Department Of Computer Science Advising Tool Technical Documentation

Last Place Champions
Rico Adrian, Brian Hooper, Nick Rohde

Last Updated: 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

Chapter 2

Website

2.1 Overview

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, "kestrel-advising.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/default`. **Note: No other service/application/protocol can use port 5000 or port 80 on the server to prevent conflicts with the website.**
For more information on the website, please see the User Guide, which contains a detailed explanation of the website. For information regarding the setup of the website, see "ServerSetup.pdf".

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 all necessary information for the website to function, including the user credentials; student graduation plans; catalogs, and degrees; courses; and student information. All these are stored within the same database, but in separate tables (these names can be altered, instructions for this can be found in "ServerSetup.pdf").

This database is encrypted and cannot be accessed without credentials. All sensitive information is stored within this database. An important thing to note is that, in order to function, the Database Handler must be given a user account with root priviledges to function properly. All interactions with the MySql Database will utilize one of several objects 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.

4.1.3 DatabaseObject Type

The DatabaseObject type is an abstract type, and it is inherited by three other types, namely, Course, Student, CatalogRequirements. It stores the common information among the three aforementioned types,

4.1.4 CatalogRequirements Type

The CatalogRequirements type stores a list of several degrees, and an ID, which represents the catalog year. This type is mainly designed as a wrapper for the degrees type.

4.1.5 DegreeRequirements Type

The DegreeRequirements type stores all course, and credit prerequisites for a given degree. This type is used by the Plan Generation Algorithm to create a graduation plan for the student's catalog year for a specific degree.

4.1.6 Course Type

The Course type stores all information regarding to courses, namely, the name, credits, times when it is offered, and prerequisites. This object can exist in two states, shallow, and complete, this is dictated by the property IsShallow. Before using the PreRequisites property, we reccommend checking the IsShallow property for **false** to ensure no problems arise as this list will be in an undefined state if IsShallow is set to true.

Shallow Course prequisites are stored in the database as a list of strings, a shallow Course object contains this list of strings, but no actual Course objects. **This is the setting that should, generally, be used.**

Complete A complete Course object contains, in addition to the list of strings, a list of Course objects as prerequisites. **This option should only be used when absolutely necessary, retrieving a complete course object will build the entire dependency tree for the course, which requires a large amount of recursion resulting in a bad time efficiency.**

4.1.7 Student Type

The Student type stores the name, ID, starting quarter, and expected graduation of a student. This type is mainly used for display purposes on the website, but the starting quarter stored in this object dictates the CatalogRequirements object that will be used to create the graduation plan for a student.

4.2 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 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.

4. **Deadlock Detector** - A thread which attempts to periodically lock the mutex lock used by other threads, if this thread does not succeed after 2 minutes, it will cause a deadlock shutdown of the software. Whehter or not the service is restarted after this depends on the settings in the "dbh.service" file. This thread exists mainly to prevent manual interactions with the service should a deadlock occur, and to execute a clean up routine gracefully as opposed to an abrupt shutdowns from a kill command. In general, no deadlocks should occur, however, it is possible for a mutex lock to become abandoned if a worker thread is killed by the operating system, in which case a deadlock would occur, this is what this

thread is trying to detect. **Note: Though it is extremely unlikely, a false positive could be detected if there are many threads using the lock for a long period of time.**

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.

4.2.1 Database Command

The Database Handler communicates with other applications through DatabaseC-ommand 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). A successfuly return will always have the CommandType *Return*, and an error will always have the CommandType *Error*.

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.

5.2.2 Storage of Data

During testing we realized some of the storage ideas we had initially were not feasible, namely, storing plans by quarter was not possible due to the large amount of conversion we would've had to do to turns a JSON string into a string array. We decided to instead store the JSON string in the database to avoid wasting time on costly conversions, which would impact the user-experience.