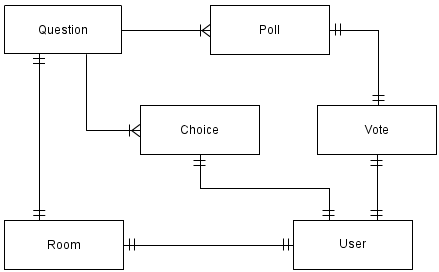
# Database Design

When designing any application, the first step is usually to define the way that data will be stored. This is most easily done on paper, using rough entity-relationship diagrams to map out the way that different database objects (entities) will interact. In Django, the database layout is known as the *models.*

*Our group spent some time discussing the best way to design the database, with most of the effort spent deciding on the polling structure. The way that polls and votes are to be stored is the most complicated part of the application, as we decided that historical data would need to be kept for post-processing and analysis once the conference/debate is finished.*

*After some revisions (which are included in the appendix in photographic form), the final entity-relationship diagram was decided on as follows:*

**

### ***Room***

*Central to the application is the* ***Room entity. This represents a conference room/debate. The Room entity has an owner, which is represented internally in Django using the User model.***

### ***Question***

***The room also has one (and only one) Question object. This represents the topic that the room is based around, and stores the question that members vote upon every time a period ends.***

### ***Poll***

***The question then references many Poll objects. These are the “instantiations” of the question for a particular period. This allows each Poll to store the votes associated with it, whilst not being deleted once the period has ended.***

### ***Choice***

***The question object also references many Choice objects. These represent the choices that are available for participants to vote on.***

### ***Vote***

***Once a participant casts a vote, a new Vote object is created. This object references both the Choice and the Poll that was taking place at that particular moment. It also references the user that cast the vote.***

## ***Justification***

***The above implementation was chosen over a number of alternatives. These alternatives include using an “archive” version of the poll, choice and vote objects in order to keep historical data.***

***The given ERD was decided upon as being the neatest (and simplest) way of solving the problem, and hinges on the fact that a Vote objects stores a reference to both the choice and the poll upon which it was cast.***

# ***API Design***

***The messaging aspect of the application, in particular, relies heavily on Javascript. Without this, messages would not be delivered to the user until the browser page is refreshed. This was deemed early on to be wholly unacceptable.***

***Therefore, an easily-consumable API was required, to allow simple acquisition of data, and to also allow data to be posted asynchronously (in the case of sending a message or casting a vote, for example).***

## ***JSON***

***The group decided to use JSON (Javascript Object Notation) to serialise the data when communicating between client and server. This is because it is such a simple, human-readable protocol. Also, it is extremely easy to serialise and de-serialise data using built-in Python and Javascript libraries.***

***The alternatives to representing the data in JSON, would be to use a format such as XML, which carries it's own advantages and disadvantages. For example, XML is less human-readable (and thus it is harder to debug an error in the application), though it is possible to give the data meaning, which is not an option with JSON – the application designer simply has to know which fields represent what. Fortunately, that is an advantage that we have with this project, so JSON offers a much easier format to work with.***

## ***API Methods***

***To achieve full functionality on the front-end of the system using Javascript, the following methods were implemented in the API:***

### ***rooms/get\_info***

***Returns relevant information pertaining to a particular room/debate. Includes the following data:***

* ***number of members***
* ***list of members***
* ***messages (all or only unread)***
* ***current room mode (conferencing or voting)***
* ***time before next poll***

### ***rooms/send\_message***

***Posts a message to a given room/debate. Returns the following:***

* ***list of unread messages for the sending user (including the message posted)***

### ***rooms/reset***

***Resets a given room/debate, deleting all current poll data and resetting the countdown to the original value (length of the period).***

### ***polling/get\_info***

***Retrieves the information for the poll that is currently in use for a given room. Returns the following information:***

* number of votes
* whether the current user has voted on the current poll or not
* poll results so far (the number of votes for each possible choice)

### ***polling/cast\_vote***

***Casts a vote for a given choice in the current poll for a given room.***