**Design**

This chapter outlines the design decisions for this Level 4 project. As it was previously described, Weave is based on top of the standalone Java application IWE. This affects to a great extent these decisions.

Storage of data

The scope of this Level 4 project does not include the creation of a web based authoring interface. This means that the design of Weave must ensure an easy and efficient way of storing the data for the examples. This data is produced in the form of XML files by the IWE author interface. In addition, Weave must support consistent modification of existing examples. A weakness of the XML files used by IWE was discovered during the process of defining the design of this aspect of Weave. Since these XML files are easy to access and modify by the authors of worked examples and this is typically much easier than modifying the worked examples using the authoring interface, one may be tempted to make changes to the examples manually. However, there are multiple XML files containing the same information about features of the same object. Given that a feature is modified in one of the files, consistency about this feature must be ensured. It is trivial having to find the same feature across multiple files and in the end correct modification is not guaranteed. Django provides an object relational model (ORM) database functionality which is the preferred method for managing the data for the worked examples because it guarantees consistency. The relationships between different objects are expressed via foreign keys. These can be explored on the ER Diagram on figure X. In addition, this adds a level of reliability that the data is valid because internal integrity checks are made before saving an object to the database. Further advantage is that this is a cleaner representation of the same data for which the Django administrator interface allows an easy means of exploring. This interface supports manual creation and modification of objects. Thus the problems of adding changes to existing examples easily and consistently are overcome by the database.

Architecture

The architecture of Weave consists of three distinct tiers as visualised on figure X.

* Presentation tier. This is the top level of the overall architecture also known as the client side web interface. It defines the appearance of the website by rendering HTML and CSS and provides means for users to interact with the application. The clients are in the form of web browsers. On every interaction, they send requests to the server in the form of Ajax GET or POST requests to ensure asynchronous communication between the client and the server.
* Django Middleware. In this tier the requests from the client are parsed and translated into ORM requests- a language understandable by the backend tier. These requests are passed forward to get or store the information in the request from/in the database. After the backend generates the response, the middleware is responsible for passing it back to the presentation tier in the form of Http response.
* Data layer. This tier represents the database in which all the information used or generated by Weave is stored in the form of objects. On GET requests the backend tier responds with an object meeting the criteria specified in the ORM request. On POST requests, the database creates a new object with the features specified in the request and stores it in the database.

User interface

Weave is intended for two distinct groups of users, each with their different needs. This is why the section on the user interface is split into two subsections describing the user interface for students and the user interface for teachers.

Student User Interface

The initial definition of this Level 4 project was to translate the student part of IWE into a more easily accessible online version. The evaluation of IWE proved that the design used for the student interface in the application is well accepted by students. This interface can be seen on Figure X.

It can be seen that careful consideration about the layout of the interface has been taken while developing the IWE student interface.

Directory structure (implementation)

1. The root directory contains:

* the database
* the population script which reads the XML files and translates them into ORM objects
* the manage.py file which serves for the management of the application

2. The /exerciser directory contains:

* \_\_init\_\_.py
* admin.py file which is used to register ORM entities with the administrator interface
* context\_processor.py file ???
* disable.py file which is used to disable Cross-site request forgery (CSRF) tokens when needed. These are tokens that ensure safe client-server interaction. They need to be disabled when the client is guaranteed to make safe requests.
* forms.py file which serves for ensuring that the information provided by the user when making POST requests is valid and performs the necessary integrity checks before saving it to the database
* models.py file which defines the entities stored in the database
* urls.py file which defines the url mappings for each web page
* views.py file where all requests from clients are served and the appropriate templates are rendered

3. templatetags

* \_\_init\_\_.py
* poll\_extras.py

4. exercises\_system\_project

* \_\_init\_\_.py
* settings.py file which defines the settings for the application
* urls.py sets the url mapping for Weave. It specifies the base for all the url patterns stored in exerciser/urls.py
* wsgi.py file which stores the Web Server Gateway Interface (WSGI) configuration for Weave. WSGI specifies a simple and universal interface between web servers and web applications of frameworks for Python

5. static

This directory contains the files needed for defining the appearance and behaviour of the user interface.

6. templates/exerciser

This directory contains the html files for the web pages.