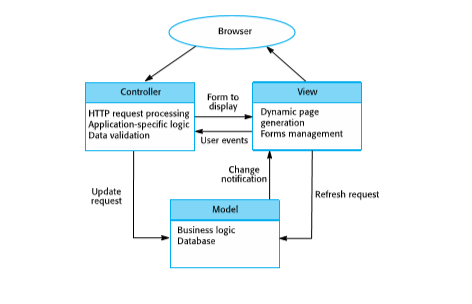
System Architectural Design

The chosen system architecture is the MVC pattern. This separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The model component manages the system data and associated operations on that data. The view component defines and manages how the data is presented to the user. The controller component manages user interaction with key cap clicks and mouse clicks for example, and passes these interactions to the view and model.

The advantages of this include: this allows the data to change independently of its representation and vice versa. It supports presentation of the same data in different ways with changes made in one representation shown in all. However, a disadvantage of this is that this can involve additional code and code complexity even for simple models and interactions.

This is how the architecture looks for a web based application.



Alternative Designs are:

The repository architecture. This is where all data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository. This pattern is used when you have a system in which large volumes of information are generated that has to be stored for a long time. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool. An advantage of this is where components can be independent—they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently. However, the repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult.

The Layered architecture. This organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. It is used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsible for a layer of functionality; when there is a requirement for multi-level security. It allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system but providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.

The system will use a selection of drop down menus, prompts and clickable options to guide the user to the selection and options they so desire. These selections do require a mouse and keyboard as specified in the requirements section of the documentation.