**Approach Paper: Voting Application**

**Objective:**  
To develop a comprehensive voting application that enables users to add candidates and voters, cast votes securely, and view real-time updates on voting statistics. The application is developed using .NET 8 for the back end and Angular 17 for the front end.

**Technology Stack:**

1. **Backend:**
   * **.NET 8:** Utilized the latest .NET framework for robust backend development.
   * **Entity Framework (EF) Core 8:** Employed EF Core for data modeling and database interaction.
   * **Dependency Injection:** Implemented dependency injection for modularity and testability.
   * **Swagger:** Integrated Swagger for API documentation and testing.
   * **Automapper:** Used Automapper for seamless mapping between DTOs and entities.
   * **SonarQube:** Leveraged SonarQube for code quality analysis and security checks.
   * **Cors:** Implemented CORS for enhanced security, restricting access to designated origins.
   * **XUnit:** Utilized XUnit for unit testing to ensure code reliability.
   * **Clean Architecture:** Adopted a clean architecture approach with four distinct layers - Application, Business Logic, Data Access, and Data Entities.
   * **SQL Server:** Employed SQL Server for database management, utilizing migrations for schema changes.
2. **Frontend:**
   * **Angular 17:** Developed the frontend using Angular 17 for a responsive and dynamic user interface.
   * **Modular Structure:** Organized code into feature modules for maintainability and scalability.
   * **Component-based Architecture:** Utilized components for encapsulating functionality and enhancing reusability.

**Key Features:**

1. **User Management:**
   * Implemented functionalities for adding candidates and voters securely.
   * Ensured data integrity with validation rules such as required fields and unique voting constraints.
   * Used pop-up dialog boxes for seamless addition of candidates and voters.
   * Implemented server-side pagination for enhanced user experience.
2. **Voting Process:**
   * Facilitated a streamlined voting process with dropdowns for selecting candidates and voters.
   * Dynamically updated voting statistics without storing redundant data in the database.
3. **Security and Validation:**
   * Integrated CORS to control cross-origin requests and enhance application security.
   * Validated APIs using Swagger without frontend interaction.
   * Ensured SOLID principles adherence and implemented error handling mechanisms.

**Additional Features and Assumptions:**

* **Dynamic Vote Count:** Votes for candidates are not stored in the database; instead, they are dynamically fetched using EF from the Vote table.
* **Single Vote Restriction:** A single voter can cast only one vote, ensuring fair voting practices.
* **Filtered Voter Dropdown:** Voter dropdown for casting a vote displays only those voters for whom the hasvoted flag is false, preventing multiple votes from the same voter.
* **Data Annotations:** Added data annotations such as required fields for candidate and voter names and enforced selection of both voter and candidate dropdowns for casting a vote.
* **Pagination:** Implemented server-side pagination to maintain readability for voter and candidate grids, enhancing user experience.
* **Pop-up Dialog:** Utilized pop-up dialog boxes for adding new voters and candidates with a discard option, improving user interaction.
* **Real-time Updates:** Newly added candidates or voters are displayed in the grid and dropdown on top in real-time.
* **Tech Concepts:** Leveraged tech concepts such as CORS, Automapper, Pagination, Swagger, Sonar, Xunit, and Code First for robust application development.

**Architecture Explanation:**

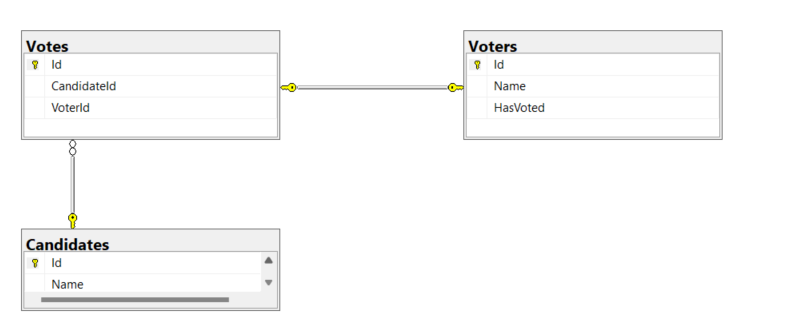
**Backend Architecture:**

1. **.NET 8 Application (VotingApp):**
   * **Entry Point (Program.cs):** Acts as the starting point of the application, initializing necessary configurations and components.
   * **Controllers:** Includes controllers for managing candidates, voters, and votes, responsible for handling incoming HTTP requests and orchestrating the flow of data.
2. **Business Layer (VotingApp.BL):**
   * **Entity Services Interfaces:** Define interfaces for all entity services, abstracting business logic from the controllers.
   * **Services:** Implements business logic for entities, encapsulating data manipulation and validation operations. These services are invoked by controllers through dependency injection.
3. **Data Layer (VotingApp.DL):**
   * **Entity Repositories Interfaces:** Define interfaces for all entity repositories, providing a contract for data access operations.
   * **Repositories:** Implements repository patterns for entities, encapsulating database interaction logic. These repositories are called by services via dependency injection.
4. **Data Entities Layer (VotingApp.DE):**
   * **Business Models:** Define DTO models for each entity, used for front-end operations as request and response objects.
   * **Data Models:** Define database entity models for each entity, representing the structure of tables in the database.
   * **Migrations:** Automatically generated scripts for database schema changes, managed by EF Core migrations.
   * **Automapper Class:** Responsible for mapping between DTOs and entities, simplifying data transfer between layers.
   * **DbContext Class:** Configured EF Core DbContext, facilitating interaction with the database using ORM capabilities.
5. **XUnit Test Layer (VotingApp.Tests):**
   * Consists of unit test cases for controllers, services, and repositories, ensuring code reliability and functionality across all layers.

**Frontend Architecture:**

* **Angular 17 Application:**
  + **Node Modules:** Utilized Node Modules for managing dependencies.
  + **Components:** Organizes features into reusable components for a modular architecture.
  + **Services:** Consumes backend APIs for data exchange, ensuring separation of concerns.
  + **Environment Configuration:** Manages environment variables for flexibility and configurability.

**Database diagram: (VotingApp)**



**Steps to Run the Application:**

**Backend:**

1. Open the solution in Visual Studio.
2. Update the connection string to connect to your local database server.
3. Run migrations (Update-database with connection string and default project set to ‘VotingApp.DE’) to create tables and columns in your local database.
4. Start the application. (F5)
5. Access the swagger via e.g.: localhost:5276/swagger.

**Frontend:**

1. Open the project in Visual Studio Code.
2. Open the Terminal.
   * Execute the command: npm run start.
3. Access the application via e.g.:localhost:4200.

**Coding Guidelines and Validation:**

* Followed standard coding guidelines including regions, commenting, optimized logic, naming conventions, unused namespaces, SOLID principles, and try-catch for error handling, etc.
* Utilized SonarQube to prevent code smells and address security issues.
* Implemented Swagger for API validation without frontend interaction.
* Validated front-end features using the application interface.

**Conclusion:** The developed voting application meets all client requirements by providing a secure, efficient, and user-friendly platform for conducting voting programme. The integration of advanced technologies, adherence to coding standards, and emphasis on security and validation ensure a robust and reliable solution.