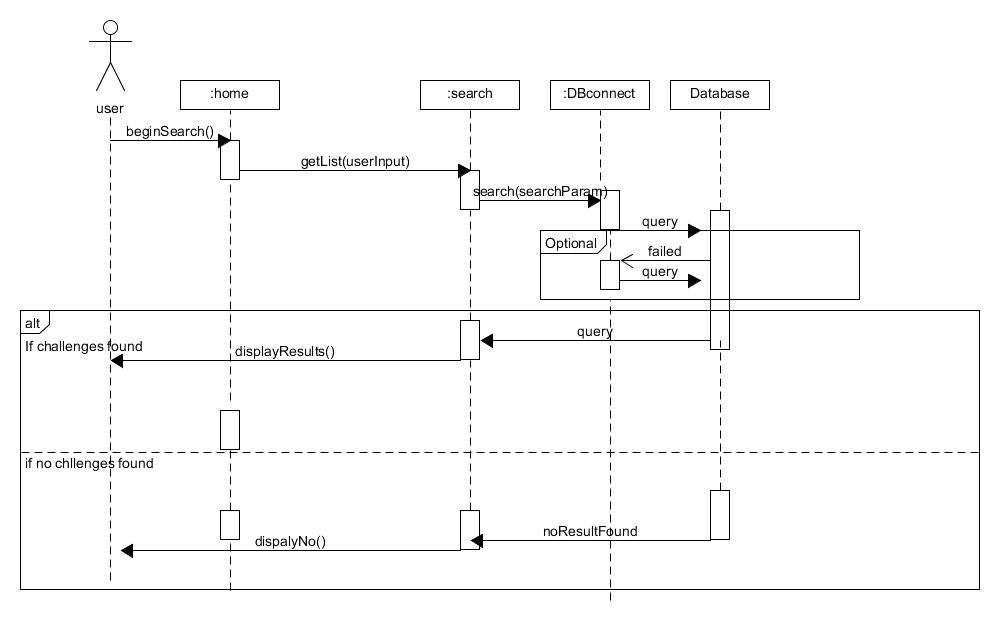
Interaction Diagrams

This begins the first set of our final included features of the front end. In the end the four major interactions a user will be doing through our interfaces. The four cases are search, view, register, and login. We found the dynamic generation of web pages not to be overly complex so there was no need for separate builder and handler classes. Since we implemented some javascript/html to aid user navigation there was no need for an interaction class to handle interaction. One page simply passes the torch and lets the next page take over. This is in line with traditional web design.

**Use Case 1: Search**

The first and primary use of the application is searching the site. This will be used by both “guest” users and registered users. This begins at the home interface which then sends the users requests through all the necessary subsystems. This is similar to Use Case two but has slightly more interaction since it can return a blank result.

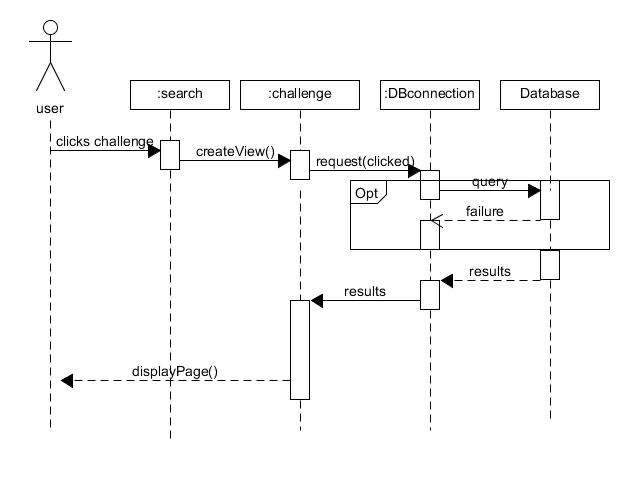


**High cohesion principle**: The user requests a search which is passed on to the search page which represents the high cohesion principle. The home only acts as an interface to handle the possible interactions between the user and sends requests to other objects to fulfil the request.

**Expert Doer principle:** By having the DBconnection be the only person that interacts with the database it allows for the other subsystems to not car about how to access the actual database. DBconnection is the only part that accesses in both the front and back ends.

**Use Case 2: View**

The second most important thing is for users to be able to access the details of the challenges. It is important to note that in these use cases after the appropriate interface on the webpage is done, it passes the request to the next class and it generates the web page. This is the basis of the design of the website. Since we know that a challenge is stored with details, we know there will not be a case of a blank query, unless there is database malfunction. This is the primary difference between use case one and 2.



**High Cohesion Principle:** When a user clicks on the challenge the interface does request to build the challenge view. By separating the creation and the displaying the application becomes more cohesive.

**Expert Doer Principle:** The results handler handles results from the database no matter what it is since it already knows how to handle challenge input there is no reason to creating another subsystem to handle data from the database.

**Use case 3 + 4: register/login**

The third use case is when a user to register on the site. When a user registers we must check that valid information is entered. Afterwards on a successful register we return user to the home page. Login is nearly identical to register, but it uses a different class because different checks need to be made to verify a already registered user. It seems redundant to list both interaction diagrams, so I focused on the register use case.

