Computer Science Advising Tool
Technical Documentation

Last Place Champions
Rico Adrian, Brian Hooper, Nick Rohde

8th of March 2018

Contents

Chapter 1

Overview

1.1 Problem

Desgin a user friendly website able to pull data from the catalog and allow the user (advisor) to easily manipulate different scenarios for the student based on multiple constraints (i.e. prerequisites, time of the class being offered in the academic year, minimum number of credits, etc.).

1.2 Requirements

1. Login Page
   (a) Accept credentials from user
   (b) Verify provided credentials
   (c) Redirect user to homepage, or display error
2. Advising Page
   (a) Retrieve existing graduation plan
   (b) Create new graduation plan
   (c) Import student transcript
   (d) Input plan constraints
   (e) Manually modify graduation plan
   (f) Print advising documents
3. Administrator Page
   (a) Modify Databases
   (b) Create/Modify/Delete users

4. Graduation Plan Generation Algorithm
    (a) Generate most efficient plan based on constraints
    (b) Finish within 2 minutes
    (c) Create a valid plan
5. Databases
    (a) Store sensitive data in an encrypted environment
    (b) Store graduation plans for all students
    (c) Store user credentials

1.3 Design

Chapter 2

Website

The website runs as a daemon, and, as such, runs permanently, even if no clients are connected. It uses nginx to forward connections from port 80 to port 5000, which is the port monitored by the website's underlying .Net MVC application.

The website has two accompanying files, namely, "CwuAdvising.service" which contains the necessary settings to run the website as a daemon; and an nginx default configuration file which establishes the forwarding between port 80 and port 5000, this file is named "default" and it must be placed in this location: /etc/nginx/sites-available/. **Note: No other service/application/protocol can use port 5000 or port 80 on the server to prevent conflicts with the website.**

Chapter 3

Algorithm

3.1 Overview

The algorithm is generating schedule courses by courses by recursion and will keep recursing until all the requirements are gone. all the requirements are going to be put in a LinkedList and all courses are from database.

3.2 Functional Description

There is a function called ListofCourses that will generate possible schedule for current quarter(just for 1 quarter) based on whether the prerequisites are met or not. It will be called until all the requirements are met.
Another function that needs to be implemented was the prereqsMet function to check whether the prerequisites of courses are met or not and whether the courses are offered in the current quarter or not.
Furthermore, the algorithm will generate the best possible course by comparing the number of quarters and by putting the courses that are prerequisites into the schedule first, for instance, the courses like CS110 and Math172 that will be important and are requirements for most courses in order to advance to the next courses(sequential courses like CS110, CS111, CS301), while general education courses or CS112 stand by themselves and mostly do not have prerequisites or can be taken in any quarters.
In the schedule.cs class, there is a nextSchedule function to increment the schedule if there are no possible courses or if maximum number of credits is reached. NextSchedule will increment quarter from Fall to Winter to Spring to Summer and will increment the year when the current quarter is Fall.

Chapter 4

Databases

4.1 MySql Database

The MySql database stores the user credentials, as well as the student graduation plans. Both are stored within the same database, but in separate tables, credentials are stored in the table "user_credentials", and graduation plans in the table "student_plans" (these names can be altered in the "DBH.service" file in the section [TABLES]).

This database is encrypted and cannot be accessed without credentials. All sensitive information is stored within this database. All interactions with the MySql Database will utilize one of two objects, namely, PlanInfo, or Credentials, these are discussed in detail below.

4.1.1 PlanInfo Type

The PlanInfo type contains the Student ID (SID), the starting quarter of a student, and an array of strings, where each index of the array contains the classes for one quarter. This type is used to access, create and modify records in the student_plans table.

4.1.2 Credentials Type

The Credentials type contains the username, password salt, and admin flag of a user. Furthermore, a password can also be given to this class, which will be ignored unless a change password Database Command is being executed. This type is used to access, create, and modify records in the user_credentials table.

4.2 NoSql Databases

The NoSql Database (DB4O) stores information about Courses, Catalogs, and Students (only directory information - e.g. Name, Starting Quarter). This database has no encryption, as it **does not** contain protected information, however, it is stored in a binary format making unauthorized access dificult.

4.2.1 Database Classes

The DB4O database stores objects of type DatabaseObject, this is an abstract type, and it is inherited by three other types, namely, Course, Student, Catalog. The DB4O database stores these objects in a serialized state, and they can be retrieved as ready-to-use C# objects, without the need for conversion or parsing.

4.3 Database Handler

The database handler is the middle man between the databases and the client(s). Its purposes are:

1. **Encapsulate database operations** - It provides query-less database access to all users. Instead, database commands are sent to the Database handler, which translates them into queries. Database commands require no knowledge as to where or how information is stored.

2. **Prevent multiple writes** - two users cannot overwrite eachother's modifications, a user can only update a database entry if it they have the current version. This is controlled via a "write-protect" property which is part of all database entries. Before a write instruction is executed, the write protect on original and new data are compared, if they do not match, the instruction is not executed to prevent overwriting changes the user may not be aware of.

3. **Manage databases** - The database handler manages the databases for the administrator. The administrator should never need to directly access the databases, except during the creation step. The handler is not capable of creating the Database, however, once the database is created it can create all required tables, and set them up.
   Note: The tables have a special format, altering this format by manually editing the database can make the table unusable.

The database handler runs in a multithreaded mode. Specifically:

1. **Master Thread** - A daemon which always runs on the server

2. **Client Thread(s)** - Each client is assigned a dedicated thread which only communicates with its assigned client, and executes all commands for this client. This allows non-critical code to be executed in parallel as much as possible, speeding up operations during periods of high-traffic.

3. **Keep Alive Thread** - A thread which periodically accesses the database to prevent the connection from timing out. This is a result of our testing, as the connection would time out over night.

Upon starting the master thread the setup will extract necessary configurations from the "DBH.ini" file to setup the running environment. After the setup has completed, it awaits new clients to connect to it through a specific TCP port (all details pertaining to the TCP/IP connection are specified in the "DBH.ini" file under the [MISC] section). The master thread runs as a daemon, and, as such, runs permanently, even if no clients are connected. Should the service database connection be lost, the service will restart to reestablish a connection to the database.

The database handler has three accompanying files, namely, "DBH Setup.pdf" which contains a detailed discussion of how to setup, start, use, and troubleshoot errors pertaining to the Database Handler; "DBH.service" which is the service file responsible for the daemon; and "DBH.ini" which is the configuration file which must be used to start the database handler.

4.3.1 Database Command

The Database Handler communicates with other applications through DatabaseCommand objects. Objects of this type contain two main components, namely, CommandType (specifying what to do, e.g. Retrieve, Delete, etc.), and the CommandOperand (an object of the DatabaseClasses type which contains information needed to execute the command, e.g. a Student object with the SID field set for a Retrieve Command). Upon reception of a command object, the Database Handler will execute it, and then send a Database Command back to the client which contains return information (e.g. A student object with all fields filled out), or an error code if the operation could not be done (e.g. the write protect on the database record did not match the one supplied in the original request).

Chapter 5

Testing

Testing was cut quite short due to the development running late, however, the components went through quite rigorous functional, and white box testing during the integration phase. We found a multitude of issues during this phase, and it unveiled many issues that were fixed before the project was submitted to the client.

5.1 Algorithm

5.1.1 StackOverflow Issues

We made sure that the algorithm won't throw StackOverflow Error by stop generating schedule for current quarter if there are still requirements left but the number of credits for this quarter haven't reached maximum credits, meaning there are no courses that can be added anymore.

5.2 Database Handler

5.2.1 Faulty Queries

During testing we detected many issues with queries, which have all been resolved. The only issue that could not be resolved regarding a query was fixed with a workaround where the query is read into a file, and then read back from the file before being executed. Without this, the MariaDB will cause a Syntax Exception, even though the syntax is correct.