performance and user experience. The strategic use of caching mechanisms, coupled with the robust foundation of ASP.NET Core and SQL Server, resulted in a well-rounded system that

not only efficiently managed HR policies but also provided users with a responsive and seamless platform.