Modular Marking Sheets

As a requirement for this project, we were asked to produce marking sheets that were modular. This means their contents had to be dynamically populated based on an input value (course code)and they must be simple to update and create new marking sheets year on year. It was also a requirement that whatever database structure we chose to use to implement the modularity described above, this database must keep a historical account of the marking sheet used in every single year.

Implementing Modularity

Looking at the marking sheet, it became clear to us that the sheet could essentially be designed as a series of 'Content' objects. A content object would consist of an outline of how a certain section of a students paper should be marked in a table format and an input section made up of three input boxes where a corrector could write a comment outlining their reasoning for awarding a certain grade, award a mark to that section and weight the section relative to the rest of the paper. A Content object can be visualised in the following rough format:

Grade Bracket	Grade Bracket	Grade Bracket	Grade Bracket
Grade bracket guideline	**Grade bracket guideline**	**Grade bracket guideline**	**Grade bracket guideline**

Area for comments	Weight
	Mark

Show above is a content object as you can see the outline part is the table with the blue banner and the input section is made up of the three text boxes underneath the outline table. We thought that if we could make a series of these where the table and the input section could be paired, associated with a degree and constructed dynamically when the marking sheet for that degree was required, then we would have achieved modularity. At the point of realising that this was the task at hand, it became clear o us that Django was a good framework for the project as one of it's main selling points is it's simple integration with databases which would surely come in handy for storing and retrieving these content objects shown above.

Django's model structure:

Django comes with a fantastic api for making and managing a MySQL database. The way that Django is structured, each part of the website is its own application (e.g in the context of this project tracking and marking are technically individual applications). Inside each application there is a file called models.py . This is the file where all the database code is kept. The code in this file is python which is transpiled into MySql commands by the Django framework. These MySql commands get executed when you run the following commands:

$ python manage.py makemigrations
$ python manage.py migrate

But for convenience, we have a shell script called updateDB.sh in the project which will execute these two commands to update the database.

Adding a Table to the model structure:

In Django, adding a table to an application's model involves:

1. Declaring the table as a class
2. Specifying the column names of the table and declaring the datatypes of each column.
3. Running the updateDB commands shown above.

An example of a table declaration in the models file is shown below:

```
class Student(models.Model):
    studentid = models.PositiveIntegerField(null=False)
    surname = models.CharField(max_length=30, null=False)
    firstname = models.CharField(max_length=30, null=False)
    email = models.CharField(max_length=40, null=False)
    studentnumber = models.IntegerField(null=False)
    degree = models.PositiveIntegerField(null=False)
    year = models.CharField(max_length=9, null=True)
    doing_compsci_project = models.PositiveIntegerField(null=True)
```

In this example, we declared a table named students. All tables must take models.Model as an input argument. The column names for this table are {studentid, surname,firstname,email,studentNumber,degree,year, doingCompSciProject} and you can see the associated datatypes on the right hand side of each assignment. The arguments passed to each field are constraints. It is possible as shown above to mandate that a field must not be null. Here you can also declare that a column is a primary key or set up a foreign key relationship here (**see docs**) . You'll notice no primary key is specified above. In this case Django has a default counter that it will specify as the foreign key by default.

Storing content objects in a database.

In order to store each content as in the database, it was necessary that we create a table to store every content possible in the marking sheet. Below you can see an image of the Content table from the marking.models.py file.

```
class Content(models.Model):
    contentId = models.IntegerField(null = False)
    title = models.CharField(max_length = 200)
    description =  models.CharField(max_length = 10000)
    inputBox = models.CharField(max_length = 1000)
    defaultWeight = models.IntegerField(default=20)
    weighted = models.BooleanField(default=True)
    degreeId = models.IntegerField(null = False)
    degreeName = models.CharField(max_length = 200)
    section = models.IntegerField(null = False)
    year = models.IntegerField(null = False)
    orderOnPage = models.IntegerField(default=1)
```

As you can see, from the Django class above, each Content object has the following form:

ContentID	Title	Description	inputBox	defaultWeight	Weighted
Int	varChar(200)	varChar(1000)	varChar(1000)	Int	Bool

degreeId	degreeName	section	Year	orderOnPage
Int	varChar(200)	Int	Int	Int

contentID:(depreciated) originally used as pk but makes sense to use Django default
title: each section comes with a title string e.g "problem statement and motivation"
description: HTML string associated for outline part of content
inputBox: HTML string associated with the input section of the content
defaultWeight: how much the section is weighted by default (corrector can change)
Weighted: specifies whether a section is weighted or not
degreeId:the degree code that the marking sheet is for e.g CS is TR033 - code is 33
Section: there are two sections that the content objects can go in. This is 1 or 2
year: specifies the year in which projects will be marked e.g 2018-2019 yr is 2019
orderOnPage: specifies in what order content objects will be ordered (1 first)

Content API

String toString():

Parameters : None
Function: Returns the title of the content as a string.

Void print():

Parameters: None
Function: prints out the title of the content.

Void resetOrderOnPage():

Parameters: None
Motivation: will be called when contents are being inserted into database. Initially all contents will be 0, before another function give the contents an integer value for the order that they will appear on the page
Function: sets the orderOnPage to 0

Void updateTitle(String newTitle):

Parameters:a string which will be set last the new title
Function: sets input string as Content.title

Marking.Models API

<QuerySet> filterByYear(int year):

Parameters: an integer representing the year whose contents you are querying

Function: returns a querySet representing the set of Content objects that pertain to a certain year

Void listContent():

Parameters: none

Function: prints out the title of all the entries in the listContent table

Void resetFormOrder():

Parameters: None

Function: iterates over all of the Contents and sets their orderOnPage to 0.

Motivation: used when formatting a marking sheet. The order that the blobs were in before is not necessarily the order they will appear in now. Each blob will have it's order on the page set to 0. This is updated when the marking sheet format is refreshed by a different script.

Content getByTitle(String Title)

Parameters: the title of the content you are searching for

Function: returns the content specified by the input title. If no such content exists, then the function returns None.

<QuerySet> fetchCourseByYear(int courseCode, int year):

Parameters:

1) the course code (e.g cs = 33)
2) The year you are looking to retrieve content information for

Function: returns all the content objects for a specific course in a specific year as a Django QuerySet.

<QuerySet> fetchContentsForPage(int courseCode, int year):

Parameters:

1) the course code (e.g cs = 33)
2) The year you are looking to retrieve content information for

Function: returns all the content objects for a specific course in a specific year that will be viewable on the marking sheet for that course. Having a Content.orderOnPage that is greater than 0 implies that that content object will be displayed on the marking page.

Void printContentSet(<QuerySet> contentObjects):

Parameters: takes in a list of Content Objects

Function: prints out the title of every Content Object in the input set