What are the biggest priorities when designing anything used by many users at once, repeatedly? Efficiency, security and user experience. I designed a simple solution that keeps all of these variables in mind while fulfilling the project requirements.

To maximize **efficiency**, my project would utilize the python ‘threading’ library to implement multithreading, querying all 3 of our tables at once and returning the results as quickly as possible. Ideally, I would also store the data in a DBMS known for swift query-return speeds, such as PostgreSQL or Cassandra. In addition, not having autocomplete results pop up until after the user has entered 3 characters will vastly decrease the number of rows returned, speeding up our application by a significant margin.

Additionally, all inputs will be thoroughly sanitized to ensure **security**. This means taking the user’s provided data and scanning it for potential injection attacks against our SQL database. I would design templates for queries that simply take in the scrubbed user input and return the rows. In addition, the utilization of TLS certificates would enhance the security of browser-to-server/server-to-browser communications. Lastly, keeping all of this data on a local network could be something to consider, with the only drawback being that employees (coaches, scouts, etc.) wouldn’t be able to access the application while on travel.

Another important variable is **user experience**. Users don’t want to type out something, hit enter, and see search results. The autocomplete function is by nature, a UX feature. Implementing a “fuzzy search” algorithm that allows for user input error would create a more seamless experience for the user by not forcing them to go back and retype (while also eliminating the need for an autocorrect function, which could get tricky with proper nouns).

To implement this autocomplete function I would utilize technologies such as Python's threading library for multithreading, ensuring that queries to the database are executed in parallel for optimal response times. As for the database management system, I'd implement a simple PostgreSQL server, or potentially Cassandra. I would also probably choose Django ORM to handle database interactions, ensuring against SQL injection by sanitizing user inputs. Lastly, I would implement a TLS certificate to guarantee our data transmissions are thoroughly protected.