Design Decisions

1. Separate Client and Server

We chose to separate the client and server into separate docker containers for two main reasons: to provide the most robust handling of state, and to allow other applications to interact with the api. As an example of the more robust nature of this architecture, the logic is all on the server rather than being divided between the client and the server, this reduces risk and possible duplication. This separation allows us to have a “dumb” client that only makes requests and allows the server to do all of the heavy lifting. As for allowing other applications to interact with our server, basically we have a controller layer that implements a basic REST API. Because of this, any application can make requests, not just the client that we have provided.

2. Technologies

The client is written using the `React.js` framework. The server is written using `.Net Core`.

3. Server Architecture

3.1 Controllers

The controller layer is exposed to the outside world as a REST API.

Generally speaking, this layer is only used to hand off the requests and should stay as simple as possible. The role of the controller is that of an orchestrator, to hand off data and instructions to the `Adapter` layer.

3.2 Adapters

The adapter layer contains the logic to make the application work. Any data that gets dynamically created should be located at this layer, also any state that is to be saved should be located here as well.

4. NLP vs Regex

Due to time constraints, we were unable to implement an NLP solution to searching the PubMed articles. We chose instead to search for relevant keywords using Regular Expression (Regex) pattern matching. For future GT developers, the `ClarityNLP` library seemed promising; however integration may prove to be challenging.