1.2 Project Requirements: Extensive Overview

The project involves building a robust and user-friendly Todo website with specific front-end development requirements. Below are the detailed requirements and features that need to be implemented:

Front-end Development

User Authentication

1. **Login and Registration Screens:**
    - **Login Screen:** Develop a login screen where users can enter their email and password to access their accounts. The screen should include form fields for email and password, a login button, and links to the registration page and password recovery (optional).
    - **Registration Screen:** Create a registration screen that collects the user's name, surname, email, and password. The form should include validation to ensure data integrity:
        - **Name and Surname:** Text fields for the user's full name.
        - **Username:** Text field for the username. (unique).
        - **Email:** An email field with validation to ensure the input is a valid email format.
        - **Password:** A password field that requires a minimum of 8 characters and displays validation feedback.
2. **Persistent Login Sessions:**
    - Implement functionality to ensure that once a user logs in, their session remains active until they explicitly log out. This can be achieved using cookies or local storage to store authentication tokens.
3. **Password Requirements:**
    - Enforce password rules during registration, requiring a minimum of 8 characters. Validate the email field to ensure the user enters a properly formatted email address.

User Session Management

1. **Logout Functionality:**
    - Provide a way for users to log out of their accounts. This typically involves a logout button in the navigation bar or a user profile menu. Logging out should clear the user's session and redirect them to the login screen.

Todo Management

1. **Navigation Tabs:**
    - **Completed and Incomplete Tabs:** Create two distinct tabs in the navigation bar labeled "Completed" and "Incomplete". These tabs should filter the todos based on their completion status.
2. **Server-side Sorting:**
    - Sort the todos by their creation date in descending order (newest to oldest) on the server-side. This ensures that todos are consistently ordered regardless of the user's device or browser.
3. **Search Functionality:**
    - Implement a search bar at the top of both the "Completed" and "Incomplete" pages. The search should filter todos based on their content, allowing users to quickly find specific tasks.
4. **Todo Status Updates:**
    - Ensure that when an incomplete todo is marked as completed, it is moved from the Incomplete list to the Completed list. This status change should be reflected in real-time.
5. **Forms for CRUD Operations:**
    - **Add Todo:** Develop a form for adding new todos, including fields for the todo's title, description, and due date.
    - **Update Todo:** Create an editable form for updating existing todos, allowing users to change the title, description, and completion status.

- **Delete Todo:** Provide a way to delete todos, either through a button in the todo item or a bulk delete option.
- **Form Validations:** Use react-hook-form to implement form validations, ensuring all fields are properly filled out and providing feedback for invalid inputs.

Design and Responsiveness

1. **Responsive Design:**
    - Use Bootstrap to ensure the website is responsive and looks good on various devices and screen sizes. This includes using Bootstrap's grid system and responsive utility classes.
2. **Atomic Design Pattern:**
    - Organize the front-end codebase following the Atomic Design Pattern. This involves breaking down the UI into small, reusable components (Atoms), combining them into larger components (Molecules and Organisms), and assembling pages (Templates) from these components.

Caching

1. **Redis Cache:**
    - Implement caching for todos using Redis. This involves storing the todos in a Redis cache to improve performance and reduce the number of database queries.
    - Ensure the cache is updated whenever a CRUD operation is performed on the todos. This includes adding new todos, updating existing ones, and deleting todos.

Browser Tab Customization

1. **Tab Image and Title:**
    - Set a custom favicon (tab image) and title for the browser tab to enhance the user experience and branding. The title should reflect the current page or section of the website the user is viewing.

Backend Development and Firebase Integration

Database Management

1. **Using Firebase Firestore for Data Operations:**
    - **Firestore Overview:** Firebase Firestore is a scalable and flexible NoSQL cloud database provided by Google Firebase. It is designed to handle real-time data synchronization and offline support, making it an ideal choice for web applications that require responsive and dynamic data handling.
    - **Data Structure:** Design the Firestore database structure to efficiently manage todo items and user data. This involves creating collections and documents that logically organize data for quick retrieval and updates.
2. **Creating and Integrating a Firebase Project:**
    - **Firebase Project Setup:**
        - Create a new Firebase project in the Firebase Console.
        - Enable Firestore and Authentication services within the project.
    - **Integration with Next.js:**
        - Install the necessary Firebase SDKs (firebase and @firebase/firestore) in the Next.js project.
        - Initialize Firebase in the Next.js application using a configuration object that includes the Firebase API key, project ID, and other necessary details. This configuration should be included in a Firebase initialization file (e.g., firebase.js).
3. **Storing Sensitive Data:**
    - **.env File:**
        - Create a .env file in the project root to securely store sensitive data such as the Firebase API key, project ID, and other configuration details.
        - Use environment variables to access these sensitive details in the application code without exposing them in the source code.
        - Example of a .env file: In Source Code.
4. **Implementing a REST-API with Next.js for CRUD Operations:**
    - **API Routes in Next.js:**
        - Use Next.js API routes to create RESTful endpoints for performing CRUD operations on todos.
        - Set up routes for creating, reading, updating, and deleting todos. These routes will interact with the Firestore database to perform the necessary operations.
    - **Example API Routes:** In Source Code.
5. **User Authentication Table in Firestore:**
    - **User Collection:**
        - Create a collection named users in Firestore to store user authentication details.
        - Each document in the users collection will represent a single user and contain fields such as name, surname, email, passwordHash, and createdAt.
        - The createdAt field will store the account creation date.
    - **Data Structure:**

        - Example of a user document: In Source Code.

6. **Email/Password Authentication:**
    - **Firebase Authentication Setup:**
        - Enable the email/password sign-in method in the Firebase Console under the Authentication section.
    - **Sign-Up Functionality:**
        - Implement a sign-up API route that creates a new user in Firebase Authentication using the email and password provided by the user during registration.
        - Upon successful registration, store additional user details (name, surname, and creation date) in the Firestore users collection.
    - **Sign-In Functionality:**
        - Implement a sign-in API route that authenticates the user using the provided email and password. Upon successful authentication, generate and return an authentication token to maintain the user's session.
