CIS 726 - Assignment 4a
Spring 2013
Kevin Dean, Russ Feldhausen, Brian Dye

Architecture

Our system architecture is based on a requester/responder architecture. We have three basic participants: the website, the database server, and the authentication server.

The website is the content from the previous project, with requests to the SQL database replaced with a new Request class. The Request class handles all communication internally, and returns the data in the form it the system would have received before (as a Course, ElectiveCourse, etc).

The database server is written based on a MessageProcessing application, where it is implemented as a command line program that does an infinite loop. On each loop iteration, the server waits for a message to appear in its RequestQueue, at which point it unpackages the message and passes it into a MessageProcessor class, where it is interpreted in terms of its datatype and requests to a local database are made. The results are packaged in a Response object, which is then pushed onto a ResponseQueue. The ResponseQueue is emptied by the website, which pulls off messages with id's matching the requests it has sent. Those id's are done as GUID strings that are concatenated onto the Label attribute of the MSMQ message. Each side of the conversation remembers that GUID so that it can sort through the messages and ensure correct transfer.

The authentication server is implemented the same as the database server, but with a different MessageProcessor class that handles Users, Roles, and UserRoles instead. Conversations between it and the website are performed identically to those between the database server and the website. Our passwords are not transferred between the two in plaintext, but are hashed before sending.

When running our system, the message queues should all be created automatically. At the start of each participants loops, they run an initializer that creates the queues it will be popping from. The website creates a DBResponse and AuthResponse, the database server creates a DBRequest, and the auth server creates an AuthRequest. Currently the names of these are hardcoded into static strings on the ObjectMessageQueue.cs file, located in the Models folder of the MessageParser project in our Visual Studio solution. If the system is run in a distributed way, then these will need to be edited manually to reflect the IP addresses of the servers being contacted.

Bandwidth

In our testing, we have found that transferring a full course listing response uses approximately 4mb of data. Requests are negligible in size. Since course lists are the largest data that will be transferred between either set of publishers and subscribers, we can assume this as the upper bound of each user. We say that at peak load of 100 users our system will be transferring up to 4mb * 100 users, or about 400mb. In practice this is much higher than we expect to see, since users are not requesting full course listings constantly, but we cannot amortize over the expected pages because we do not have an accurate model of user interaction with our system. To lower this amount, we could feasibly paginate the request model, so that each request returns up to n entries from the database starting at offset m (this data would be embedded in the serialized request object). This would still not lower our actual upper bound, as all autocomplete queries require the entire course listing to perform their function.