6 Task 2.3

Question 6. Draw a class diagram of Task Assignment module as comprehensive as possible

Solution

6.1 Theoretical basis: Class diagram

In UML, class diagrams are one of six types of structural diagram. Class diagrams are fundamental to the object modeling process and model the static structure of a system. Depending on the complexity of a system, we can use a single class diagram to model an entire system, or we can use several class diagrams to model the components of a system.

Class diagrams are the blueprints of the system or subsystem. We can use class diagrams to model the objects that make up the system, to display the relationships between the objects, and to describe what those objects do and the services that they provide.

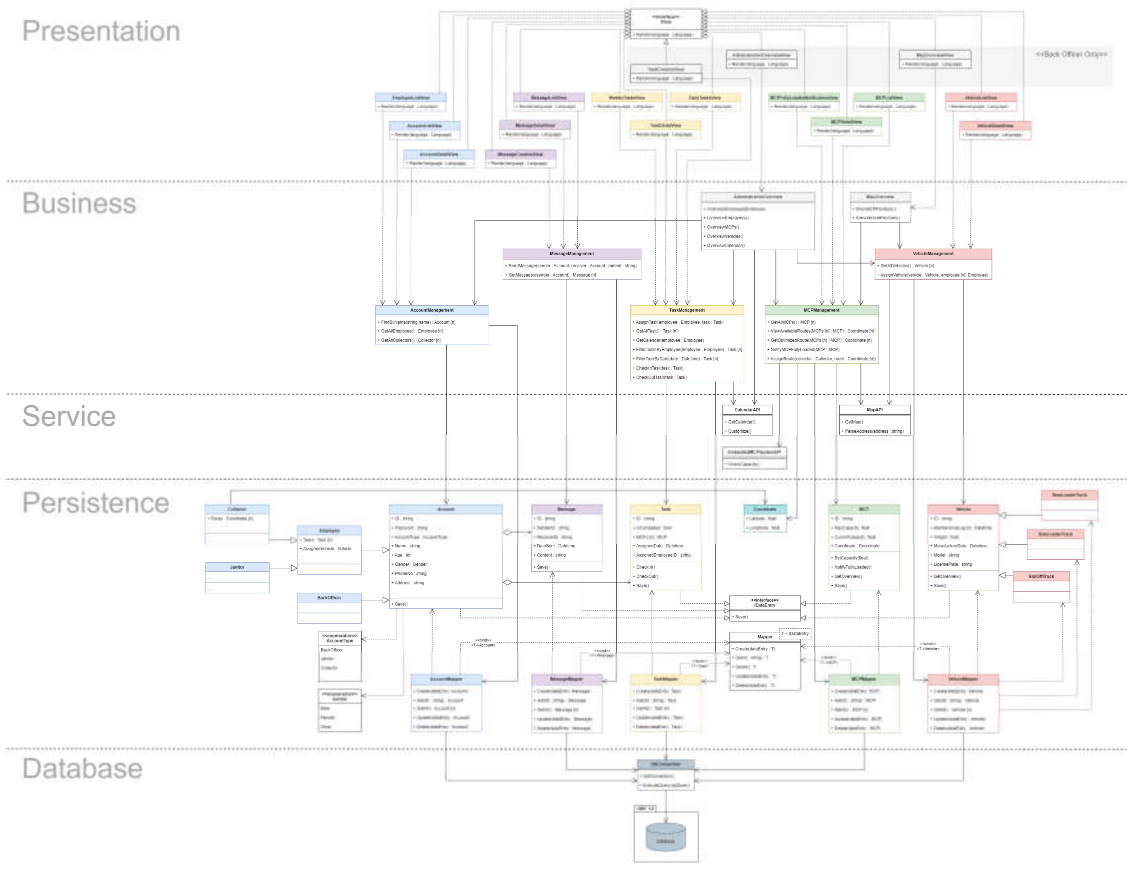
Popular model elements of a class diagram:

- **Classes:** In UML, a class represents an object or a set of objects that share a common structure and behavior. Classes, or instances of classes, are common model elements in UML diagrams. The structure of a class in class diagram consists of: class's name, attributes (optional) and operations (optional).
- **Objects:** In UML models, objects are model elements that represent instances of a class or of classes. We can add objects to our model to represent concrete and prototypical instances. A concrete instance represents an actual person or thing in the real world.
- **Packages:** Packages group related model elements of all types, including other packages.
- **Enumerations:** In UML models, enumerations are model elements in class diagrams that represent user-defined data types. Enumerations contain sets of named identifiers that represent the values of the enumeration. These values are called enumeration literals.
- **Data types:** In UML diagrams, data types are model elements that define data values. We typically use data types to represent primitive types, such as integer or string types, and enumerations, such as user-defined data types.
- **Subsystems:** A subsystem groups elements that together provide services such that other elements may access only those services and none of the elements themselves. And while packages allow us to partition our system into logical groups and relate these logical groups, subsystems allow us to consider what services these logical groups provide to one another.

Some concepts that are widely used in class diagram:

- **Dependencies:** A dependency from a source element (called the client) to a target element (called the supplier) indicates that the source element uses or depends on the target element; if the target element changes, the source element may require a change.
- **Realizations:** A realization from a source element (called the realization element) to a target element (called the specification element) indicates that the source element supports at least all the operations of the target element without necessarily having to support any attributes or associations of the target element.
- **Generalizations:** A generalization between a more general element and a more specific element of the same kind indicates that the more specific element receives the attributes, associations and other relationships, operations, and methods from the more general element.
- **Aggregation:** Aggregation is whole-part relationship between an aggregate, the whole, and its parts. This relationship is often known as a *has-a* relationship.
- **Composition:** Composition, also known as composite aggregation, is a whole-part relationship between a composite (the whole) and its parts, in which the parts must belong only to one whole and the whole is responsible for creating and destroying its parts when it is created or destroyed. This relationship is often known as a *contains-a* relationship.
- **Association ends:** An association end is an endpoint of the line drawn for an association, and it connects the association to a class. An association end may include any of the following items to express more detail about how the class relates to the other class or classes in the association:
  - Role name.
  - Navigation arrow.
  - Multiplicity specification.
  - Aggregation or composition symbol.
  - Qualifier.

6.2 Class diagram of Task Assignment module



Follow this link for a PNG with better resolution, or this link for the diagram file.

Explanation: This class diagram uses the Layered Architecture approach with 5 different layers. Every layer is made up from various components, each handles or assists a specific task of task assignment module. To fully describe our class diagram, we will use 2 different approaches for explanation:

- **Horizontal approach:** In this approach, we will provide a brief description of each layer.

  1. **Database layer:** This layer is used to store the data for the system. The database will store the data and the DBConnection class will connect this layer to persistence layer for retrieving and updating data.
  2. **Persistence layer:** Business layer and database layer are not allowed to communicate directly due to the principle of layered architecture. This layer facilitates the interaction between business layer and the database by retrieving data from database and passing them to business layer as well as sending requests from business layer to database layer.

3. **Service layer:** This layer will be the place to store external APIs like map API, which provides an interface for working with map, and use them to establish a connection between components of the business layer and outside resources such as map, MCP system, calendar, etc. This connection allows components in business layer get use of such resources, thus greatly reduce their workload and difficulty in implementation.
4. **Business layer:** This layer is the core of the whole system. It contains all the logic behind the operation of task assignment module. Business layer plays a role as a "function", it receives requests from users, processes them based on the logic and produces the desirable outputs.
5. **Presentation layer:** This layer's responsibility is to display information on the users' screen based on the types of users and types of contents that the users wish to observe. Furthermore, it also has the ability to change displayed language based on users' preference.

- **Vertical approach:** As we can see in the diagram, components relating to each other can be grouped, with each group has its own functionalities. We color-coded the groups for better visualization. In this approach, we will explain the role of each group of components:

  1. **Account system:** This system deals with managing all accounts in the UWC 2.0 software. The managing process can be described as follow: First of all, business layer receives requests related to users' account. It then passes the requests to appropriate components. In this case, AccountManagement will handle the requests based on its logic. When invoked, AccountManagement will send a request to AccountMapper to retrieve or update data in database. If retrieving data is the case, the data are then put in the Account class to be encapsulated in an Account object before sent back to AccountManagement. The data are then processed and displayed on the screen using relevant classes in presentation layer if necessary.
  2. **Message system:** This system is responsible for processing messages (creating, sending, receiving, etc). Its managing mechanism is almost the same as that of account system except for one thing: Message class gets data from both MessageMapper and Account class (to indicate sender and receiver).
  3. **Task system:** This system is specified for tasks handling and is similar in management process to message system. It also take advantage of calendar API to work with calendar.
  4. **Coordinate:** This class is used to stored the coordinate of MCPs retrieved from the database and is used in MCP managing process.
  5. **MCP system:** This system is responsible for working with MCPs' data. It managing mechanism is almost the same as that of account system except for one thing: MCPManagement class gets data from both MCPMapper and Coordinate class (to determine MCPs' location). It also uses APIs from service layer to interact with map and sensors in each MCP.

6. **Vehicle system:** This system focuses on working with vehicles' data and has analogous mechanism to account system. It also uses map API to work with the map.
7. **Overview system:** Once it receives requests, it invokes corresponding management systems and APIs to get the overview data.
8. **APIs:** This group includes MapAPI, CalendarAPI and EmbededMCPSystemAPI. These API provide the interface to work with external resources like map, calendar or sensors in MCPs.
9. **DBConnection:** This class plays a role as the middleman between persistence and database layer. It receives requests from mapper classes and provide appropriate services to work with the database.