Interface Specification

[TOC]

frontend-database

Interface 1 frontend::Login ==> database::Login

```
{
    "InterfaceId": {
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {"type": "string"},
    "UserName": {
        "type": "string",
        "minLength": 1
    },
    "PassWord": {
        "type": "string",
        "minLength": 1
    }
}

return with

{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "miLLength": 1
    },
    "MatchToken": {"type": "boolean"},
    "Interest": {
        "type": "string",
        "minimumLength": 1
    }
}
```

- Tip1: InterfaceId means the id of interface that we use the identify the interface index and json content. We have 33 interfaces in total so the InterfaceId will range from 1 to 33.
- Tip 2: CurrentUser is None if this is a anonymous visiting, otherwise it is the UserName.

- Tip3: "Interests" in return JSON is set to null if the user has not been set.(login for the first time)
- Description of the interface: When the user logs in, this interface is called. On the front end, the user needs to input the Username and Password, which are then sent to the database. The database performs verification, and the result of the verification is indicated by the returned match_token.

Interface 2 frontend::Location ==> database::Location

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "CurrentLocation": {
        "type": "object::Location",
        "properties": "Illustrated in appendix"
    }
}
```

- Description of the interface: When the user needs data related to location information, such as viewing a map, they need to provide the CurrentLocation to the database.

Interface 3 frontend::Store ==> database::Store

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "StoreName": {
        "type": "string",
        "minLength": 1
    }
}

return with

{
    "InterfaceId":{
```

```
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "StoreList": {
        "type": "array",
        "items": {
            "type": "object::Store",
            "properties": "Illustrated in appendix"
        },
        "minItems": 0,
        "uniqueItems": true
    }
}
```

- Tip1: This provides the function of querying the store information given the store name.
- Description of the interface: When a user wants to inquire about store information, they provide the StoreName to the database. Since the StoreName provided by the user might be incomplete or there may be stores with the same name, the database should return a list of stores that meet the criteria.

Interface 4 frontend::HuntedStore ==> database::HuntedStore

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "HuntedStoreIdList": {
        "type": "array",
        "items": {
            "type": "HistoryVisit",
            "properties": {
                "StoreId": {
                    "type": "integer",
                    "minimum": 0
                },
                "VisitTime": {
                    "type": "data-time"
                }
            }
        },
        "minItems": 0,
```

```
        "uniqueItems": true
    }
}
```

- Tips1: HuntedList is like a "Browsing history".
- Description of the interface: When a user browses new stores, they need to provide the HuntedStoreIdList to the database for updating the content stored in the database.

Interface 5 frontend::HuntedStore==>database::HuntedStore

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    }
}


return with

{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "HuntedStoreIdList": {
        "type": "array",
        "items": {
            "type": "HistoryVisit",
            "properties": {
                "StoreId": {
                    "type": "integer",
                    "minimum": 0
                },
                "VisitTime": {
                    "type": "data-time"
                }
            }
        },
        "minItems": 0,
```

```
        "uniqueItems": true
    }
}
```

- Tips1: This is for displaying the HuntedList.
- Tips2: This interface should be called by front end and then be responded by database.
- Description of the interface: When a user needs to view their browsing history, the database provides information about the HuntedStoreIdList to the front end.

Interface 6 frontend::Item ==> database:Item

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "ItemName": {
        "type": "string",
        "minLength": 1
    }
}

return with

{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "ItemList": {
        "type": "array",
        "items": {
            "type": "Item",
            "properties": "Illustrated in appendix"
        },
        "minItems": 0,
        "uniqueItems": true
    }
}
```

- Tips1: This provides the function of querying the item information given the item name.
- Description of the interface: When a user wants to inquire about item information, they provide the ItemName to the database. Since the ItemName provided by the user might be incomplete or there may be items with the same name, the database should return a list of items that meet the criteria.

Interface 7 frontend::Map ==> database::Map

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "MyLocation": {
        "type": "object::Location",
        "properties": "Illustrated in appendix"
    },
    "RequestType": {
        "type":"integer",
        "minimum": 1,
        "maximum": 2
    },
    "BSSIDList": {
        "type": "array",
        "items": {
            "type":"string",
            "minimum": 17,
            "maximum": 17,
        },
        "minItems": 0,
        "uniqueItems": true
    },
}

return with

{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "StoreList": {
        "type": "array",
```

```
        "items": {
            "type": "object::Store",
            "properties": "Illustrated in appendix"
        },
        "minItems": 0,
        "uniqueItems": true
    }
}
```

- Description of the interface: RequestType=1 means this a request searching for nearby stores provided with "MyLocation". RequestType=2 means this is a request for a list recommended stores. The user provides the RequestType and MyLocation to the database to obtain the needed StoreList.
- BSSID is the Basic Service Set IDentifier, used to identify the owner (typically a shop) of the the WIFI. This can help identifying which shop the user is close to.

Interface 8 frontend::Feedback2Store ==> database::Feedback2Store

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "Feedback":{
        "type": "object::Feedback2Store",
        "properties": "Illustrated in appendix"
    }
}
```

- Tip1 : Feedback is a rating that evaluate the recommendation results.
- Tip2 : "rating" should be a positive integer from 1 to 10.
- Description of the interface: Users evaluate the store recommendations provided by the recommendation system, specifically including comments and ratings, which are transmitted to the database.

Interface 9 frontend::Feedback2Item ==> database::Feedback2Item

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
```

```
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "Feedback":{
        "type": "object::Feedback2Item",
        "properties": "Illustrated in appendix"
    }
}
```

- Description of the interface: Similar to Interface8. Users evaluate the recommended item results from the recommendation system, specifically including comments and ratings, which are transmitted to the database.

Interface 10 frontend::Customer ==> database:: Customer

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "UserName": {
        "type": "string",
        "minLength": 1
    },
    "PassWord": {
        "type": "string",
        "minLength": 1
    },
    "Birthday": {
        "type": "date"
    },
    "Interests": {
        "items": "string",
        "minimumlength": 1
    }
}
```

- Tip1: This is used to update customer information stored in the database.
- Tip2: Interests are users' preference used recommendation.
- Description of the interface: When a user wishes to update their information, they provide the database with new details such as Birthday, Interests, and CurrentLocation. It should be noted that for security reasons, before updating the information, the user needs to re-enter their Username and Password for verification.

Interface 11 frontend:: Customer==>database::Customer

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    }
}

return with

{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "UserName": {
        "type": "string",
        "minLength": 1
    },
    "PassWord": {
        "type": "string",
        "minLength": 1
    },
    "Birthday": {
        "type": "date"
    },
    "Interests": {
        "items": "string",
        "minimumlength": 1
    }
}
```

- Tip1: This is used to show the customer information on the screen. Password should be set to "None".
- Description of the interface: The database provides the current user with relevant information about other users, without the need to enter a password.

Interface 12 frontend::Registration ==> database::Registration

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "UserName": {
        "type": "string",
        "minLength": 1
    },
    "PassWord": {
        "type": "string",
        "minLength": 1
    }
}

return with
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "successful_token": {"type": "boolean"}
}
```

- Tip1: this is used for registration and to check whether the UserName is unique.
- Description of the interface: Users register by entering their Username and Password on the front end, which are then transmitted to the database. The database checks if the same Username already exists and returns a successful_token to the front end to indicate whether the registration was successful or not.

---

algorithm-database

Interface 13 database::Store ==> algorithm::Store

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
```

```
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "Stores": {
        "type": "array",
        "items": {
            "type": "object::Store",
            "preperties": "Illustrated in appendix"
        },
        "minItems": 1,
        "uniqueItems": true
    },
    "Location": {
        "type": "object::Location",
        "properties": "Illustrated in appendix"
    }
}
```

- Tip1: this one is used for generating recommendation user for specific user
- Description of the interface: database delivers stores based on the location of specific user. for instance, when we gonna recommended stores for user A, database should deliver all the stores nearby user A. the threshold might be 1km or 2km or other specific distance. And recommendation system use the data provided by this interface to generate recommendation result.

Interface 14 database::Customer ==> algorithm::user

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "UserId": {
        "type": "integer",
        "minimum": 0
    },
    "UserData": {
        "type": "object::Customer",
        "preperties": "Illustrated in appendix"
    }
}
```

- Tip1: this one is the specific user who calls recommendation service
- Tip2: customer should conclude data such as his preference and history review.
- Description of the interface: recommendation system needs the preference of the user to generate recommendation.

Interface 15 database::Item ==> algorithm::recommendation algorithm

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "ItemData": {
        "type": "array",
        "items": {
            "type": "object::Item",
            "preperties": "Illustrated in appendix"
        },
        "minItems": 1,
        "uniqueItems": true
    }
}
```

- Tip1: actually this interface is for extension functions in the future.

Interface 16 algorithm::recommendation ==> database:: algorithm

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "RecommendedStores": {
        "type": "array",
        "items": {
            "type": "object::Store",
            "preperties": "Illustrated in appendix"
```

```
        },
        "minItems": 1,
        "uniqueItems": true
    }
}
```

- Tip1: this one is to deliver recommendation back to database
- Description of the interface: recommendation in this interface is considered to be an array of stores. the size of this array is concrete. when there is not enough interested store around. the recommendation might recommend something not in user's preference as exploration.

Interface 17 database:: algorithm ==> algorithm::recommendation

```
{
    "InterfaceId":{
        "type":"integer",
        "minimum": 1,
        "maximum": 33
    },
    "CurrentUser": {
        "type": "string",
        "minLength": 1
    },
    "Feedback":{
        "type": "object::Feedback2Store",
        "properties": "Illustrated in appendix"
    }
}
```

- Tip1: this one is about rating the recommendation
- description of the interface: database deliver the feedback which is about rating the recommendation. and algorithm might adjust its recommendation strategy based on this feedback.

---

web-database

- **Interface 18 web::Registration ==> database::Registration (Register)**

```
{
   "UserName": {
        "type": "string",
        "minLength": 1
    },
    "UserPassword": {
        "type": "string",
        "minLength": 1,
        "maxLength": 6
```

```
    }
}

return with

{
    "MatchToken": {"type": "boolean"}
}
```

- Description of the interface: This occurs when a new store owner attempts to register its account, which returns an indicator asserting whether the registration is successful.
- Tip1: At the database end, if an existing store owner account holds the same username, the registration should be considered unsuccessful.

Interface 19 web::Login ==> database::Login (Login)

```
{
    "UserName": {
        "type": "string",
        "minLength": 1
    },
    "UserPassword": {
        "type": "string",
        "minLength": 1,
        "maxLength": 6
    }
}

return with

{
    "MatchToken": {"type": "boolean"}
}
```

- Description of the interface: This takes place when a store owner or the administrator attempts to login to their accounts, which returns an indicator asserting whether the login process is successful.
- Tip1: At the database end, if the username and the password match a registered account, the login should be considered successful.
- Tip2: Currently the username of the administrator account is "admin", so the role of the user logged-in can be determined by the username.

Interface 20 web::Management ==> database::Administrator (Check userData)

```
{
    [empty]
}
```

```
return with

{
    "UserData" : {
        "type": "array",
        "items": {
            "UserId":{
                "type":"integer",
                "minimum": 0,
            },
            "UserName": {
                "type": "string",
                "minLength": 1
            },
            "UserPassword": {
                "type": "string",
                "minLength": 1
            },
            "Birthday": {
                "type": "date"("string")
            },
            "Interests": {
                "items": "string",
                "minimumlength": 1
            },
            "Email": {
                "type": "string"
            },
        },
        "minItems": 0
    }
}
```

- Description of the interface: This procedure describes when the administrator check the user data. When returning password, database just need to return a random string.

Interface 21 web::Management ==> database::Administrator (Update userData)

```
{
    "UserId":{
        "type":"integer",
        "minimum": 0,
    },
    "UserName": {
        "type": "string",
        "minLength": 1
    },
    "UserPassword": {
        "type": "string",
        "minLength": 1
```

```
    },
    "Birthday": {
        "type": "date"("string")
    },
    "Interests": {
        "items": "string",
        "minimumlength": 1
    },
    "Email": {
        "type": "string"
    }
}

return with

{
    [empty]
}
```

- Description of the interface: This procedure describes when the administrator update the user data. Since UserId is the primary key of the UserData, **if database receive a request with existed UserId, that means an update to this user; if database receive a new UserId, that means adding a new user.**

Interface 22 web::Management ==> database::Administrator (Delete userData)

```
{
    "UserId":{
        "type":"integer",
        "minimum": 0,
    }
}

return with

{
    [empty]
}
```

- Description of the interface: This procedure describes when the administrator delete a user's data. Given the database an UserId to decide which user to delete.

Interface 23 web::Management ==> database::Administrator (Check storeData)

```
{
    [empty]
}
```

```
return with

{
    "StoreData" : {
        "type": "array",
        "items": {
            "UserId":{
                "type":"integer",
                "minimum": 0,
            },
            "UserName": {
                "type": "string",
                "minLength": 1
            },
            "UserPassword": {
                "type": "string",
                "minLength": 1
            },
            "StoreName": {
                "type": "string",
                "minLength": 1
            },
            "StoreLocation": {
                "type": "string"
            },
            "AvgRate": {
                "type": "float"
            }
        },
        "minItems": 0
    }
}
```

- Description of the interface: This procedure describes when the administrator check the store owner data. When returning password, database just need to return a random string.

Interface 24 web::Management ==> database::Administrator (Update storeData)

```
{
    "UserId":{
        "type":"integer",
        "minimum": 0,
    },
    "UserName": {
        "type": "string",
        "minLength": 1
    },
    "UserPassword": {
        "type": "string",
        "minLength": 1
    },
```

```
    "StoreName": {
        "type": "string",
        "minLength": 1
    },
    "StoreLocation": {
        "type": "string"
    }
}

return with

{
    [empty]
}
```

- Description of the interface: This procedure describes when the administrator update the store owner data. Since UserId is the primary key of the StoreData (**NOTE: STORE OWNER ALSO OWNS ITS USERNAE AND PASSWORD SINCE IT NEED TO LOG IN THE SYSTEM**), **if database receive a request with existed UserId, that means an update to this store owner; if database receive a new UserId, that means adding a new store owner.**

Interface 25 web::Management ==> database::Administrator (Delete storeData)

```
{
    "UserId":{
        "type":"integer",
        "minimum": 0,
    }
}

return with

{
    [empty]
}
```

- Description of the interface: This procedure describes when the administrator delete a user's data. Given the database an UserId to decide which store owner to delete.

Interface 26 web::Analytics ==> database::Analytics (Check Analysis)

```
{
    [empty]
}

return with

{
```

```
    "SelectedComments":{
        "type": "array",
        "items": {
            "Comments":{
                "type":"string",
            },
            "Rating": {
                "type": "integer",
                "minimum": 1,
                "maximum": 10
            }
        },
        "minItems": 1
    },
    "OverallAdvices": {
        "type": "array",
        "items": {
            "type": "string"
        },
        "minItems": 1
    }
}
```

- Description of the interface: This describes the analytical data that is obtained by the administrator: "SelectedComments" is a list consists of all the comment that have a lower rating of 3, containing its comments and ratings; "OverallAdvices" refers to a string that generated by algorithms, to be specific, the advice or advices provided by AI for systematic refinement.

Interface 27 web::StoreOwner ==> database::Store (Check Store Info)

```
{
    "UserName": {
        "type": "string",
        "minLength": 1
    }
}

return with

{
    "StoreName": {
        "type": "string",
        "minLength": 1
    },
    "StoreLocation": {
        "type": "string",
        "minLength": 1
    },
    "AvgRate": {
        "type": "float",
        "minimum": 0,
```

```
        "maximum": 10
    },
    "Feedback": {
        "type": "array",
        "items": {
            "Comments":{
                "type":"string",
            },
            "Rating": {
                "type": "integer",
                "minimum": 1,
                "maximum": 10
            }
        },
    }
}
```

Description of the interface: This occurs when a store owner attempts to checking the statistics of the store. Giving the database the **"UserName"** of a store owner, it need to return us with the information o its store (1 store owner owns 1 store). "StoreName" refers to the name of the store, "StoreLocation" is a string consists of a pair o integer separated by comma (For example "100,50"). "AvgRate" refers to the average rating of all feedbacks happening in a specific store. "Feedback" refers to the feedbacks happening after each single purchase.

Interface 28 web::StoreInfo ==> database::Store (Update Store Info)

```
{
    "UserName": {
        "type": "string",
        "minLength": 1
    },
    "StoreName": {
        "type": "string",
        "minLength": 1
    },
    "StoreLocation": {
        "type": "string",
        "minLength": 1
    }
}

return with

{
    [empty]
}
```

- Description of the interface: This procedure occurs when a store owner wants to update its store info, specifically, name and location of the store. **Given the UserName of the store owner, the**

database should update the StoreName and StoreLocation of this store owner's store.

Interface 29 web::Item ==> database::Item (Check ItemData)

```
{
    "UserName": {
        "type": "string",
        "minLength": 1
    }
}

return with

{
    "ItemData" : {
        "type": "array",
        "items": {
            "ItemId":{
                "type":"integer",
                "minimum": 0,
            },
            "ItemName": {
                "type": "string",
                "minLength": 1
            },
            "ItemPrice": {
                "type": "integer",
                "minimum": 1
            },
            "ItemDescription": {
                "type": "string",
            },
            "ItemImage": {
                "type": "base64"("string")
        },
        "minItems": 0
    }
}
```

- Description of the interface: This procedure occurs when a store owner wants to access his items' information. Given the "UserName" of the store owner, the database should return all the item that belongs to this store owner.

Interface 30 web::Item ==> database::Item (Update ItemData)

```
{
    "UserName": {
        "type": "string",
        "minLength": 1
```

```
    },
    "ItemId":{
        "type":"integer",
        "minimum": 0,
    },
    "ItemName": {
        "type": "string",
        "minLength": 1
    },
    "ItemPrice": {
        "type": "integer",
        "minimum": 1
    },
    "ItemDescription": {
        "type": "string",
    },
    "ItemImage": {
        "type": "base64"("string")
    }
}

return with

{
    [empty]
}
```

- Description of the interface: This procedure describes when the administrator or store owner update the item data. Since ItemId is the primary key of the ItemData, **if database receive a request with existed ItemId, that means an update to this item; if database receive a new ItemId, that means adding a new item.** Also provide the UserName, so if it is a add operation, database will know who owns this ietm.

Interface 31 web::Item ==> database::Item (Delete ItemData)

```
{
    "ItemId":{
        "type":"integer",
        "minimum": 0,
    }
}

return with

{
    [empty]
}
```

- Description of the interface: This procedure describes when the administrator or store owner delete a item's data. Given the database an ItemId to decide which item to delete.

Appendix:

object::Location

```
{
    "latitude": {
        "type": "number"
    },
    "longitude":{
        "type": "number"
    },
    "country" : {
        "type": "string",
        "minLength": 0
    },
    "state" : {
        "type": "string",
        "minLength": 0
    },
    "city": {
        "type": "string",
        "minLength": 0
    },
    "street": {
        "type": "string",
        "minLength": 0
    },
    "number": {
        "type": "string",
        "minLength": 0
    },
    "floor": {
        "type": "string",
        "minLength": 0
    },
    "zipcode": {
        "type": "string",
        "minLength": 0
    }
}
```

object::Item

```
{
    "ItemId": {
        "type": "integer",
```

```
            "minLength": 0
        },
        "ItemName": {
            "type": "string",
            "minLength": 1
        },
        "ItemPrice": {
            "type": "number",
            "minimum": 0
        },
        "ItemDescription": {
            "type": "string",
            "minLength": 0
        },
        "ItemImage": {
            "type": "base64",
            "minLength": 0
        },
        "ItemStoreId": {
            "type": "integer",
            "minLength": 0
        },
        "ItemStoreName": {
            "type": "string",
            "minLength": 1
        },
        "customerVisits": {
            "type": "integer"
        }
    }
```

- Tip1: Same items in different stores have different ItemId, which means ItemId is unique.

object::Store

```
    {
        "storeId": {
            "type": "integer",
            "minLength": 0
        },
        "storeName": {
            "type": "string",
            "minLength": 1
        },
        "location": {
            "type": "object:Location",
            "preperties": "Illustrated in appendix"
        },
        "items": {
            "type": "array",
            "items": {
```

```
            "type": "object::Item",
            "preperties": "Illustrated in appendix"
        },
        "minItems": 1,
        "uniqueItems": true
    },
    "StoreDescription": {
        "type": "string",
        "minLength": 0
    }
}
```

object::Feedback2Store

```
{
    "StoreId": {
        "type": "integer",
    },
    "comment": {
        "type": "string",
        "minLength": 0
    },
    "rating": {
        "type":"int",
        "minimum": 1,
        "maximum": 10
    },
    "UserName": {
        "type": "string",
        "minLength": 1
    }
}
```

- Description: This is used to give feedback to the recommendation results (a store) and helps optimize recommendation algorithm.

object::Feedback2Item

```
{
    "ItemId": {
        "type": "integer",
    },
    "comment": {
        "type": "string",
        "minLength": 0
    },
    "rating": {
        "type":"int",
```

```
            "minimum": 1,
            "maximum": 10
        },
        "UserName": {
            "type": "string",
            "minLength": 1
        }
    }
```

- Description: This is used to give feedback to the purchased item and helps stores to offer better service.

object::User

```
{
    "userId": {
        "type": "integer",
        "minimum": 0,
    },
    "UserEmail": {
        "type": "email"
    },
    "UserName": {
        "type": "string",
        "minLength": 1
    },
    "HashedPassword": {
        "type": "string",
        "minLength": 1
    },
    "Interests": {
        "items": "string",
        "minimumlength": 1
    },
    "HuntedStoreIdList": {
        "type": "array",
        "items": {
            "type": "HistoryVisit",
            "properties": {
                "StoreId": {
                    "type": "integer",
                    "minimum": 0
                },
                "VisitTime": {
                    "type": "data-time"
                }
            }
        },
        "minItems": 0,
        "uniqueItems": true
    }
}
```

---

update from v.6.6 to v.7.0:

1. In Interface 7, we add a BSSIDList. BBSID is the Basic Service Set IDentifier, used to identify the owner (typically a shop) of the the WIFI. This can help identifying which shop the user is close to.