TASK 3: ARCHITECTURE DESIGN

1 ARCHITECTURAL APPROACH

Our team's MWC product will be a website so we choose to use **MVC** architectural approach **(Model - View - Controller) model**.

1.1 THEORY

About MVC design pattern

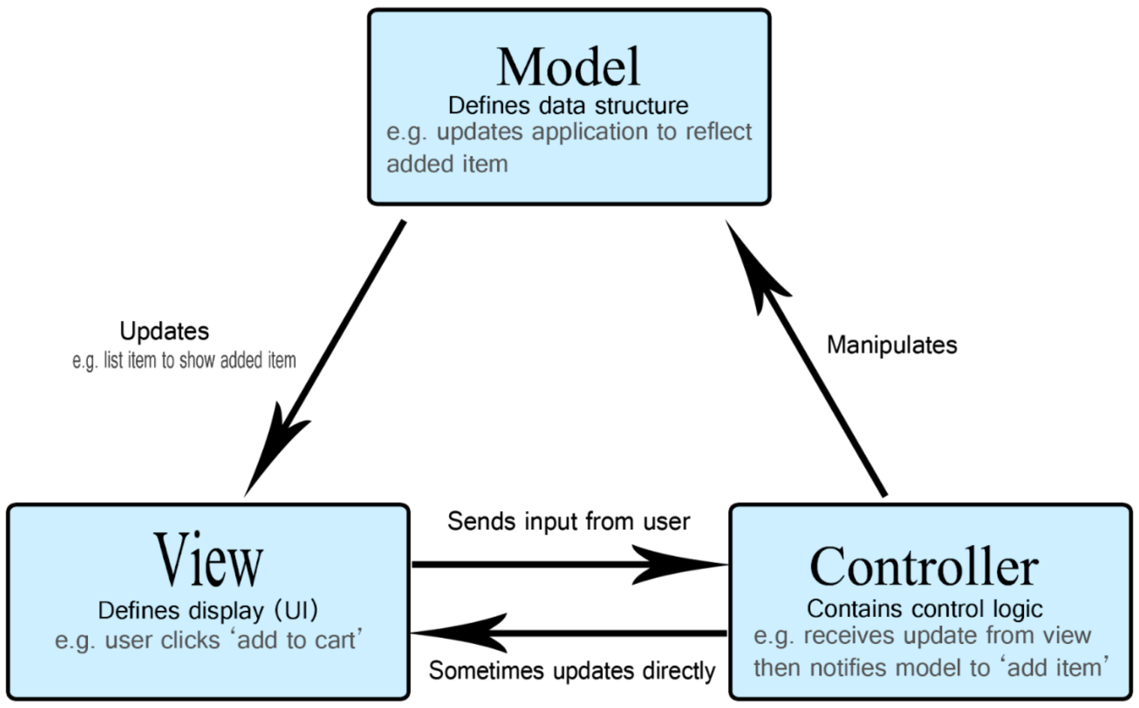
MVC (Model-View-Controller) is a pattern in software design which specifies that an application consists of data model, presentation information, and control information. It emphasizes a **separation between the software's business logic and display**. This "separation of concerns" provides for a better division of labor and improved maintenance. Some other design patterns are based on MVC, such as MVVM (Model-View-Viewmodel), MVP (Model-View-Presenter), and MVW (Model-View-Whatever).

To explain clearly, we took a simple shopping list app which used the MVC model to design. Now we go into **3 parts** of the MVC design pattern.

<u>The Model</u>

The model defines what **data** the app should contain. If the state of this data changes, then the model will usually notify the view (so the display can change as needed) and sometimes the controller (if different logic is needed to control the updated view).

Going back to the shopping list app, the model would specify what data the list items should contain — item, price, etc. — and what list items are already present.


Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

The View

The view defines **how the app's data should be displayed.**

In the shopping list app, the view would define how the list is presented to the user, and receive the data to display from the model.

The Controller

The controller **manipulates** the database in model according to events triggered by external sources (i.e view) in a **logical** way. Since the view and model are connected, the action of the controller will automatically change the view.

So for example, a shopping list could have input forms and buttons that allow us to add or delete items. These actions require the model to be updated, so the input is sent to the controller, which then manipulates the model as appropriate, which then sends updated data to the view.

You might however also want to just update the view to display the data in a different format, e.g., change the item order to alphabetical, or lowest to highest price. In this case the controller could handle this directly without needing to update the model.

1.2 SOLUTION

We will divide our system into **4 big modules** discussed in task 1:

1. Account
2. Task Assignment
3. Contact
4. MCP’s state updation

1.2.1 Account Module:

Input: Username and Password (Log in ID).

Function: This module will take in user login information (user name, password, ..), compared with the registered ID in the database and return false if no ID is found or user’s data if ID is found. We do not have to care about the new register since every employee’s account is provided by back officers.

Output: User’s authentication and ID.

1.2.2 Task Assignment Module

Input: User ID, Task Information (workers' ID, vehicle's ID and its status (if any), MCP's ID, routine (if any), date and time)

Function: This module will take user ID from the system to identify if the users are a back officer or not, if false then they cannot access this module, if true they will next consequently insert task's information when the system processes. The task will be assigned to the workers and shown in the calendar according to the workers' ID and date and time in the input.

Output: An updated calendar with proper tasks' assignment.

1.2.3 Contact module

Input: Message, Sender ID, Receiver ID

Function: This module will take the messages from the sender and send the message through the network for the receiver.

Output: Text output at the receiver's view.

1.2.4 MCP's state updation

Input: MCP's id, new state

Function: This function will process the MCP whose ID matches the input ID, discard its old state and update the new one.

Output: New state of MCP

1.2.5 MVC adaptation

1. Account Module:

Model: Database of register's User ID

View: Authentication UI (for user to type in their log in ID)

Controller: Receive users' login ID -> Ask Model to retrieve login ID from database, if Model cannot find such an ID, directly update View to show Reject notification, if Model finds such an ID, Model sends notification to View to display user's data view of that ID.

2. Task Assignment Module:

Model: Calendar containing all tasks

View: Calendar UI for back officers to assign

Controller: Receive assignment information from back officers, update Model then distributed the new calendar (discussed further in 3.2)

3. Contact module:

Model: Chat log of users

View: Chatbox between users

Controller: Receive messages from sender, update it into Model, Model updates that message in both user's data and then notify the View to update newly sent data.

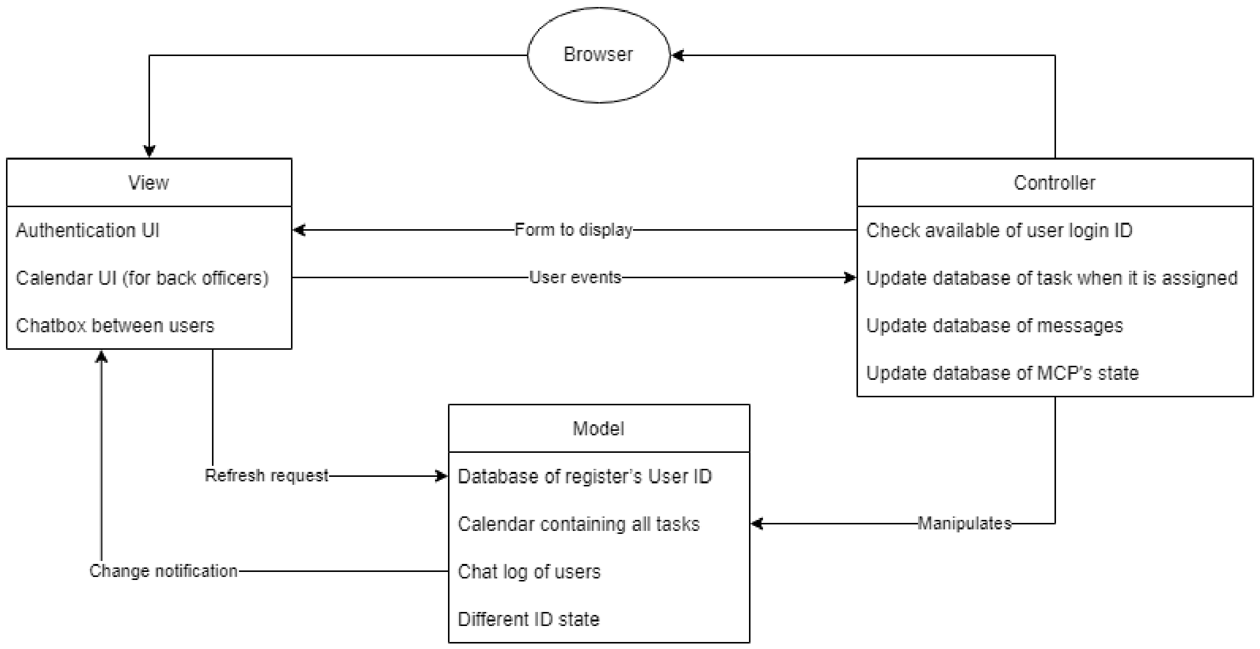
4. MCP's state updation:

(This updation will be performed by a sensor so we need no UI)

Model: Different ID state.

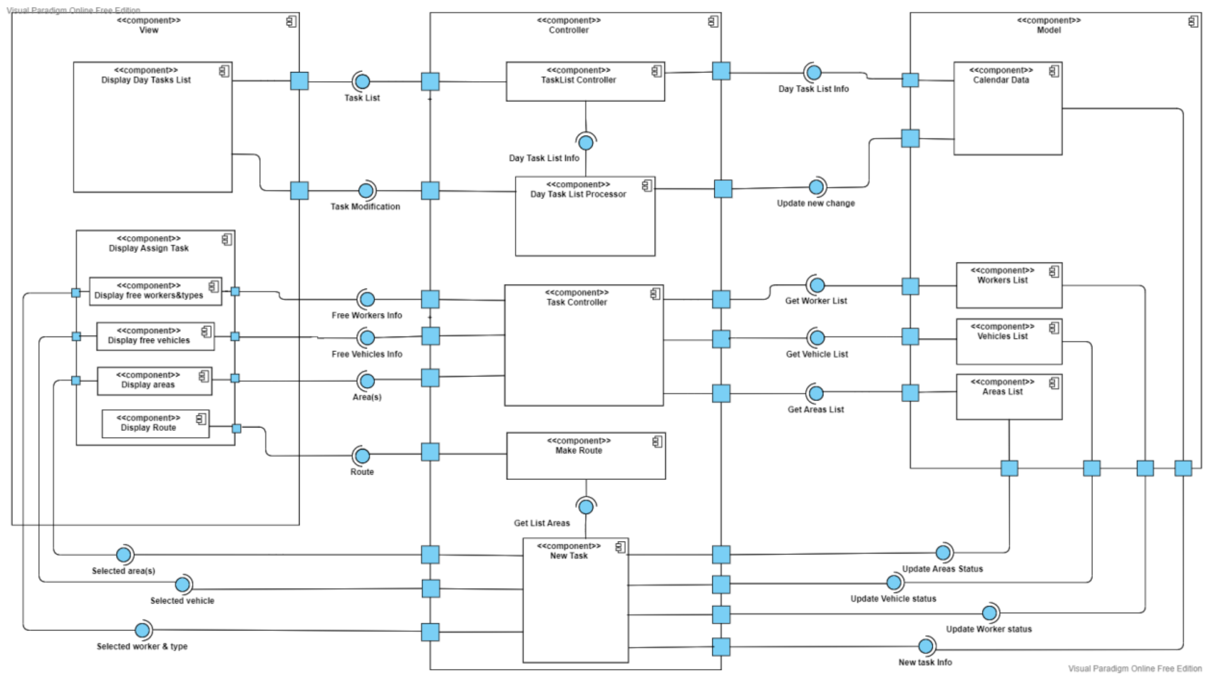
View: No UI is needed

Controller: Receive state from MCP, update it to Model


Browser
View
Authentication UI
Calendar UI (for back officers)
Chatbox between users
Controller
Check available of user login ID
Update database of task when it is assigned
Update database of messages
Update database of MCP's state
Form to display
User events
Model
Database of register's User ID
Calendar containing all tasks
Chat log of users
Different ID state
Refresh request
Change notification
Manipulates

2 IMPLEMENTATION DIAGRAM

2.1 IMPLEMENTATION DIAGRAM


Visual Paradigm Online Free Edition
<<component>> View
<<component>> Display Day Tasks List
Task List
<<component>> Controller
<<component>> TaskList Controller
Day Task List Info
<<component>> Model
<<component>> Calendar Data
Day Task List Info
Task Modification
<<component>> Day Task List Processor
Update new change
<<component>> Display Assign Task
<<component>> Display free workers&types
<<component>> Display free vehicles
<<component>> Display areas
<<component>> Display Route
Free Workers Info
Free Vehicles Info
Area(s)
Route
<<component>> Task Controller
Get Worker List
Get Vehicle List
Get Areas List
<<component>> Workers List
<<component>> Vehicles List
<<component>> Areas List
<<component>> Make Route
Get List Areas
<<component>> New Task
Selected area(s)
Selected vehicle
Selected worker & type
Update Areas Status
Update Vehicle status
Update Worker status
New task Info
Visual Paradigm Online Free Edition

2.2 DESCRIPTION

At the calendar user-interface, BackOfficer chooses one day in the calendar. Component *TaskList Controller* gets assigned-task information on selected day and displays information in component *Display Day Tasks List.*

In the situation that **BackOfficer** modifies a task that has been assigned before, all modification will be processed in component *Day Task List Processor*, then component *TaskList Controller* obtains adjusted information and displays it in component *Display Day Tasks List*. Modification is then updated into component *Calendar Data* when it has been confirmed.

In the situation that BackOfficer needs to create new tasks in one day, component *Task Controller* will get information from *Workers List, Vehicles List*, *Areas List* and render it on the interface for **BackOfficer** to choose. Then, the component *New Task* will combine that information. Component *Calendar Data*, *Workers List, Vehicles List*, *Areas List* will be updated via component *New Task*.

In the situation that the worker type of the *New Task* is **Collector**, component *Make Route* will get the areas list selected from the *New Task* to make the route.

END OF REPORT