Seger David 17-054-693

UbiComp – Assignment 2 and 3

Assignment 2

Please find details on how to run the application in the README. I have implemented the following functionalities:

- Searching: The HoloLens plays music to help you concentrate while you search for something
- Reading: Say any word out loud to get its definition displays!
- Inspecting: The HoloLens automatically takes a picture of what you are inspecting, so you can use reverse image search to find the object

Please note, that the hand-in video cuts off when the inspection help is triggered, since it takes a photo and stops the recording for that.

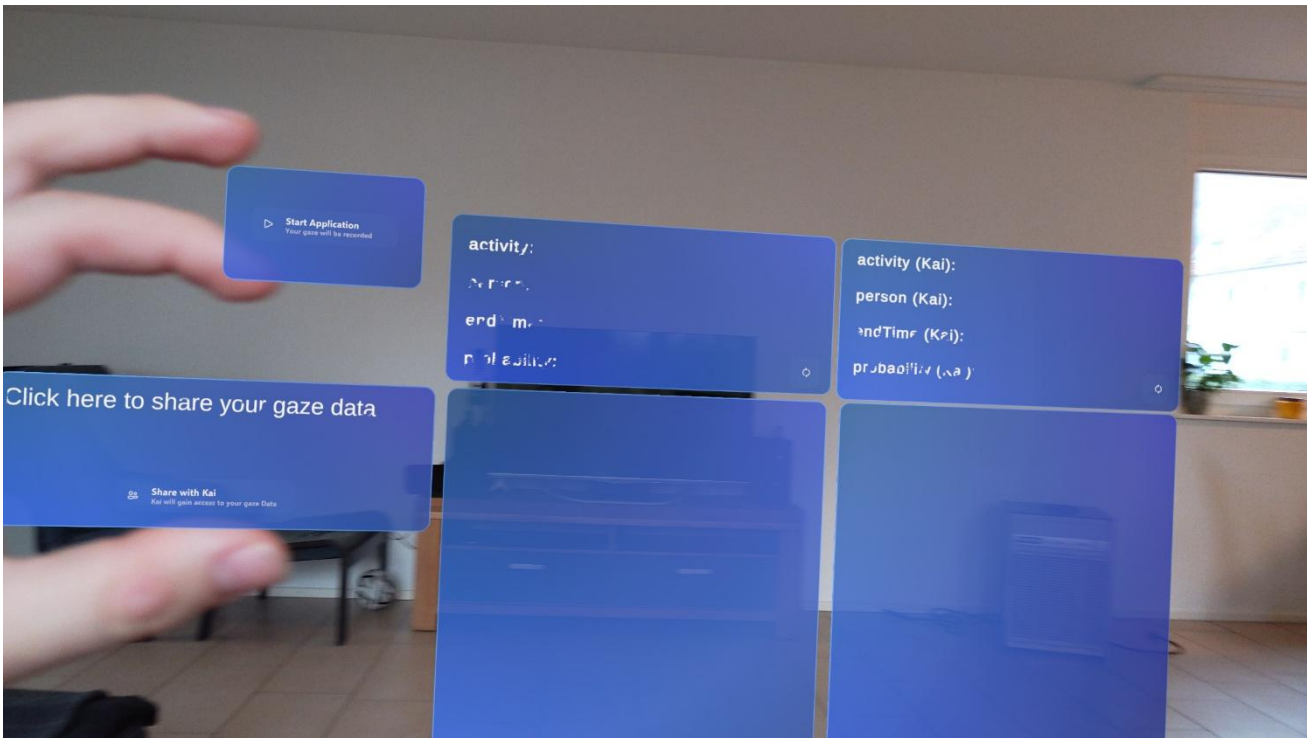
Assignment 3

How to run

After trying to implement everything in C# using the ICS unity solid pod library, I quickly hit a road block with how slow the library is, and that I had trouble accessing my data using the authenticated webclient. After much consideration I rewrote the application to access the data with REST server calls to a typescript server I have based on the program I wrote for the different tasks we had to implement in this assignment. To run the server, simply execute the typescript file found in the project “SOLID_Server”. Inside the unity project (called “Assignment_2_Reloaded”), locate the script “SolidPodRetriever.cs” and adjust line 19, set the variable “restServerUrl” to the ip and port of the server where you are running the SOLID_Server project. Do the same in the file “ShareGazeData.cs”, line 13, variable “server”. Once these changes are done, rebuild the application and deploy to the hololens to run!

The Application

The HoloLens app has 5 SOLID related fields, 1 field with a button to share data with Kai from the course, 2 fields to show info queried from the CurrentActivity.ttl file (one from my own pod, one from Kais pod), and 2 fields to showcase the supplementary material found on DBPedia to the activity (once again one from my own data, and one from Kai):



By clicking on the share button field, a method is triggered that adds a rule to the gazeData acl, which allows Kai to read from my currentActivities.ttl. The Activity info fields can be refreshed by clicking the buttons on the lower right:

The comment fields get automatically filled once the refresh button has been hit. The comment is gathered by a federated query across my/kais profile, the robot pod and dbPedia.

Besides the unity application, there is a typescript file (not the SOLID_Server) that implements different functions that were asked for in the different tasks. It is based on the typescript file that was provided as a starting point. Commented out are example how to use the script, these example correspond to the implementations of the different tasks, so in theory, by uncommenting all examples and running the script, you should complete all non-unity tasks. You can find this in the “SolidPosInteractions” folder.

Pitfalls

The biggest difficulty was definitely working with unity itself. The library I used to implement directly accessing SOLID pods was very slow and prone to errors, which made me rebuild the whole application. Deploying something to the HoloLens is tedious and takes a long time, thus I spent way more time handling the infrastructure than actually solving the tasks.

Another pitfall was the .acl rule format. The others from the course and me spent some brain power on giving access to each others Pods, with quite a few mistakes leading to bloated and weird looking final .acl files. Once you get the hang of it its easy, but it is a steep learning curve.

The same also goes for querying using SPARQL, it is an unusual querying language, especially when you are used to query with SQL. It took a few tutorials and explanation videos to wrap my head around the syntax.

The lack of UI around the SOLID server was a bit annoying, if you made a mistake, f.e. upload a file with the wrong mediatype or something like that, it was hard to correct that by deleting the file again.

Task 4b)

The „friend“ (I created a second account on the SOLID server to test these things out) has access to the “myhobbies.txt” file, as well as to the “myFriendsInfo.txt” file. They have access to the myhobbies.txt file because we added a specific .acl rule to the “test” container allowing them to access the resource. Since we added the “acl:default” setting pointing to the test container, the second account has access to all newly created files in “test”, since this rule means that, when no explicit .acl rule is given for a resource, it defaults to this rule. The second account has no access to the “myFamilyInfo.txt” file, neither read nor write, since we have not added any authorization rule for this container and this user.