This is a Java configuration class named `VoterApplicationConfig`, which provides various beans related to user authentication for the application. Let's go through the code and explain each part:

1. `@Configuration`: This annotation marks the class as a configuration class, allowing Spring to detect and apply the configurations defined within this class.

2. `@RequiredArgsConstructor`: This Lombok annotation automatically generates a constructor with required arguments for the class fields. In this case, it will create a constructor with an argument for the `repository` field.

3. `private final VoterRepository repository;`: This field holds an instance of `VoterRepository`, presumably a custom repository for managing voters' data.

4. `@Bean`: This annotation is used to define beans (i.e., Spring-managed objects) that can be automatically configured and used by the Spring application context.

5. `public UserDetailsService userDetailsService()`: This method defines a bean for `UserDetailsService`, which is used by Spring Security to load user details during authentication.

6. `return username -> repository.findByEmail(username).orElseThrow(() -> new UsernameNotFoundException("User not found"));`: The implementation of the `UserDetailsService` bean is a lambda function that takes a username (in this case, an email) as input and queries the `VoterRepository` to find a user with the provided email. If the user is found, the details are returned; otherwise, a `UsernameNotFoundException` is thrown.

7. `public AuthenticationProvider authenticationProvider()`: This method defines a bean for `AuthenticationProvider`, which is used by Spring Security for authenticating users.

8. `DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();`: This line creates an instance of `DaoAuthenticationProvider`, a Spring Security-provided implementation of `AuthenticationProvider`.

9. `authProvider.setUserDetailsService(userDetailsService());`: This sets the previously defined `UserDetailsService` bean as the user details service to be used by the `DaoAuthenticationProvider`.

10. `authProvider.setPasswordEncoder(passwordEncoder());`: This sets the `PasswordEncoder` bean (created in the next method) as the password encoder for the `DaoAuthenticationProvider`.

11. `public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception`: This method defines a bean for `AuthenticationManager`, which is used by Spring Security to handle the authentication process.

12. `return config.getAuthenticationManager();`: This line retrieves the `AuthenticationManager` from the `AuthenticationConfiguration`. The `AuthenticationManager` bean is automatically configured by Spring Security, and this method retrieves it for use in the application.

13. `public PasswordEncoder passwordEncoder()`: This method defines a bean for `PasswordEncoder`, which is responsible for encoding and decoding passwords.

14. `return new BCryptPasswordEncoder();`: This line creates an instance of `BCryptPasswordEncoder`, a strong password encoder provided by Spring Security.

In summary, the `VoterApplicationConfig` class provides essential configuration beans for user authentication and security in the application. It sets up a custom `UserDetailsService` that loads user details from the `VoterRepository`. It also creates an `AuthenticationProvider` using `DaoAuthenticationProvider` and sets the `UserDetailsService` and `PasswordEncoder`. Finally, it configures the `AuthenticationManager`, which is essential for handling the authentication process.

Overall, this class plays a significant role in configuring user authentication in the application and ensuring secure password storage with bcrypt encryption.