CS 6310 Project 4: Bringing It Together

Reflective Memo

Final team arrangement:

Daniel Rozen drozen3

James Rooke jrooke3

Shivendra Srivastava ssrivastava64

Blaine Spear bspear8

Alan Hyde ahyde7

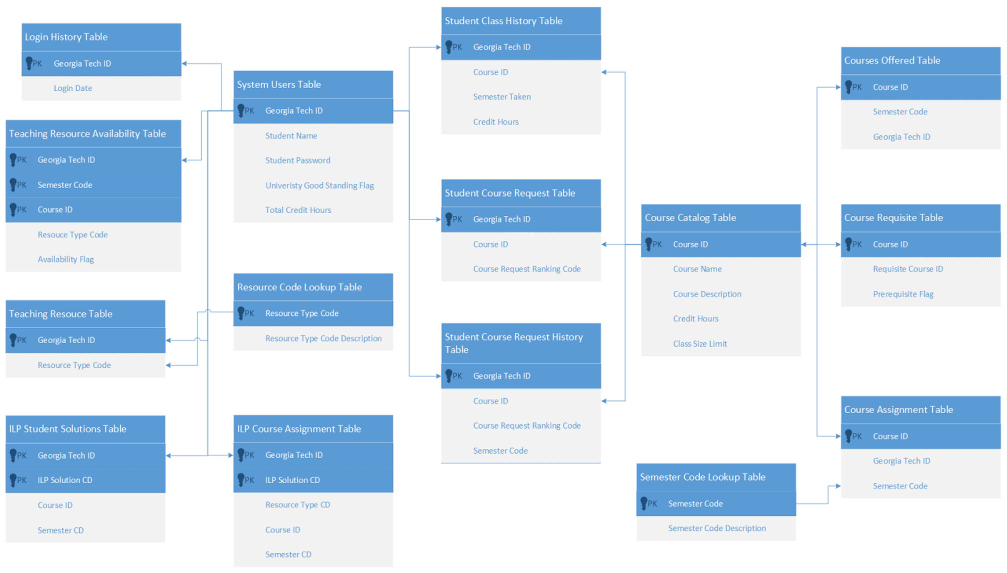
1. Summary of System Specifications

Data Model: Before Implementation

Our team's original plan was to use a technology stack where a web based interface written in HTML, CSS, and JavaScript would capture and pass inputs to Spring MVC, which would then pass those inputs into a MySQL database. The information in the database would then be used by the core Java application which would use it to build constraints and initiate the Gurobi solver. Once Gurobi calculated a solution, that solution would be added to the database. Spring MVC would then pull the solution and populate the web interface output pages.

The initial data model diagram, which was obtained from project 3, was used as a starting point for us in order to design the underlying structures in the MySQL database. The information gathered for the initial data model was taken from our past experiences with previous projects in the class. Our initial logical data model diagram looked like the following:

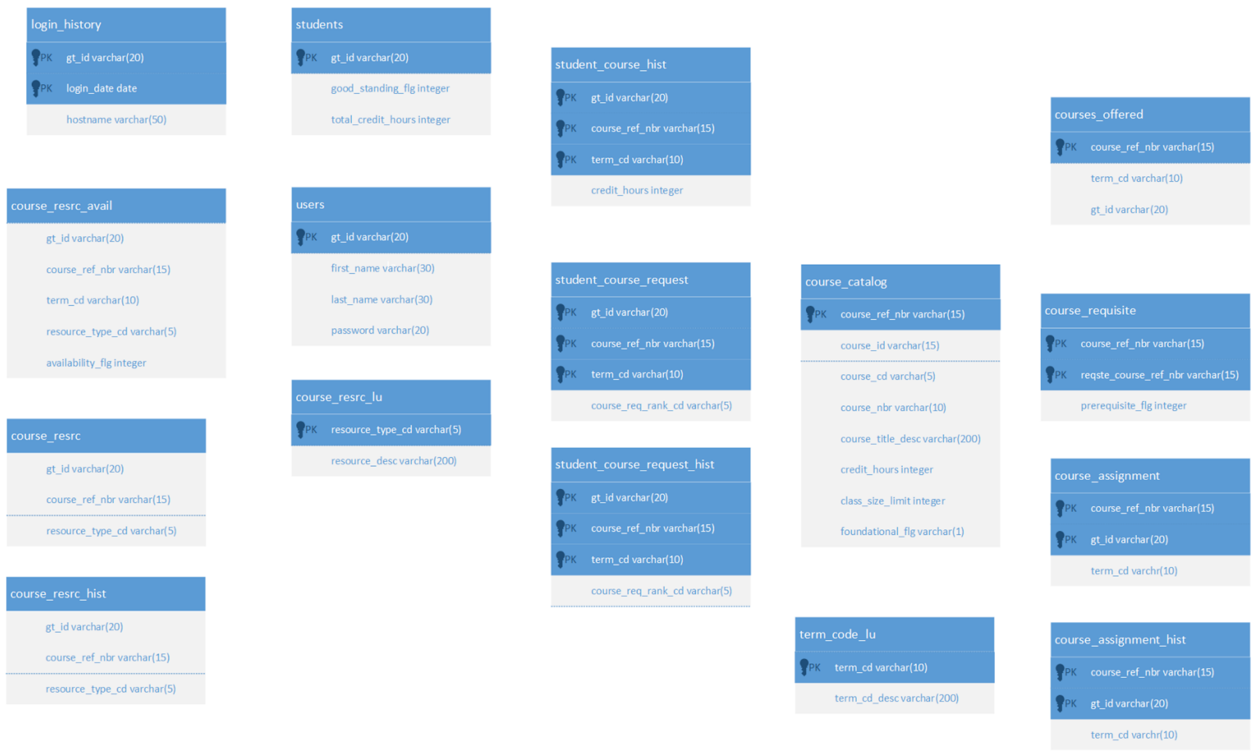
Initial Data Model Diagram



<u>Data Model: After Implementation</u>

As we worked through the project requirements and began coding our final solution, we were able to analyze more closely our needs for the data model. We discovered that we needed to make some changes to our data model design to better fit the end product we were creating. The final data model diagram was changed to the following:

Final Data Model Diagram

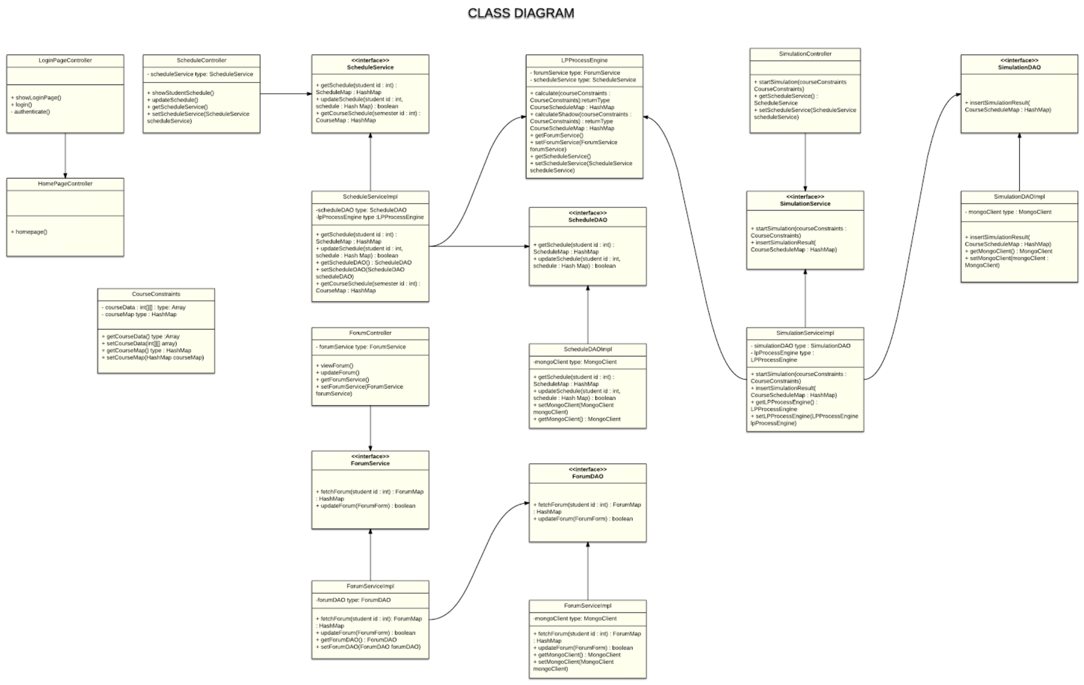


The main differences between the initial data model diagram and the final product center around modifying several tables and their attributes, adding several new tables to the design that we discovered a need for during our development process, and lastly removing several tables from the design that we did not need at all. These changes were also derived after we were given an official set of data to work with for the project.

<u>Class Diagram: Before Implementation</u>

A significant amount of code from Project 1 was reused, specifically from projects where Gurobi was the solver of choice, as Gurobi was found to be a much faster, more efficient, and user friendly LP solver. As a result, our class diagrams changed from one relevant to a GLPK specific project to one relevant to a Gurobi specific project.

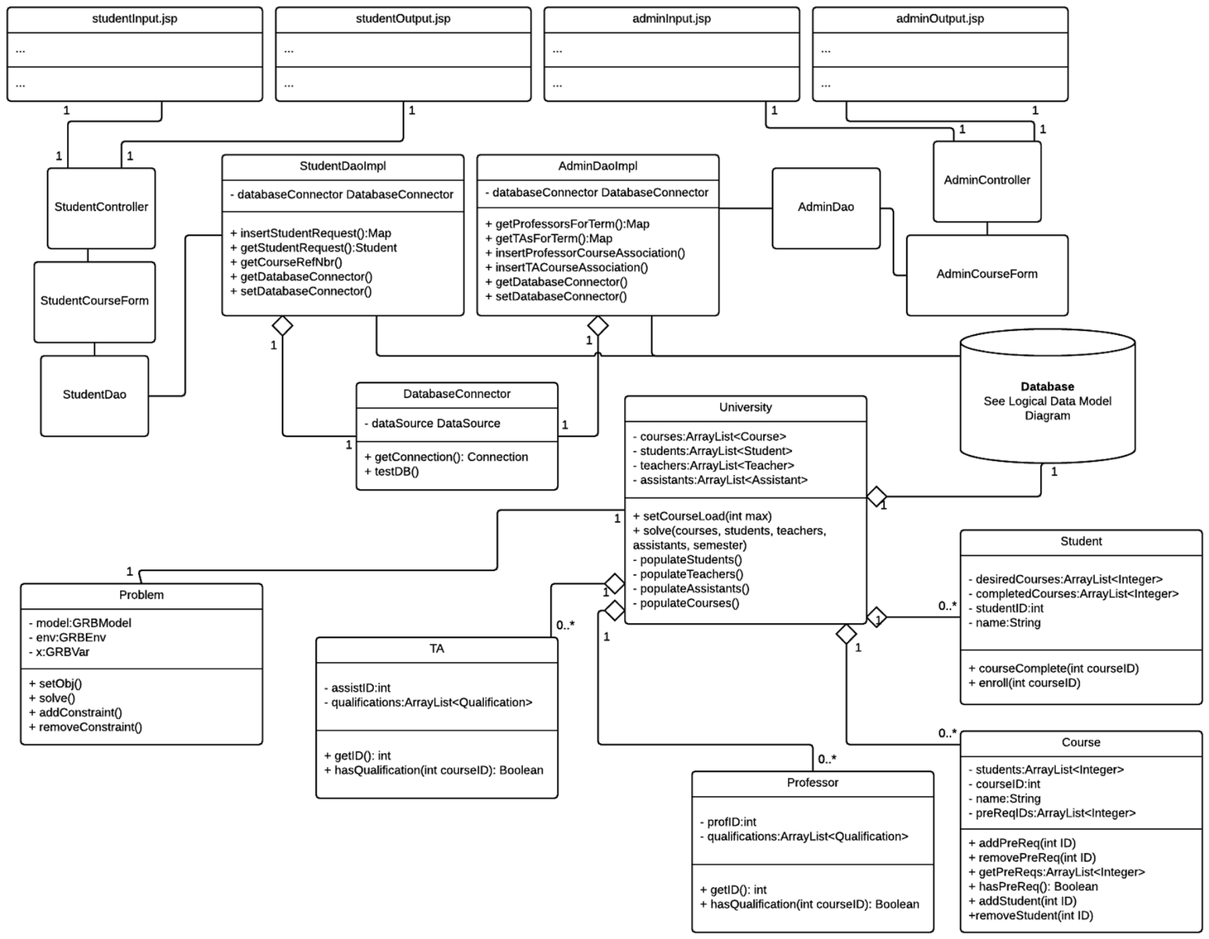
Old Class Diagram


CLASS DIAGRAM
LoginPageController
+ showLoginPage()
+ login()
- authenticate()
HomePageController
+ homepage()
ScheduleController
- scheduleService type: ScheduleService
+ showStudentSchedule()
+ updateSchedule()
+ getScheduleService()
+ setScheduleService(ScheduleService scheduleService)
CourseConstraints
- courseData : int[][] : type: Array
- courseMap type : HashMap
+ getCourseData() type :Array
+ setCourseData(int[][] array)
+ getCourseMap() type : HashMap
+ setCourseMap(HashMap courseMap)
<<interface>> ScheduleService
+ getSchedule(student id : int) : ScheduleMap : HashMap
+ updateSchedule(student id : int, schedule : Hash Map) : boolean
+ getCourseSchedule(semester id : int) : CourseMap : HashMap
ScheduleServiceImpl
-scheduleDAO type: ScheduleDAO
-lpProcessEngine type :LPProcessEngine
+ getSchedule(student id : int) : ScheduleMap : HashMap
+ updateSchedule(student id : int, schedule : Hash Map) : boolean
+ getScheduleDAO() : ScheduleDAO
+ setScheduleDAO(ScheduleDAO scheduleDAO)
+ getCourseSchedule(semester id : int) : CourseMap : HashMap
ForumController
- forumService type: ForumService
+ viewForum()
+ updateForum()
+ getForumService()
+ setForumService(ForumService forumService)
<<interface>> ForumService
+ fetchForum(student id : int) : ForumMap : HashMap
+ updateForum(ForumForm) : boolean
ForumServiceImpl
-forumDAO type: ForumDAO
+ fetchForum(student id : int): ForumMap : HashMap
+ updateForum(ForumForm) : boolean
+ getForumDAO() : ForumDAO
+ setForumDAO(ForumDAO forumDAO)
LPProcessEngine
- forumService type: ForumService
- scheduleService type: ScheduleService
+ calculate(courseConstraints : CourseConstraints):returnType CourseScheduleMap : HashMap
+ calculateShadow(courseConstraints : CourseConstraints) : returnType CourseScheduleMap : HashMap
+ getForumService()
+ setForumService(ForumService forumService)
+ getScheduleService()
+ setScheduleService(ScheduleService scheduleService)
<<interface>> ScheduleDAO
+ getSchedule(student id : int) : ScheduleMap : HashMap
+ updateSchedule(student id : int, schedule : Hash Map) : boolean
ScheduleDAOImpl
-mongoClient type: MongoClient
+ getSchedule(student id : int) : ScheduleMap : HashMap
+ updateSchedule(student id : int, schedule : Hash Map) : boolean
+ setMongoClient(MongoClient mongoClient)
+ getMongoClient() : MongoClient
<<interface>> ForumDAO
+ fetchForum(student id : int) : ForumMap : HashMap
+ updateForum(ForumForm) : boolean
ForumServiceImpl
-mongoClient type: MongoClient
+ fetchForum(student id : int) : ForumMap : HashMap
+ updateForum(ForumForm) : boolean
+ getMongoClient() : MongoClient
+ setMongoClient(MongoClient mongoClient)
SimulationController
+ startSimulation(courseConstraints CourseConstraints)
+ getScheduleService() : ScheduleService
+ setScheduleService(ScheduleService scheduleService)
<<interface>> SimulationService
+ startSimulation(courseConstraints : CourseConstraints)
+ insertSimulationResult(CourseScheduleMap : HashMap)
SimulationServiceImpl
- simulationDAO type : SimulationDAO
- lpProcessEngine type : LPProcessEngine
+ startSimulation(courseConstraints : CourseConstraints)
+ insertSimulationResult(CourseScheduleMap : HashMap)
+ getLPProcessEngine() : LPProcessEngine
+ setLPProcessEngine(LPProcessEngine lpProcessEngine)
<<interface>> SimulationDAO
+ insertSimulationResult(CourseScheduleMap : HashMap)
SimulationDAOImpl
- mongoClient type : MongoClient
+ insertSimulationResult(CourseScheduleMap : HashMap)
+ getMongoClient() : MongoClient
+ setMongoClient(mongoClient : MongoClient)

Class Diagram: After Implementation

The new class diagram added some additional and new classes designed to work efficiently with Gurobi and to pass information between the interface and the database.

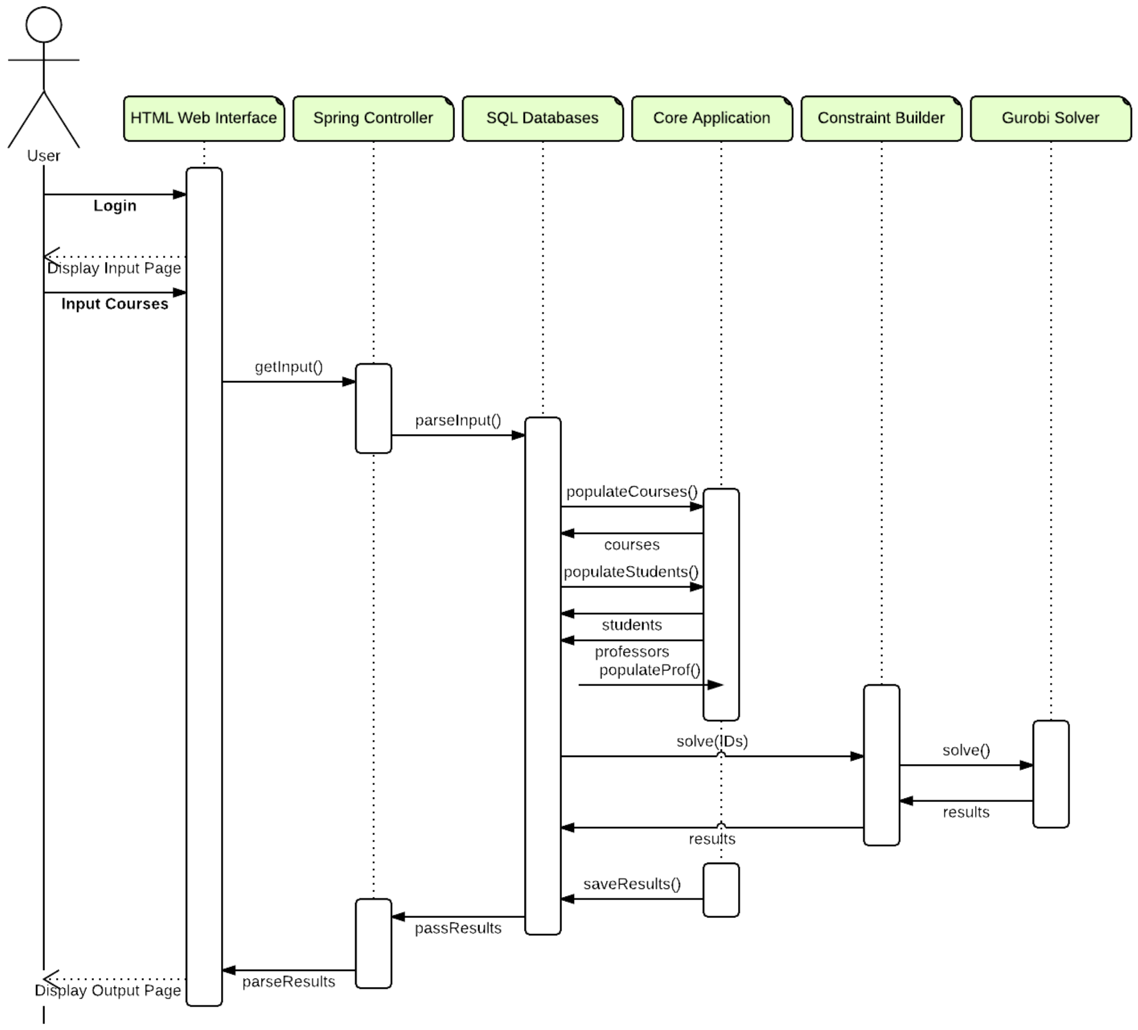
New Class Diagram



Interaction Diagrams

The user-system interactions were diagrammed using a sequence diagram. They remain relatively unchanged from Project 3 to Project 4. However, because the interaction for both types of users (student and admin) ended up causing the system to behave so similarly, the admin and student user sequence were combined into one general sequence diagram that demonstrates overall system behavior.

Interaction Diagram



Requirements Analysis Matrix (Table)

The following matrix/table lists which functional and nonfunctional requirements were successfully implemented by the end of the project.

Requirements in red were not implemented with accompanying description explaining why the requirement was not implemented.

Functional Requirements

Requirement #	Description
F1	System will be used by multiple end users and administrators simultaneously.
F2	System will use a front end graphical user interface for all users.
F3	System will persist data through the use of a relational database management system (MySQL).
F4	System will perform detailed class scheduling activities using an integer program solver, to be chosen by the development team.
F5	All users will be required to authenticate their access to the system through an assigned username and password.
F6	Output data will be presented to all end users through the software's front end graphical user interface.
F7	The system must be able to dynamically calculate, optimize, and re-optimize schedules for one to many students and administrators simultaneously.
F8	In additional to normal processing of schedules for students and administrators, the system must be able to run in 'shadow mode' where administrators can set different analytical specifications than those run by end user students.

	Reason not implemented: Due to time constraints, the design team was unable to meet this goal. Instead, all changes and requested courses will be immediately active in the database, and will call a new run to the LP solver. Also, the LP solver will always submit its solution to the database.
F9	A list of courses offered in a given semester is required as input to the system.
F10	A list of available professors for teaching in a given semester is required as input to the system.
F11	A list of courses to be taught for a given semester is required as input to the system.
F12	A list of desired classes (desiderata) from all students is required as input to the system.
F13	A complete list of all student statuses, including seniority rankings for each student, is required as input to the system.
F14	A list of teaching assistants available to teach for the given semester is required as input to the system.
F15	An optional list of which teaching assistants will be assigned to each class may be provided to the system. The rest will be assigned by the LP solver.
F16	A list of courses effectively offered for a given semester is required as output from the system.
F17	A list of students assigned to each offered course is a required output from the system.
F18	A list of teacher - course assignments is a required output from the system.
F19	A list of teaching assistant – course assignments is a required output from the system.
F20	System will use Java as its primary programming/development language.

F21	System will use JSP pages as its primary web interface to end users.
F22	System will use Java Swing toolset for GUI interface design. **Reason not implemented:** Instead of writing the program interface in Swing, we used a web interface with the interface built dynamically through Spring MVC.

Non-Functional Requirements

Requirement #	Description
NF1	System shall be available for use to all users within each semester's registration period.
NF2	System shall be available for use by administrators both within and outside a semester's registration period.
NF3	System shall have bi-weekly to monthly maintenance with no more than a 2 hour outage window. **Reason not implemented:** This non-functional requirement was beyond the scope of this project and we were unable to sufficiently ensure that it was met.
NF4	System shall be accessible to all users through access to the internet.
NF5	System shall have a MTTR (mean time to recovery) of four hours. **Reason not implemented:** This non-functional requirement was beyond the scope of this project and we were unable to sufficiently ensure that it was met.
NF6	System should allow for switching out the core scheduler processing engine easily.
NF7	System will only be available to students already admitted to The Georgia Institute of Technology OMSCS program.

NF8	System will have full database backups done on a daily basis. **Reason not implemented:** For this smaller prototype, which was tested in a virtual environment, we did not implement a daily backup system due to time constraints.
NF9	All system core/backend components should run on a Linux platform.
NF10	System should be designed for maximum availability.
NF11	System must use the most up to date information available, and present that information to all end users and administrators.
NF12	System must be able to offer services to all Georgia Institute of Technology OSMCS students simultaneously during each registration period. This would be considered the "peak" or maximum number of simultaneous users.
NF13	Users of the system must be able to execute Java on the machine where the GUI is running from.
NF14	Java functionality must be written in such a way that if needed, it can be easily ported from one platform to another.
NF15	System will initially be developed to work within the CS6310 Ubuntu provided virtual environment.
NF16	System will use Spring MVC for the web framework.

2. Linking Software Elements to Design Elements

The following table breaks down each major component of the design diagrams and lists their associated Java object, JavaServer Pages (JSP), and SQL files.

Web Interface	Spring MVC	Core Application (LP Solver)	Database
admin.jsp login.jsp student.jsp welcome.jsp	AdminController.java GATechAuthController.java HomePageController.java StudentController.java AdminDao.java StudentDao.java AdminDaoImpl.java StudentDaoImpl.java DatabaseConnector.java AdminCourseForm.java StudentCourseForm.java AdminService.java StudentService.java AdminServiceImpl.java StudentServiceImpl.java AdminCourseFormValid.java StudentCourseFormValid.java	Course.java Professor.java Student.java TA.java Problem.java University.java Main.java	create_database.sql createtables.sql create_users.sql drop_database.sql drop_tables.sql drop_users.sql grant_privleges.sql load_data.sql

Within this table, the Problem.java file contains all the algorithms that builds the constraints and objectives for the Gurobi solver and initiates the solver.

The University.java file moves data between the database and the Problem.java object and contains many of the methods listed in our interaction diagram, such as populateCourses(), populateStudents(), populateProf(), and solve().

3. Reflection upon what worked and did not work during Implementation

What worked during implementation

Choosing Gurobi over GLPK or any other LP solver allowed the team to build on and reuse a substantial amount of code from Project 1, which made some parts of the constraint construction quicker. Gurobi is much more user friendly and easily understood than GLPK and contributed to the timely success of Project 4. In addition to simply completing the project, Gurobi also runs quicker and outputs a solution much faster than GLPK.

Another very important tool that we all used for the project was GitHub. With GitHub we were able to store all our base code in a centralized location and easily make necessary changes through the development process. Since GitHub is such a valuable and flexible tool, it allowed us as a team to work more efficiently and effectively together.

The VM was used in order to avoid installation and configuration issues. The choice of including a MySQL database in our class virtual environment was a good one, and overall MySQL worked very well. The database is easy to use and is very effective. Creating the database objects needed for the project was easy, loading the provided data into the database was quick, and interacting with the database for extracting data to provide to the project was efficient and effective.

What didn’t work during implementation

Due to the fact that the learning curve associated with Spring was quite steep, using a different application to web interface framework from Spring MVC would have reduced the amount of confusion and sped up progress.

We were having difficulty modeling student course preferences and student seniority in Gurobi. However, after a lot of work, we found the bug and were able to correct it. The bug consisted of issues pulling down from the database: multiple records were being pulled for each student and numbers were being mixed up when they were being inserted into the data structures. As a result, the wrong students were being prioritized.

It was too difficult to model specializations as there was only one specialization currently available that can be satisfied by the courses currently available.

From a data model perspective, there were several tables initially included in the design document that were not needed. These tables were thought to be important in the overall implementation, but in the end it was decided that they were not needed at all. There was no impact from removing these tables from the end product.

4. Technical elements of the course that were most useful to our project

Project 1 was useful for learning how to use and implement Gurobi, which was the core optimization engine for our project. Project 2 was useful for learning about software architecture and design for the project and what architecture and design principles to apply to the project. Project 3 was useful for collaborating and taking the best from each of our projects.

The lectures were useful for teaching us about class diagrams and all the other relevant diagrams such as sequence diagrams. Learning to diagram our project on a high level in advance of implementation was beneficial because it helped to visualize the solution and alter or reorganize components before it affected written code.

The class diagrams we reviewed from the lectures and built for Project 1 and Project 2 were essential for defining the objects and composition of the Java application and how it fit with the web framework and LP solver. The StateChart Diagram assignment was useful for getting practice designing statecharts and it helped us track how the system would respond to user inputs. Architectural styles lectures and assignment 6 were useful to the end project, because it helped us choose the various architectures we used in this project. The Design Patterns lecture was perhaps the most useful of all lectures in the class. Had that lecture been assigned earlier in the course it would have greatly impacted our design approaches. As such we were too far into implementation to alter our approach.

5. Alternative Design Approaches

If we had to start the project all over again, an alternative design approach would be that our team would have utilized a different web interface to application framework other than Spring MVC. This is due to the fact that Spring turned out to have a steep learning curve that exceeded the expertise and time availability of our team. It was difficult for members of the team to contribute to this portion as there was one expert for Spring.

The price of Gurobi may become a factor in the long-term. If the software license proves to be too expensive, especially for a non-academic user, the project might have to be modified to use a different engine such as GNU Linear Programming Kit (GLPK), a free open source solver. Because the architecture used for this project is designed for abstraction and reuse, switching the solver or web interface to a new library should not affect the other components of the project and require minimal reprogramming.

Regarding the back end database implementation for this project, the MySQL database implementation worked well. We feel that this was the case in part due to the small dataset size and the limited number of concurrent users. However, a major design change that we recommend in cases where the software implementation supported a large number of concurrent users or a larger data set would be to use an Oracle database to store data instead of MySQL. Oracle databases are designed to support enterprise level applications on a large scale, whereas MySQL is designed more for non-enterprise level applications and smaller scale use.

DDL Appendix

Below is the DDL used to create the database schema and tables inside the local VM provided for this class.

This same SQL can be found in the project subdirectory, in the ‘database’ subdirectory, inside the ‘create_tables.sql’ file.

In order to correctly populate the MySQL database inside the class VM to run the project, you must run the “runme_create_all.sh” shell script.

Specific directions on this can also be found inside the ‘readme.txt’ file found in the same ‘database’ subdirectory.

```
cat create_tables.sql
--
-- First get into the scheduling database;
--

use scheduling;


--
-- Now create the tables needed for the project;
--

--
-- USERS table
--

create table users
(
gt_id varchar(20) not null,
first_name varchar(30),
last_name varchar(30),
password varchar(20),
constraint users_pk primary key (gt_id)
);


--
-- STUDENTS table
--

create table students
(
gt_id varchar(20) not null,
good_standing_flg integer,
total_credit_hours integer,
constraint users_pk primary key (gt_id)
);


--
-- LOGIN_HISTORY table
--

create table login_history
(
gt_id varchar(20) not null,
login_date date,
hostname varchar(50),
constraint login_history_pk primary key (gt_id, login_date)
);


--
-- COURSE_RESRC_LU table
--

create table course_resrc_lu
(
resource_type_cd varchar(5),
resource_desc varchar(200),
constraint course_resrc_lu_pk primary key (resource_type_cd)
);


--
-- COURSE_RESRC table
--

create table course_resrc
(
gt_id varchar(20),
course_ref_nbr varchar(15),
resource_type_cd varchar(5)
);


--
-- COURSE_RESRC_HIST table
--

create table course_resrc_hist
(
gt_id varchar(20),
course_ref_nbr varchar(15),
resource_type_cd varchar(5)
);

--
-- COURSE_RESRC_AVAIL table
--
```

```
create table course_resrc_avail
(
gt_id varchar(20),
course_ref_nbr varchar(15),
term_cd varchar(10),
resource_type_cd varchar(5),
availability_flg varchar(1)
);


--
-- COURSE_CATALOG table
--

create table course_catalog
(
course_ref_nbr varchar(15),
course_id varchar(15),
course_cd varchar(5),
course_nbr varchar(10),
course_title_desc varchar(200),
credit_hours integer,
class_size_limit integer,
foundational_flg varchar(1),
constraint course_catalog_pk primary key (course_ref_nbr)
);


--
-- COURSES_OFFERED table
--

create table courses_offered
(
course_ref_nbr varchar(15),
term_cd varchar(10),
gt_id varchar(20),
constraint courses_offered_pk primary key (course_ref_nbr)
);


--
-- COURSE_REQUISITE table
--
```

```
create table course_requisite
(
course_ref_nbr varchar(15),
reqste_course_ref_nbr varchar(15),
prerequisite_flg integer,
constraint course_requisite_pk primary key (course_ref_nbr, reqste_course_ref_nbr)
);


--
-- COURSE_ASSIGMENT table
--

create table course_assignment
(
course_ref_nbr varchar(15),
gt_id varchar(20),
term_cd varchar(10),
constraint course_assignment_pk primary key (course_ref_nbr, gt_id)
);


--
-- COURSE_ASSIGMENT_HIST table
--

create table course_assignment_hist
(
course_ref_nbr varchar(15),
gt_id varchar(20),
term_cd varchar(10),
constraint course_assignment_pk primary key (course_ref_nbr, gt_id)
);
--
-- TERM_CODE_LU table
--

create table term_code_lu
(
term_cd varchar(10),
term_cd_desc varchar(200),
constraint term_code_lu_pk primary key (term_cd)
);
```

```
--
-- STUDENT_COURSE_HIST table
--

create table student_course_hist
(
gt_id varchar(20),
course_ref_nbr varchar(15),
term_cd varchar(10),
credit_hours integer,
constraint student_course_hist_pk primary key (gt_id, course_ref_nbr, term_cd)
);


--
-- STUDENT_COURSE_REQUEST table
--

create table student_course_request
(
gt_id varchar(20),
course_ref_nbr varchar(15),
term_cd varchar(10),
course_req_rank_cd varchar(5),
constraint student_course_req_pk primary key (gt_id, course_ref_nbr, term_cd)
);



--
-- STUDENT_COURSE_REQUEST_HIST table
--

create table student_course_request_hist
(
gt_id varchar(20),
course_ref_nbr varchar(15),
term_cd varchar(10),
course_req_rank_cd varchar(5),
constraint student_course_req_pk primary key (gt_id, course_ref_nbr, term_cd)
);
```