Author: Pratibha Sundar

Course Management Backend System Design Document

1 Table of Contents

2 Introduction

In this document, we describe a course management backend api project implemented using google app engine in Python 2.7+. The following sections explains what are the functional requirements, how the data models are designed and apis implemented, what are the test cases and which of them are addressed. In addition, it also suggests how this project can be enhanced. The current system is deployed and tested in local environment even though an app id (course-manage) has been created for it at Google Cloud Platform for deploying at GAE.

3 Functional Requirements

The functional requirements given for this project are as follows:

a. Allows user to add /edit/remove student information:
    i. Student Id, name, department, address, phone number, study list, etc.
b. Allows user to add/edit/remove course information:
    i. Course id, department, name, instructor, time, place, etc.
c. Allows user to add/edit/remove course schedule:
    i. Quarter, department, courses provided in the quarter, etc.
d. Allows user to enroll/drop a student to/from a course
e. Allows user to list students enrolled in a course
f. Allows user to list courses a student enrolls in
g. Allows user to list courses a department provides in a specific quarter

NOTE:
The functional requirement “edit”(Update) a record is not implemented yet. The datastore method “put()” provided by Google App Engine is an upsert operation meaning that the command is the same for both insert and update of an entity. However, there is a need to distinguish between the two operations from application point of view to avoid any accidental edits to the data. The code for update is similar to that of insert with some minor changes in handling dependencies.

4 Technical Specification

This section explains what platforms and versions are used in the implementation.

	Name	**Version**
Platform	Google App Engine SDK	1.9.40
Programming Language	Python	2.7
IDE	Eclipse	Luna SR 2 (4.4.2)

Plug-in	PyDev	

5 System Design

This section of the document explains the data model design and the features of APIs accessing those models. First we will describe the data models and their dependencies and then explain the functionalities of APIs.

5.1 Data Model Design

This section reviews data model design required by the course management system. Models are created using NDB Client Library. There are five models each containing department, student, course, schedule and enrollment data.

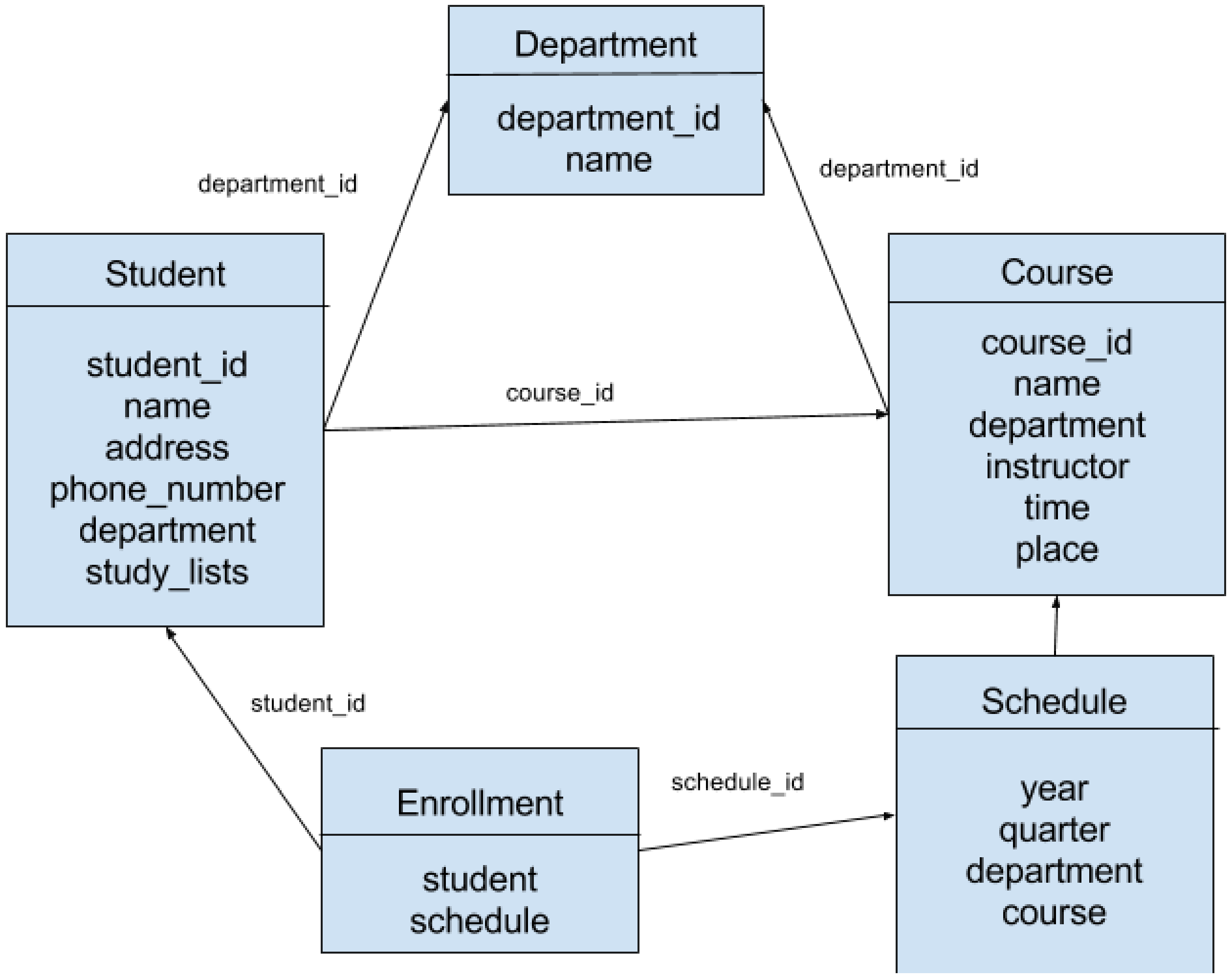

Figure 1: Entity Relationship Diagram for Course Management Backend Data Model using NDB lib

5.1.1 Department

This model contains list of departments. This kind does not have any dependency to any other kinds.

Name	**Property Type**	**Description**
department_id	ndb.IntegerProperty()	Uniqely identifies a department entity
name	ndb.StringProperty()	Department name

Table 1: Properties of Department kind

5.1.2 Course

This model contains list of courses provided by a department and for each course the instructor name, when the course is taken and the venue of the course. This kind is dependent on Department kind for accessing department ID and name. The department ID has to be validated against Department kind.

Name	**Property Type**	**Description**
course_id	ndb.IntegerProperty()	Uniquely identified a course entity
name	ndb.StringProperty()	Course name
department	ndb.KeyProperty(kind=Department)	Holds the key of department the course belongs to. The key must be in keys of department entity
instructor	ndb.StringProperty()	Instructor name
time	ndb.StringProperty()	Holds time range
place	ndb.StringProperty()	Course venue

Table 2: Properties of Course kind

5.1.3 Student

This model contains all student information such as student ID, name, their address, contact number, which department a student belongs to and the list of courses that student has been enrolled in. This kind is dependent on Department kind in the same way as Course kind. In addition, this kind also stores for each student, list of courses the student has enrolled in. Hence, the course IDs also need to be validated against Course kind.

Name	**Property**	**Description**
student_id	ndb.IntegerProperty()	Uniquely identifies a student entity
name	ndb.StringProperty()	Student name
address	ndb.StringProperty()	address/residence of student
phone_number	ndb.IntegerProperty()	Phone number
department	ndb.KeyProperty(kind=Department)	Department the student belongs to
study_lists	ndb.KeyProperty(kind = Course, repeated = True)	Lists of courses the student has been enrolled

Table 3: Properties of Student kind

5.1.4 Schedule

This model contains list of courses scheduled for a quarter. A quarter is uniquely identified by the quarter number and the year. The course ID, quarter and year uniquely identifies a schedule. The department ID is retrieved from the Course kind and is not part of the input when adding or deleting.

Name	**Property**	**Description**
year	ndb.IntegerProperty()	Year the schedule belongs to
quarter	ndb.IntegerProperty()	Quarter the schedule belongs to
department	ndb.KeyProperty(kind=Department)	Department the course of current schedule belongs to
course	ndb.KeyProperty(kind=Course)	Course scheduled

Table 4: Properties of Schedule kind

5.1.5 Enrollments

This model contains list of all the enrollments for a given quarter. Enrollment information includes which student is enrolled in what course. This kind is dependent on Student and Schedule kinds. When a schedule is deleted, all enrollments to that schedule automatically gets deleted. When a student is enrolled in a course, the course also gets added to the student's study lists.

Name	**Property**	**Description**
student	ndb.KeyProperty(kind=Student)	Student being enrolled for the given schedule
schedule	ndb.KeyProperty(kind=schedule)	Schedule given

Table 5: Properties of Enrollment kind

5.2 Api Design

This section is an overview of all the APIs that exposes the data model for course admin operations such as add/view/edit course information.

5.2.1 Department API

This API exposes Department entities for operations like add and delete departments. This is not part of functional requirement but is needed to populate department data which is required by other kinds that references the department kind.

Method	**Input**	**Affected Kinds**	**Dependencies**
Insert	department_id (required) name	Department	
Delete	Department_id (required)	Department	Kinds: Courses, Student, Enrollment Need to delete/update department info in these kinds before deleting Department

Table 6: Department API methods

5.2.2 Course API

This API exposes Course entities for operations like add and delete courses offered by a department. This API addresses functional requirement (b). [Assumption: Course ID is unique across departments]

Method	**Input**	**Affected Kinds**	**Dependencies**
Insert	course_id (required) name department_id instructor time place	Course	Check for valid department ID

Delete	course_id (required)	Course	<u>Kinds:</u> Student, Schedule, Enrollment To delete from Course, delete course from Schedule. Which automatically deleted courses in Student and Enrollment

Table 7: Course API methods

5.2.3 Student API

This API exposes the Student kind for operations like add and delete student information. It addresses functional requirement (a) and (f). Student ID needs to be unique to a student across all departments.

Method	**Input**	**Affected Kinds**	**Dependencies**
Insert	student_id (required) name department_id address phone_number	Student	Validate Department ID
Delete	Student_id (required)	Student	<u>Kinds:</u> Enrollment Check if student is enrolled in any courses or not. Delete enrollments before deleting Student.
View	Student_id (required)	Student (Read-only)	<u>Description:</u> Lists courses a student enrolls in

Table 8: Student API methods

5.2.4 Schedule API

This API exposes the Schedule kind for operations like adding/deleting a schedule. It addresses a functional requirement (c) and (g).

Method	**Input**	**Affected Kinds**	**Dependencies**
Insert	course_id (required) year (required) quarter (required)	Schedule	<u>Kind:</u> Course Check for valid course ID. Fetch department ID from Course Kind
Delete	course_id (required) year (required) quarter (required)	Schedule Enrollment Student	When a schedule gets deleted, enrollment to that course also gets deleted automatically. Hence, it affects both Student and Enrollment kinds.

View	department_id (required) year (required) quarter (required)	Schedule (Read-only)	Description: List courses offered by a department in a quarter

Table 9: Schedule API methods

5.2.5 Enrollments API

This API exposes the Enrollment kind for operations like add and delete and addresses functional requirement (d) and (e). This kind has dependency over Schedule and Student kinds.

Method	**Input**	**Affected Kinds**	**Dependencies**
Insert	student_id (required) course_id (required) quarter (required) year (required)	Enrollment Student	When a student is enrolled in a course, update the student's study lists with that course
Delete	student_id (required) course_id (required) quarter (required) year (required)	Enrollment Student	When a student is dropped from a course, delete that course from the student's study lists
View	course_id (required)	Enrollment (Read-only) Schedule (Read-only)	Description: List students enrolled in a course

Table 10: Enrollments API methods

6 Test cases

This section describes different test cases performed and their status on each of the APIs for unit testing. It also describes any improvements that can be done on the current implementation and suggests enhancements to the system.

		Department API		**Course API**	
Operation	**Test Case**	**Description**	**Status**	**Description**	**Status**
Insert					
	Insert Entity	Check duplicate ID	ok	Check duplicate ID	ok
		Insert entity	ok	Insert entity	ok

	Constraints (Ref)		ok	Department (ID)	ok
	Check for required fields		Not Done		Not Done
Delete	**Delete Entity**	Validate ID	ok	Validate ID	ok
		Delete entity	ok	Delete entity	ok
	Dependencies (Delete / Check if not found)	Course (Check) Student (Check) Schedule (Check) Enrollment (Check)	ok	Student (Check) Schedule (Check) Enrollment (Check)	ok

Table 11: Test cases status for Department and Course APIs

		Student API		**Schedule API**		**Enrollment API**	
Operation	**Test Case**	**Description**	**Status**	**Description**	**Status**	**Description**	**Status**
Insert							
	Insert Entity	Check duplicate ID	ok	Check duplicate ID	ok	Check duplicate ID	ok
		Insert entity	ok	Insert entity	ok	Insert entity	ok
	Constraints (Ref)	Department (ID)	ok	Course (ID)	ok	Student (ID) Schedule (ID)	ok
	Dependencies (Update)					Student (Study lists)	ok
	Check for required fields		Not Done		Not Done		Not Done
Delete	**Delete Entity**	Validate ID	ok	Validate ID	ok	Validate ID	ok
		Delete entity	ok	Delete entity	ok	Delete entity	ok
	Dependencies (Delete / Check if not found)	Enrollment (Check)	ok	Student (Check) Enrollment (Check)	ok	Student (Delete study lists)	ok
View	**View entities**	Courses taken by a Student	ok	Courses offered by a department	ok	Students enrolled in a course	ok

				for a given quarter			

Table 12: Test cases status for Student, Schedule and Enrollments API

6.1 Improvements

This section talks about how this api can be improved and where it lacks.

1. Implement Update/Edit functionality: Though this is a part of the functional requirement, it is not currently supported.
2. Check for required inputs: For each API, if a required input is missed, it should raise an error message
3. Validate input values: The current implementation does not check for valid input values for each property and implementing this avoids invalid/junk entries into data. Examples of properties where this feature can be implemented are:
    a. ID fields: Fix the length of ID fields such as Student ID or Course ID or Department ID and check if input matched the length (E.g. all department IDs has to be 3 digit integers). More complex, check for specific format of IDs, for e.g. student ID can be a concatenation of 3 digit department with 3 digit roll number.
    b. Phone number: phone number has to be 10 digits long with specific area code etc.
    c. Year: Year has to be within a valid range. In this context, current year, previous year and next year e.g. (2015, 2016, 2017)
    d. Place: place has to be from a given set of values (It has to represent a legitimate place within the university/college campus).
4. Time property in Course Kind: Currently, the time is a StringProperty which doesn't check for valid time values or enforce specific format. Alternatively, this can be designed as repeated DateTimeProperty or TimeProperty containing both start and end times.
5. Use debug mode for testing: Currently, debugging is done using log messages. Instead, add breakpoints and deploy the app in debug mode for troubleshooting avoid unnecessary clutters caused by log messages.
6. Code reusability. Need to modularize code and reuse them for simple operations like delete/add. Most of the functions such as retrieving an entity given an ID is implemented as an class methods. Instead implementing them as functions improve reusability of code.

6.2 Enhancements

This section describes application enhancements required to complete the ReSTful course management application.

1. Complete the functional requirements (Update)
2. Deploy the app to GAE than hosting it local
3. Automate test cases and provide the test script
4. Add Front End Code with Admin Login feature

7 References

[1] Google App Engine Python Docs, https://cloud.google.com/appengine/docs/python
[2] NDB Cheat Sheet, https://docs.google.com/document/d/1AefylbadN456_Z7BZOpZEXDq8cR8LYu7QgI7bt5V0Iw/edit
[3] Stack Overflow, http://stackoverflow.com
[4] Udacity, https://www.udacity.com/