A red and yellow sign

Description automatically generated with low confidence

FINAL REPORT

Chu Manh Hai – 20184253 | ITSS

Appendix:

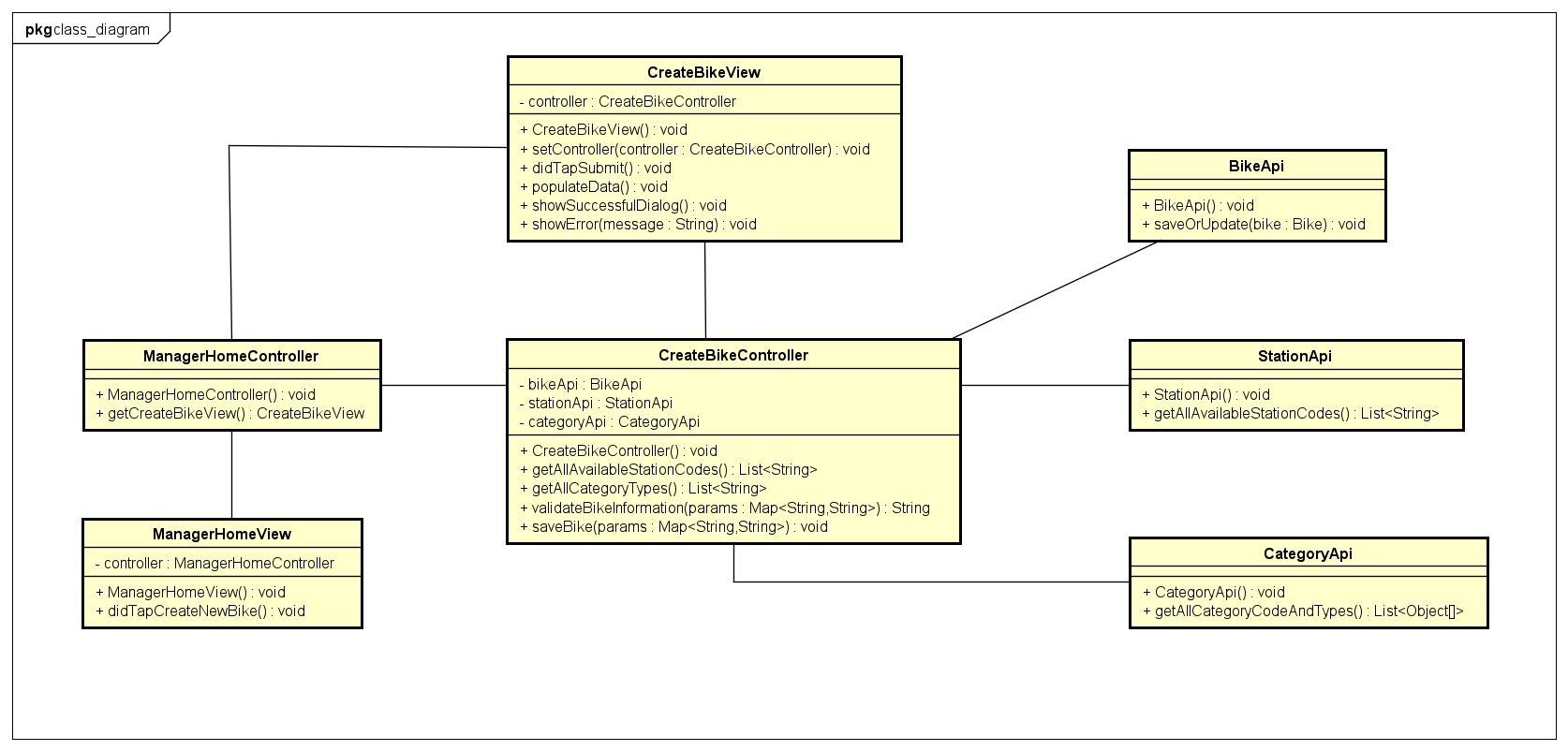
1. Use case specification
2. Use case analysis
3. GUI/subsystem design
4. Apply SOLID principles and patterns
5. Detail design
6. Self contribution
7. **Use case specification**

|  |  |  |  |
| --- | --- | --- | --- |
| **Use case Code** | UC002 | **Use case name** | Create bike’s information |
| **Actor** | manager | | |
| **Precondition** | Already login as manager role | | |
| **Main flow of event**  **(success)** | |  |  |  | | --- | --- | --- | | **#** | **Doer** | **Action** | |  | manager | Login successfully | |  | system | Show manager home screen | |  | manager | Click add “create new bike” button | |  | system | Show a form to add a new bike’s information | |  | manager | Fill in bike’s information | |  | manager | Click on submit button | |  | system | Validate information | |  | system | Notify a success message | |  | system | Redirect to list of bikes screen | | | |
| **Alternative flow of event** | |  |  |  | | --- | --- | --- | | **#** | **Doer** | **Action** | | 1a. | system | Ask user to login if he/she has not already login as manager role | | 6a. | system | If user click on “back” button, redirect to manager home screen | | 6b. | system | If user click on “exit” button, redirect to login screen | | 6c. | system | If form is not filled in completely, submit button will be disable | | 8a. | system | If information is invalid, show error message and require to re-enter information of invalid fields | | | |
| **Post condition** | No | | |

\*Input data of bike information:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Field** | **Description** | **Required?** | **Valid condition** | **Example** |
| 1. | name | Name of bike | yes | Not null/empty | Mount-S |
| 2. | manufacturer | Name of manufacturer | yes | Not null/empty | Kona Bikes |
| 3. | category | Type of bike | yes | Not null/empty | E-bike |
| 4. | weight | Weight of bike (kg) | yes | Positive Number | 34 |
| 5. | cost | Cost of bike ($) | yes | Positive number | 200 |
| 6. | station code | Station code | yes | Positive integer | 2 |
| 7. | Manufacturer date | Date of creation | yes | Date | 12-1-2021 |

1. **Use case analysis**
   1. Sequential diagram  
      A picture containing table

      Description automatically generated
   2. Class diagram  
      
2. **GUI design/subsystem design**
   1. Transition diagram  
      Diagram

      Description automatically generated
   2. Screen specification
      * Manager home screen

|  |  |  |  |
| --- | --- | --- | --- |
| Screen image | Control | Operation | Function |
| Graphical user interface, application  Description automatically generated | “Create new station” button | Click | Display a form to create new station |
| “Create new bike” button | Click | Display a form to create new bike |
| “exit” button | Click | Navigate back to login screen |

* + - Create new bike screen

|  |  |  |  |
| --- | --- | --- | --- |
| Screen image | Control | Operation | Function |
| Graphical user interface  Description automatically generated | Area to display form | Initial | Display a form for user to create new bike information |
| “Submit” button | Click | Submit a form to handle new bike creation |
| “back” button | Click | Return to manager home screen |
| “exit” button | Click | Return to login screen |

* + - fields attributes:

|  |  |  |  |
| --- | --- | --- | --- |
| Item name | Number of characters (bytes) | Type | Remarks |
| Name | 30 | String | Type |
| Manufacturer | 30 | String | Type |
| Category | 30 | String | Get from Database |
| Weight | 4 | Positive float | none |
| Cost | 4 | Positive float | none |
| Date | 30 | String | Must convert into string |
| Station code | 4 | Positive integer | Get from Database |

* 1. Subsystem design  
     Diagram

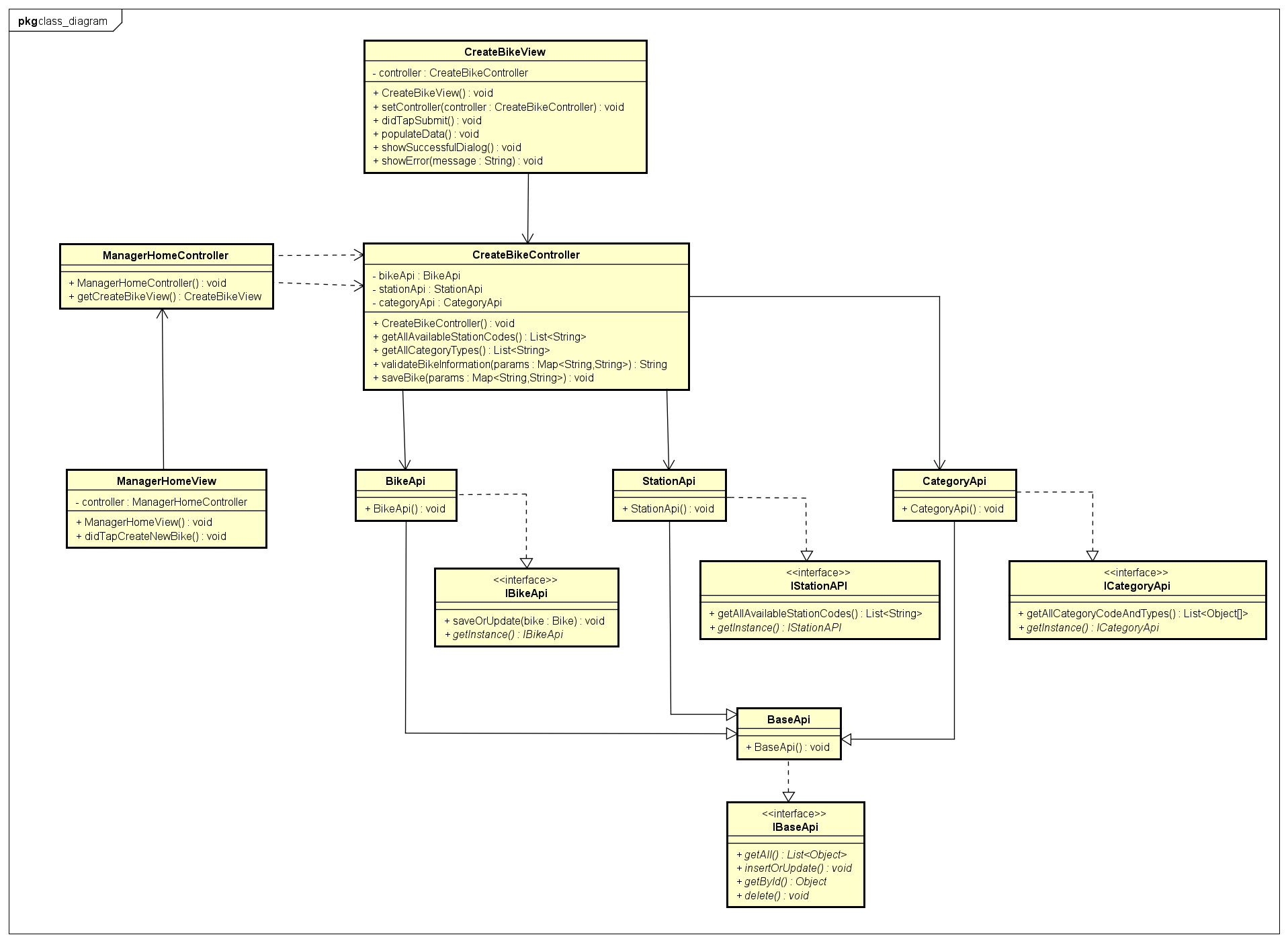
     Description automatically generated

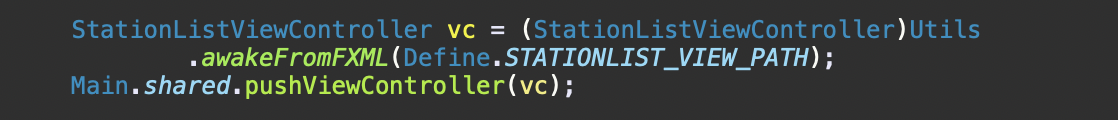
1. **Apply SOLID principle and patterns**
   1. ***SOLID principle:***
      * Single-responsibility principle: Each class has its own responsibility, such as: BikeApi class is responsible for connecting to database and manipulate CRUD operation about bike; CreateBikeView class is responsible for user interface of Create Bike screen and connect human user with our application;…
      * Open-Closed principle: We have BaseApi class that is already implemented basic CRUD operations. Therefore, BikeApi, StationApi,… classes can inherit and implement some new operation/method. It is opened for other API classes to extend from and close for BaseApi to be modified  
          
        
      * Liskov substitution principle: For example, every time we use BaseApi to get bike’s information, it could be substituted by BikeApi, which does the same functionality
      * Interface segregation principle: We divide subsystems into many smaller interfaces, each one will be responsible for each purpose. Such as IStationApi interface is responsible for station’s information manipulation.  
          
        Text

        Description automatically generated with medium confidence
      * Dependency inversion principle: Each subsystem classes will have an interface corresponding, such as: BikeApi class have IBikeApi interface,… Therefore, if BikeApi changes, it does not change implementation of other classes that depend on BikeApi  
          
          
          
        
   2. ***Patterns:***
      * Singleton pattern:
        + Since only one object Application exists when the app is running, we could save singleton of Application, from there, we have access for all information, such as: current stage, scene, …  
            
          Text

          Description automatically generated
        + We also apply singleton for all subsystem classes. It helps us not to initialize subsystem and connect/close to database so many times 🡺 save time, memory  
            
          Text

          Description automatically generated
2. **Detail design**
   1. Sequential diagram  
      A picture containing table

      Description automatically generated
   2. Class diagram  
      
3. **Self contribution**
   1. Build code base. That is the foundation of an application. It defines hierarchy of app: stage, scene, node, … And how screens can communicate and navigate backward, forward. From the foundation, my teammates can build their own screens synchronously and use pre-defined method/operation to communicate/navigate with others.  
      For example:  
        
      Text

      Description automatically generated  
        
      In Main class, I have pre-define “pushViewController” method to navigate to other screen. If we want to navigate to screen “Station Information”, we just need to call like:  
        
      
   2. responsible for “Create new bike information” usecase
   3. responsible for summarizing and create diagrams for group homework