COMP2001 : Information Management & Retrieval

TUTORIAL SUPPORT APPLICATION: APPLICATION FACT SHEET

Accompanying this fact sheet are podcasts via Panopto labelled "How to..".

REQUIREMENTS

All applications have functional and non-functional requirements. Our functional requirements are best expressed in the user stories below.

FUNCTIONAL REQUIREMENTS

Our application is going to be a tutorial support application. The following user stories are a starting point for a minimum viable product – but we will only get as far as implementing the first one. The others are provided for you to see how the bigger picture emerges and how it fits in

1. As a student I wish to alert the lecturer that I need some help
2. As a student I wish to see that my request for help has been acknowledged
3. As a lecturer I wish to acknowledge the request
4. As a lecturer I wish to reset the request

NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements are characteristics of a system which are not the activities described above. There are a number of different types listed below.

Technical requirements:

Our application is going to be a web-based application that interacts with the Hue lights in SMB109. It will run on a web server and be written in either PHP or ASP.NET. The database will be remotely hosted on either socem1 or proj-mysql – dependent upon the technology stack selected. The interface for the student will be via a desktop browser, the interface for the lecturer however will be tailored for a mobile phone.

Performance requirements: Response times are not within the scope of this sample application. These are determined by the network set up within the lab.

Usability requirements: The interfaces will conform to accessibility rules as per the W3 validator.

Reliability requirements: These are outside of the scope of the application and are dependent upon the set-up of the labs and servers.

Security requirements: OWASP top 10 vulnerabilities are to be examined and addressed where appropriate. Lecturers will be required to log in but students are not. Only name information will be available to the lecturer to view.

PLANNING

The next part of the process is to iterate through the user stories to plan out the details behind that user story. You can find out more about agile development planning from the reading list in the section labelled "planning".

Problem solving, computational thinking, pseudocode – are all processes you would use to iterate through the user story to break it down. Our first high level cut may be this:

1. Browser is opened at Request Help page
2. Student enters details for request
    a. Name of student
    b. Date and Time of session
    c. Module code
    d. Row number in lab
    e. Seat number in row
    f. Few words about the issue
3. Student clicks button
4. Data saved to database
5. Alert triggered for Lecturer
6. Response returned from system to confirm correct entry

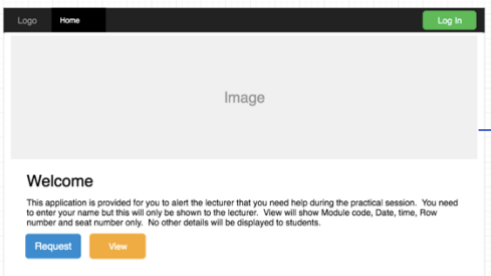
STORYBOARDS

Our storyboards allow us to model the application from the viewpoint of the end user. We can consider our usability aspects here, ensure our application complies with the law around accessibility and ensure an application with a good modern look and feel to it.

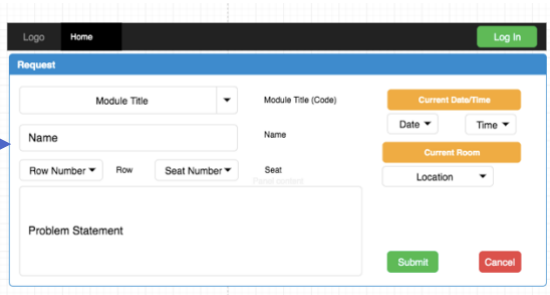
When starting to think about our storyboard we need to initially plan how the user will navigate through the application. We can take the pseudocode from above and start to plan how the interface might look.

In the example below I have used the Bootstrap template from Draw.io to create the storyboard. The sample below is quite small so the full size images are provided in the Unit 9 folder as a suggested layout.

Index.php



Request.php



Modelling is an important part of the system development. It helps to clarify and refine the requirements. Systems become very complex very quickly and so the model helps simplify things – after all a picture can convey a thousand words. However, one model does not fit all and you need different models that serve different parts of the system. Models are our memory aids.

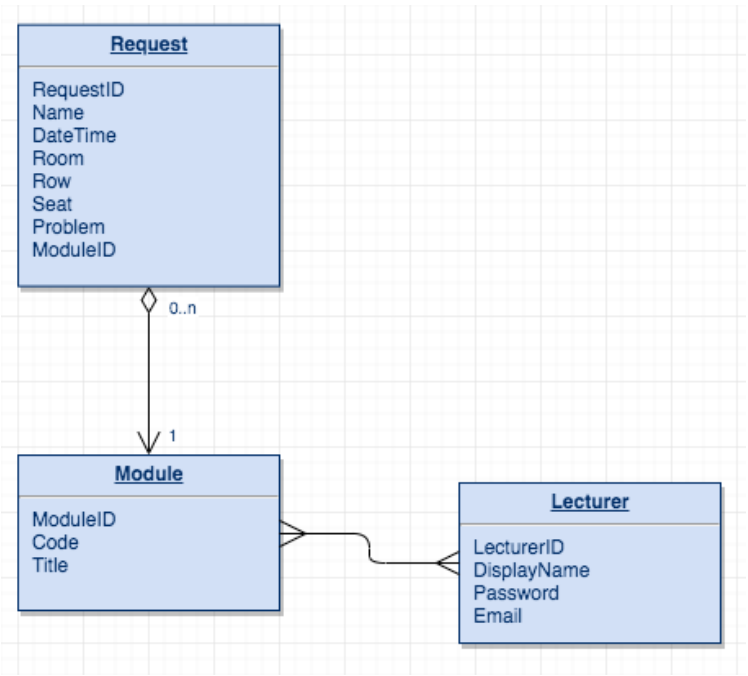
The models we will look at are the logical models – the ones that look at the functional requirements.

Remember that there is always more than one way to solve a problem and this is only ONE suggestion on how this might be done. This is not a "model" answer that is perfect – they do not exist.

ENTITY RELATIONSHIP DIAGRAM

This set of models considers our data in the database layer. We look at what needs to be stored, we normalise it and then consider both top down and bottom up to check the quality.

As a reminder, top down design starts by identifying the data sets and then defines the data elements for each of those sets. The process involves the identification of different entity types and the definition of each entity's attributes. The bottom up approach first identifies the data elements (items) and then groups them together in data sets. In other words it first defines the attributes and then groups them to form entities. You may want to remind yourself of your 1st year Database module when you did Entity Relationship diagramming (top down) and normalisation (bottom up).



The ER diagram will help us create our SQL statements that build our database tables.

Here we only have one many-to-many relationship to resolve, that between Module and Lecturer.

From this we can move to data dictionary which in turn will help us create our SQL statements to create the database.

NOTE: Whilst we could use the Object-Relational Mapping (ORM) capabilities of the framework we might use, it is best to ensure we have an SQL file representing our tables, views and triggers. If you did choose to use the ORM – then ensure you export your database as an .sql file.

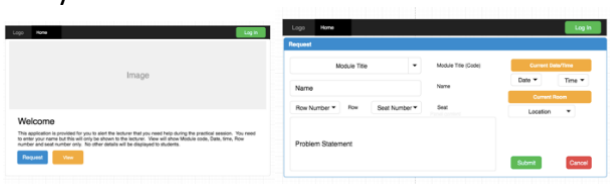
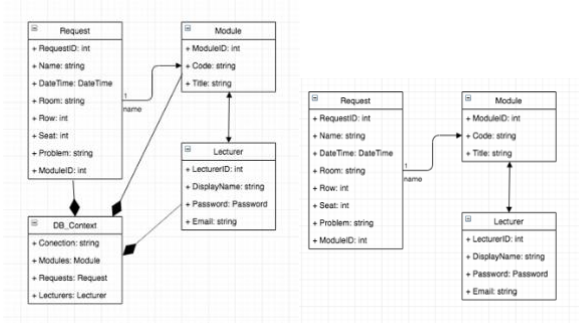
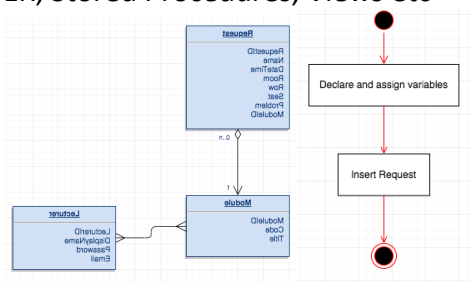
UML DIAGRAMS

These diagrams model the structure and behaviour of our code. From here we can determine our architecture and how the different parts of the application interact. We must consider how

established patterns help us fit our application together and apply those patterns to make good quality code.

Software Architecture is the "big picture" structural aspects of an information system. Two of the most important aspects are the division of the software into classes and the distribution of those classes across processing locations and specific computers.

Given our three-layer architecture, we can see the following:

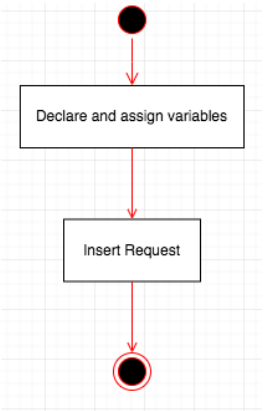
Interface Layer	Storyboards 
Business Layer	Class diagrams, Unit of Work, Repository 
Data Layer	ER, Stored Procedures, Views etc  .

The UML diagrams of most interest are the Class diagram and the Activity Diagram. We can use the Activity diagram for representing our stored procedures. We could use a table as follows to represent the properties for the stored procedures:

Operation Name	Enter_Request
Involved Tables	Request
Overview	Request entry
Pre-condition	Request table has been created
Post-condition	A new record is added into Request table
Input parameters	@Name, @Room, @Row, @Seat, @Problem, @ModuleID
Output parameters	None

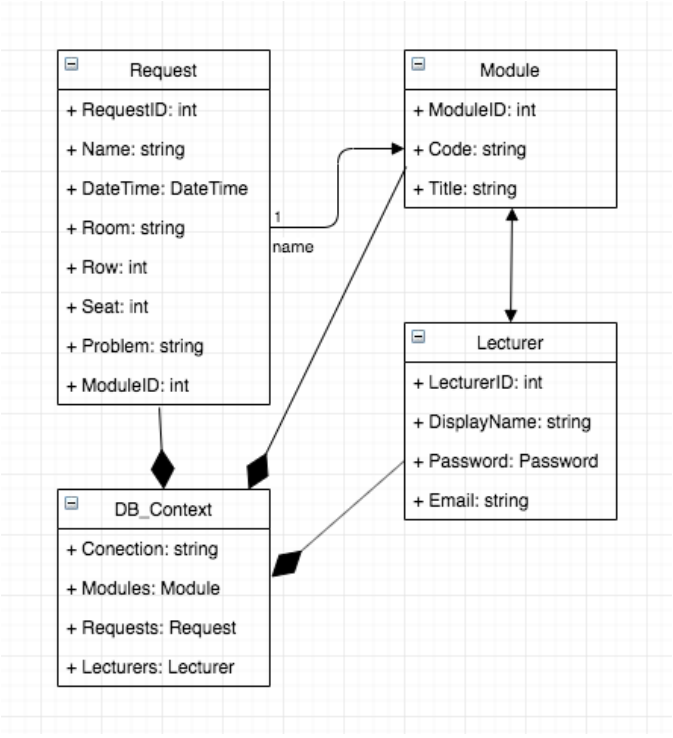
Return Value	None

A simple activity diagram will suffice for a stored procedure. The example below shows the Enter_Request stored procedure listed above.



CLASSES

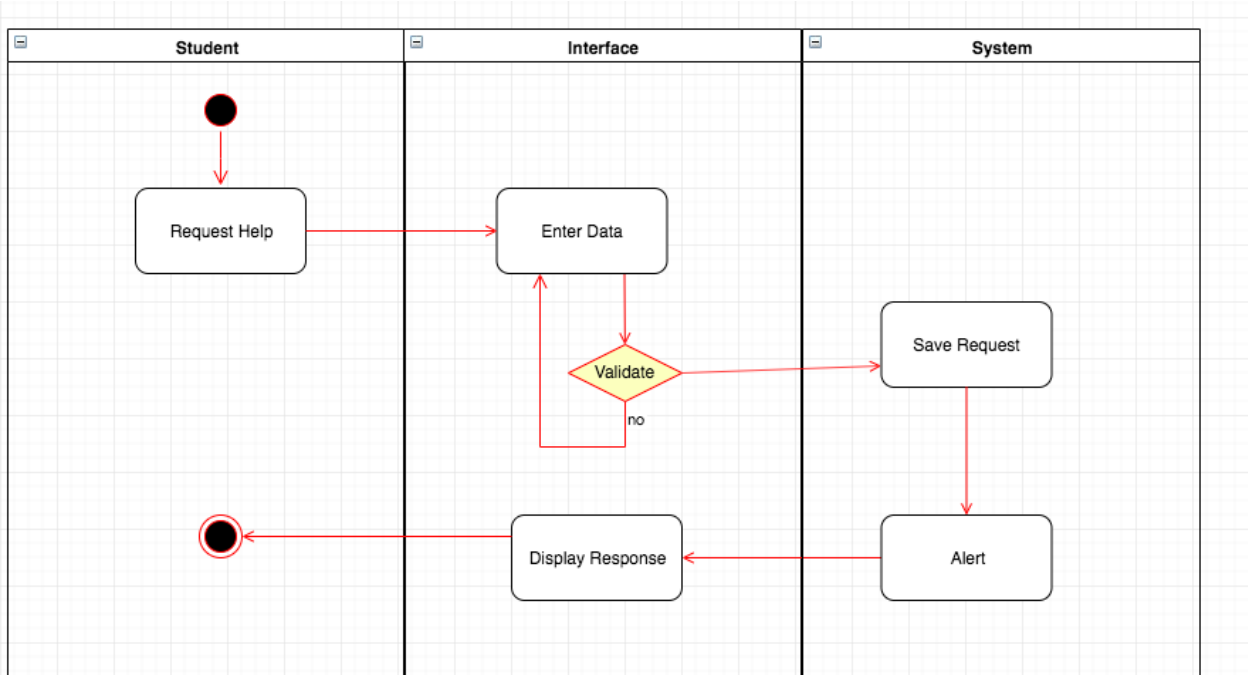
Our preliminary class diagram involves the repository pattern and the unit of work. Our initial draft might look like this.



Not all classes within a system will be specified. This is just to get us started. The objective of building these diagrams is to understand the business need and so focus is often only on specifying the classes that fulfil that need. In our case, User story 1.

ACTIVITY DIAGRAMS

Having considered the entities and classes within our system, we need to consider now how they interact. Here we turn our pseudocode into an activity diagram.



This combination now should give us what we need to think about the flow of code within our application.

SEQUENCE DIAGRAM

From here we might finally explore a sequence diagram to force us to think about the MVC pattern combined with the Unit of Work.

In this example we interact with elements of the system before calling out to the RESTful API which is the lights. This is external to the system.