TEST PDF 4:

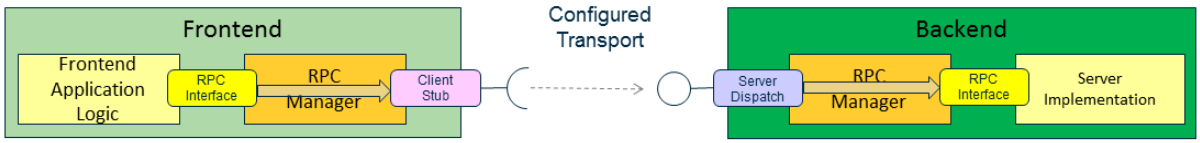
This is for testing the edge cases for the python code. To see if the number of characters is more in each styles instead of Text strings.... will the code work properly

ALPHABETS

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

PARAGRAPH

the functionality of an iTwin.js app is typically implemented in separate components that run in different processes, potentially on different machines. These components communicate through interfaces. These interfaces can either be implemented as Rpc or Ipc. For web applications, iTwin.js uses *RpcInterfaces* or RPC.



The diagram above shows an app frontend requesting operations from some backend. The terms *client* and *server* specify the two *roles* of an RpcInterface:

- *client* -- the code that runs on the frontend, and calls methods on an RpcInterface.
- *server* -- the code that runs on the backend, and implements the RpcInterface.

Classes that derive from RpcInterface define a set of operations implemented by a server, callable from a client.

As shown, client and server work with the RpcManager to manage the available RpcInterfaces. RpcManager exposes a client "stub" on the client side that forwards RPC requests. On the other end, RpcManager uses a server dispatch mechanism to relay the request to the implementation in the server. In between the two is a transport mechanism that marshalls the data passed from the client to the serverover an appropriate communications channel. The transport mechanism is encapsulated in a *configuration* that is applied at runtime.

A typical app frontend will use more than one remote component. Likewise, a server can contain and expose more than one component. For example, the app frontend might need two interfaces, Interface 1 and Interface 2. In this example, both are implemented in Backend A.