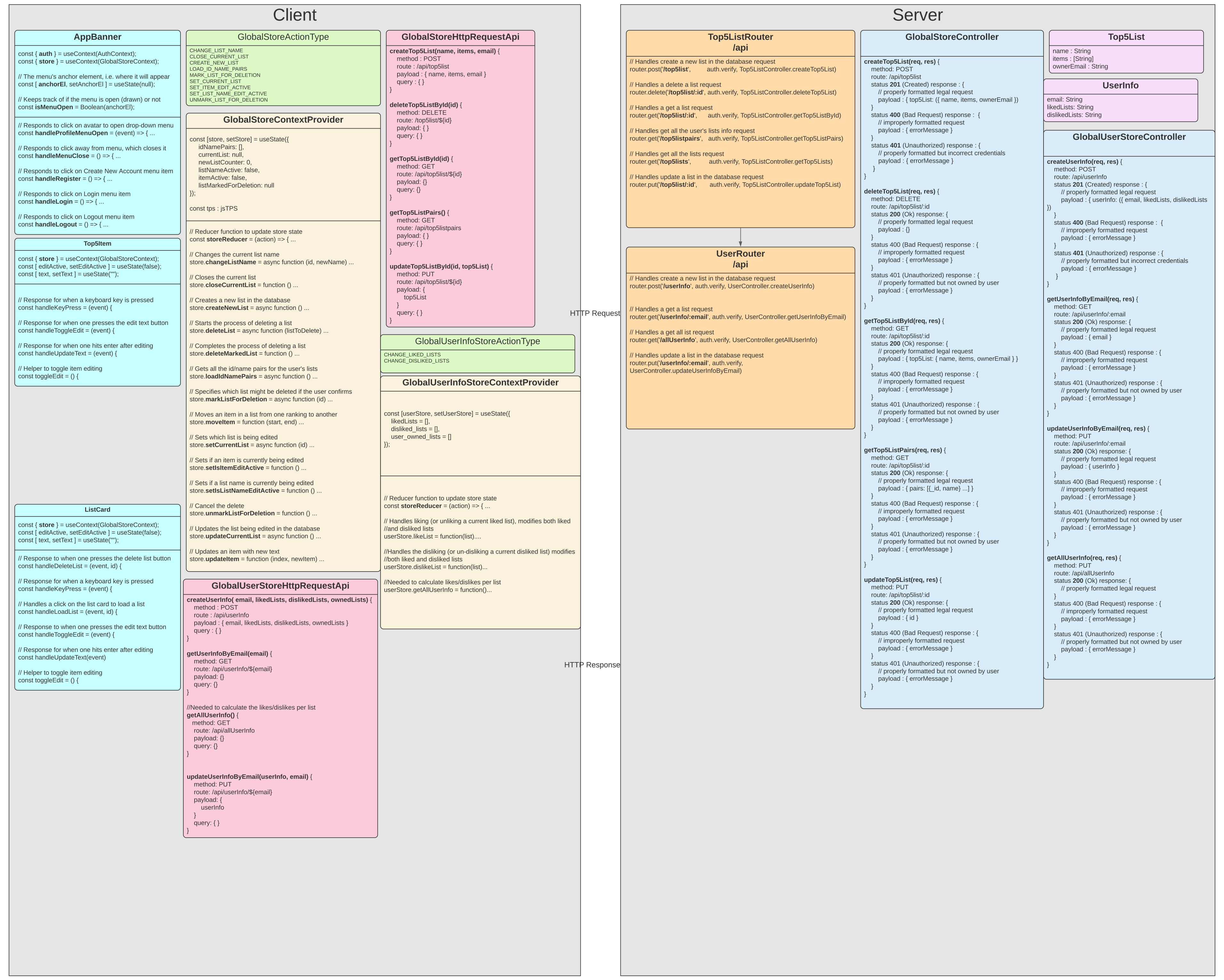


Client

AppBanner
```
const { auth } = useContext(AuthContext);
const { store } = useContext(GlobalStoreContext);

// The menu's anchor element, i.e. where it will appear
const [ anchorEl, setAnchorEl ] = useState(null);

// Keeps track of if the menu is open (drawn) or not
const isMenuOpen = Boolean(anchorEl);

// Responds to click on avatar to open drop-down menu
const handleProfileMenuOpen = (event) => { ...

// Responds to click away from menu, which closes it
const handleMenuClose = () => { ...

// Responds to click on Create New Account menu item
const handleRegister = () => { ...

// Responds to click on Login menu item
const handleLogin = () => { ...

// Responds to click on Logout menu item
const handleLogout = () => { ...
```

Top5Item
```
const { store } = useContext(GlobalStoreContext);
const [ editActive, setEditActive ] = useState(false);
const [ text, setText ] = useState("");

// Response for when a keyboard key is pressed
const handleKeyPress = (event) {

// Response for when one presses the edit text button
const handleToggleEdit = (event) {

// Response for when one hits enter after editing
const handleUpdateText = (event) {

// Helper to toggle item editing
const toggleEdit = () {
```

ListCard
```
const { store } = useContext(GlobalStoreContext);
const [ editActive, setEditActive ] = useState(false);
const [ text, setText ] = useState("");

// Response to when one presses the delete list button
const handleDeleteList = (event, id) {

// Response for when a keyboard key is pressed
const handleKeyPress = (event) {

// Handles a click on the list card to load a list
const handleLoadList = (event, id) {

// Response to when one presses the edit text button
const handleToggleEdit = (event) {

// Response for when one hits enter after editing
const handleUpdateText(event)

// Helper to toggle item editing
const toggleEdit = () {
```

GlobalStoreActionType
```
CHANGE_LIST_NAME
CLOSE_CURRENT_LIST
CREATE_NEW_LIST
LOAD_ID_NAME_PAIRS
MARK_LIST_FOR_DELETION
SET_CURRENT_LIST
SET_ITEM_EDIT_ACTIVE
SET_LIST_NAME_EDIT_ACTIVE
UNMARK_LIST_FOR_DELETION
```

GlobalStoreContextProvider
```
const [store, setStore] = useState({
    idNamePairs: [],
    currentList: null,
    newListCounter: 0,
    listNameActive: false,
    itemActive: false,
    listMarkedForDeletion: null
});

const tps : jsTPS

// Reducer function to update store state
const storeReducer = (action) => { ...

// Changes the current list name
store.changeListName = async function (id, newName) ...

// Closes the current list
store.closeCurrentList = function () ...

// Creates a new list in the database
store.createNewList = async function () ...

// Starts the process of deleting a list
store.deleteList = async function (listToDelete) ...

// Completes the process of deleting a list
store.deleteMarkedList = function () ...

// Gets all the id/name pairs for the user's lists
store.loadIdNamePairs = async function () ...

// Specifies which list might be deleted if the user confirms
store.markListForDeletion = async function (id) ...

// Moves an item in a list from one ranking to another
store.moveItem = function (start, end) ...

// Sets which list is being edited
store.setCurrentList = async function (id) ...

// Sets if an item is currently being edited
store.setIsItemEditActive = function () ...

// Sets if a list name is currently being edited
store.setIsListNameEditActive = function () ...

// Cancel the delete
store.unmarkListForDeletion = function () ...

// Updates the list being edited in the database
store.updateCurrentList = async function () ...

// Updates an item with new text
store.updateItem = function (index, newItem) ...
```

GlobalUserStoreHttpRequestApi
```
createUserInfo( email, likedLists, dislikedLists, ownedLists) {
    method : POST
    route : /api/userInfo
    payload : { email, likedLists, dislikedLists, ownedLists }
    query : { }
}

getUserInfoByEmail(email) {
    method: GET
    route: /api/userInfo/${email}
    payload: {}
    query: {}
}

//Needed to calculate the likes/dislikes per list
getAllUserInfo() {
   method: GET
    route: /api/allUserInfo
    payload: {}
    query: {}
}


updateUserInfoByEmail(userInfo, email) {
    method: PUT
    route: /api/userInfo/${email}
    payload: {
        userInfo
    }
    query: { }
}
```

GlobalStoreHttpRequestApi
```
createTop5List(name, items, email) {
    method : POST
    route : /api/top5list
    payload : { name, items, email }
    query : { }
}

deleteTop5ListById(id) {
    method: DELETE
    route: /top5list/${id}
    payload: { }
    query: { }
}

getTop5ListById(id) {
    method: GET
    route: /api/top5list/${id}
    payload: {}
    query: {}
}

getTop5ListPairs() {
    method: GET
    route: /api/top5listpairs
    payload: { }
    query: { }
}

updateTop5ListById(id, top5List) {
    method: PUT
    route: /api/top5list/${id}
    payload: {
        top5List
    }
    query: { }
}
```

GlobalUserInfoStoreActionType
```
CHANGE_LIKED_LISTS
CHANGE_DISLIKED_LISTS
```

GlobalUserInfoStoreContextProvider
```
const [userStore, setUserStore] = useState({
   likedLists = [],
   disliked_lists = [],
   user_owned_lists = []
});

// Reducer function to update store state
const storeReducer = (action) => { ...

// Handles liking (or unliking a current liked list), modifies both liked
//and disliked lists
userStore.likeList = function(list)....

//Handles the disliking (or un-disliking a current disliked list) modifies
//both liked and disliked lists
userStore.dislikeList = function(list)...

//Needed to calculate likes/dislikes per list
userStore.getAllUserInfo = function()...
```

HTTP Request

HTTP Response

Server

Top5ListRouter
/api
```
// Handles create a new list in the database request
router.post('/top5list',        auth.verify, Top5ListController.createTop5List)

// Handles a delete a list request
router.delete('/top5list/:id', auth.verify, Top5ListController.deleteTop5List)

// Handles a get a list request
router.get('/top5list/:id',      auth.verify, Top5ListController.getTop5ListById)

// Handles get all the user's lists info request
router.get('/top5listpairs',   auth.verify, Top5ListController.getTop5ListPairs)

// Handles get all the lists request
router.get('/top5lists',          auth.verify, Top5ListController.getTop5Lists)

// Handles update a list in the database request
router.put('/top5list/:id',      auth.verify, Top5ListController.updateTop5List)
```

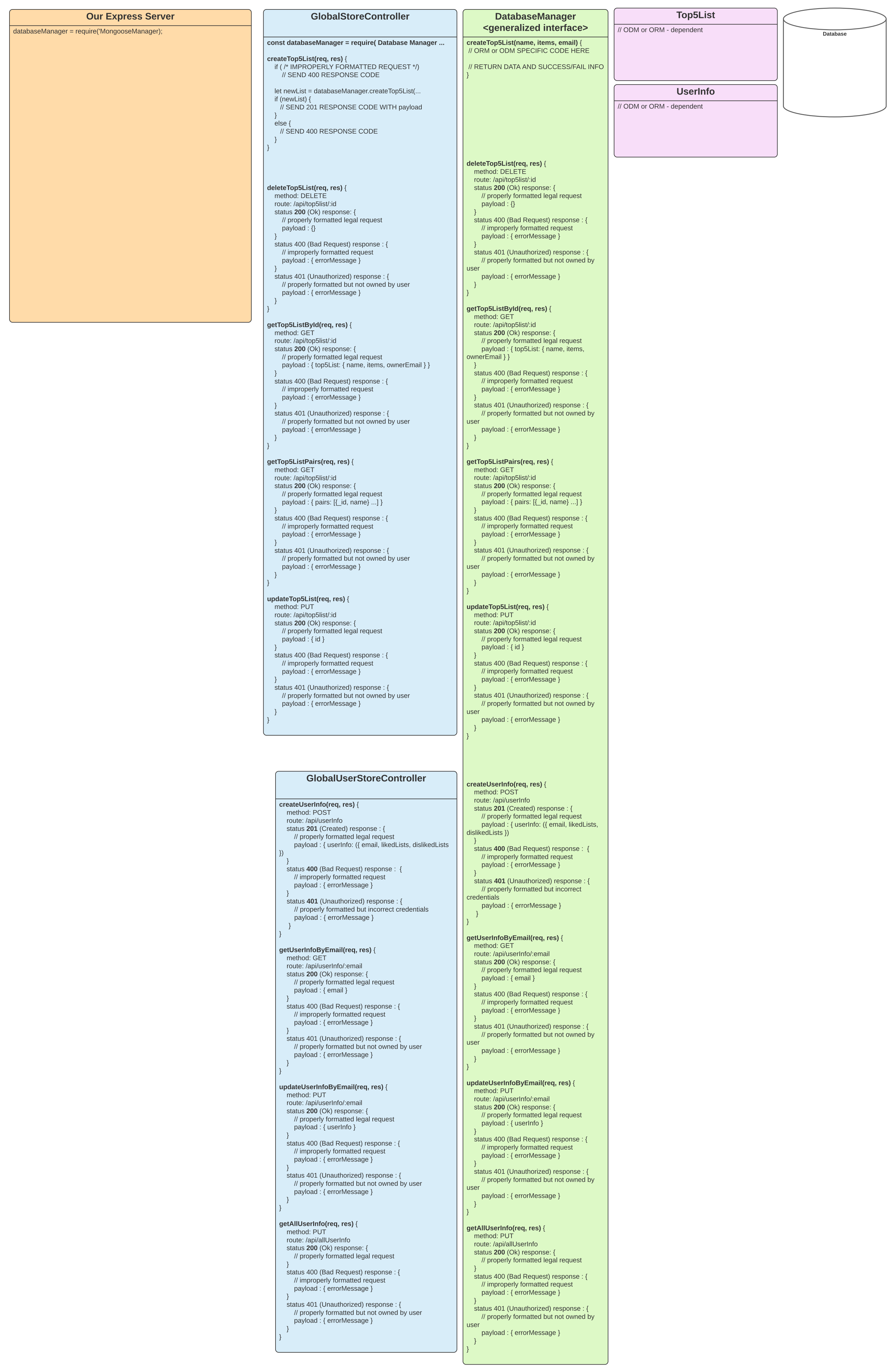
UserRouter
/api
```
// Handles create a new list in the database request
router.post('/userInfo', auth.verify, UserController.createUserInfo)


// Handles a get a list request
router.get('/userInfo/:email', auth.verify, UserController.getUserInfoByEmail)

// Handles a get all ist request
router.get('/allUserInfo', auth.verify, UserController.getAllUserInfo)

// Handles update a list in the database request
router.put('/userInfo/:email', auth.verify,
UserController.updateUserInfoByEmail)
```

GlobalStoreController
```
createTop5List(req, res) {
    method: POST
    route: /api/top5list
    status 201 (Created) response : {
        // properly formatted legal request
        payload : { top5List: ({ name, items, ownerEmail })
    }
    status 400 (Bad Request) response :  {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but incorrect credentials
        payload : { errorMessage }
     }
}

deleteTop5List(req, res) {
    method: DELETE
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : {}
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
    }
}

getTop5ListById(req, res) {
    method: GET
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : { top5List: { name, items, ownerEmail } }
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
    }
}

getTop5ListPairs(req, res) {
    method: GET
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : { pairs: [{_id, name} ...] }
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
    }
}

updateTop5List(req, res) {
    method: PUT
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : { id }
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
    }
}
```

Top5List
```
name : String
items : [String]
ownerEmail : String
```

UserInfo
```
email: String
likedLists: String
dislikedLists: String
```

GlobalUserStoreController
```
createUserInfo(req, res) {
    method: POST
    route: /api/userInfo
    status 201 (Created) response : {
        // properly formatted legal request
        payload : { userInfo: ({ email, likedLists, dislikedLists
})
    }
    status 400 (Bad Request) response :  {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but incorrect credentials
        payload : { errorMessage }
     }
}

getUserInfoByEmail(req, res) {
    method: GET
    route: /api/userInfo/:email
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : { email }
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
    }
}

updateUserInfoByEmail(req, res) {
    method: PUT
    route: /api/userInfo/:email
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : { userInfo }
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
    }
}

getAllUserInfo(req, res) {
    method: PUT
    route: /api/allUserInfo
    status 200 (Ok) response: {
        // properly formatted legal request
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
    }
}
```



Our Express Server
```
databaseManager = require('MongooseManager);
```

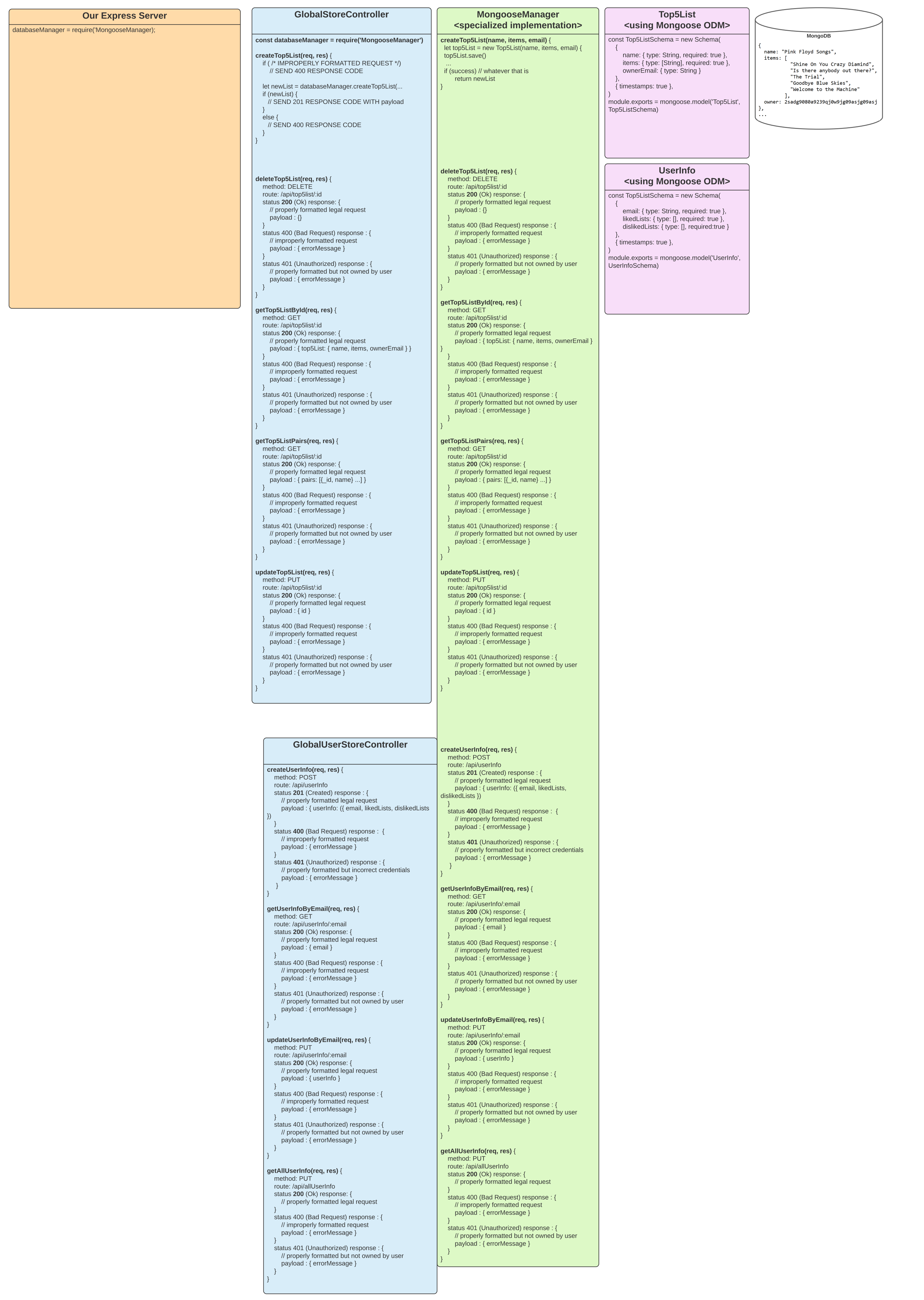
GlobalStoreController
```
const databaseManager = require( Database Manager ...

createTop5List(req, res) {
    if ( /* IMPROPERLY FORMATTED REQUEST */)
        // SEND 400 RESPONSE CODE

    let newList = databaseManager.createTop5List(...
    if (newList) {
       // SEND 201 RESPONSE CODE WITH payload
    }
    else {
       // SEND 400 RESPONSE CODE
    }
}


deleteTop5List(req, res) {
    method: DELETE
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : {}
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
    }
}

getTop5ListById(req, res) {
    method: GET
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : { top5List: { name, items, ownerEmail } }
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
```

DatabaseManager
<generalized interface>
```
createTop5List(name, items, email) {
 // ORM or ODM SPECIFIC CODE HERE

 // RETURN DATA AND SUCCESS/FAIL INFO
}


deleteTop5List(req, res) {
    method: DELETE
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : {}
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by
user
        payload : { errorMessage }
    }
}

getTop5ListById(req, res) {
    method: GET
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : { top5List: { name, items,
ownerEmail } }
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
```

Top5List
```
// ODM or ORM - dependent
```

UserInfo
```
// ODM or ORM - dependent
```

GlobalUserStoreController



Our Express Server
```
databaseManager = require('MongooseManager);
```

GlobalStoreController
```
const databaseManager = require('MongooseManager')

createTop5List(req, res) {
    if ( /* IMPROPERLY FORMATTED REQUEST */)
        // SEND 400 RESPONSE CODE

    let newList = databaseManager.createTop5List(...
    if (newList) {
       // SEND 201 RESPONSE CODE WITH payload
    }
    else {
       // SEND 400 RESPONSE CODE
    }
}


deleteTop5List(req, res) {
    method: DELETE
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : {}
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
    }
}

getTop5ListById(req, res) {
    method: GET
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : { top5List: { name, items, ownerEmail } }
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
```

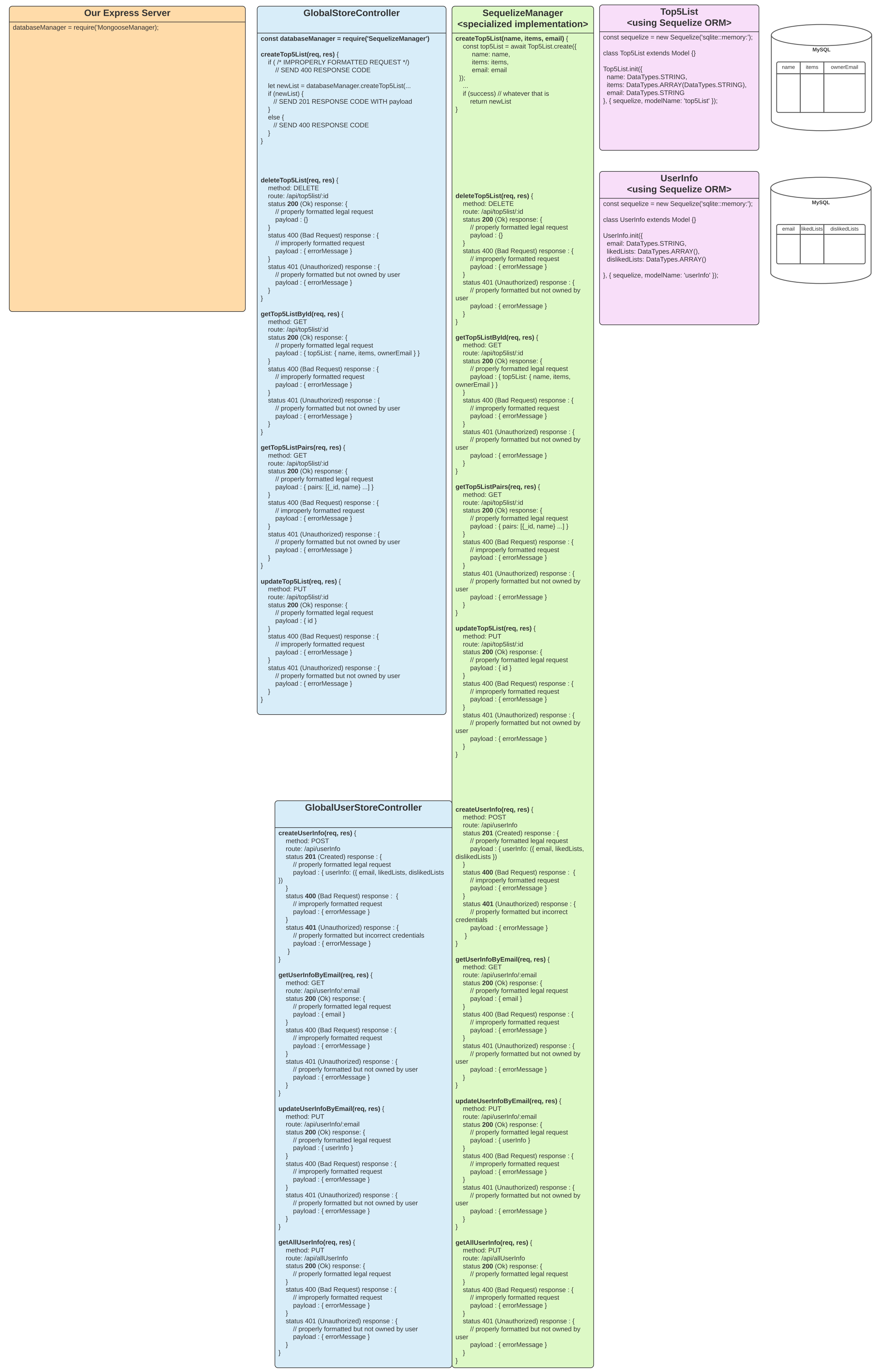
MongooseManager
<specialized implementation>
```
createTop5List(name, items, email) {
  let top5List = new Top5List(name, items, email) {
  top5List.save()
   ...
  if (success) // whatever that is
       return newList
}


deleteTop5List(req, res) {
    method: DELETE
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : {}
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
    }
}

getTop5ListById(req, res) {
    method: GET
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : { top5List: { name, items, ownerEmail }
}
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
```

Top5List
<using Mongoose ODM>
```
const Top5ListSchema = new Schema(
    {
        name: { type: String, required: true },
        items: { type: [String], required: true },
        ownerEmail: { type: String }
    },
    { timestamps: true },
)
module.exports = mongoose.model('Top5List',
Top5ListSchema)
```

UserInfo
<using Mongoose ODM>
```
const Top5ListSchema = new Schema(
    {
        email: { type: String, required: true },
        likedLists: { type: [], required: true },
        dislikedLists: { type: [], required:true }
    },
    { timestamps: true },
)
module.exports = mongoose.model('UserInfo',
UserInfoSchema)
```

```
{
  name: "Pink Floyd Songs",
  items: [
           "Shine On You Crazy Diamind",
           "Is there anybody out there?",
           "The Trial",
           "Goodbye Blue Skies",
           "Welcome to the Machine"
         ],
  owner: 2sadg9080a9239qj0w9jg09asjg09asj
},
...
```

GlobalUserStoreController



Our Express Server
```
databaseManager = require('MongooseManager);
```

GlobalStoreController
```
const databaseManager = require('SequelizeManager')

createTop5List(req, res) {
    if ( /* IMPROPERLY FORMATTED REQUEST */)
        // SEND 400 RESPONSE CODE

    let newList = databaseManager.createTop5List(...
    if (newList) {
       // SEND 201 RESPONSE CODE WITH payload
    }
    else {
       // SEND 400 RESPONSE CODE
    }
}


deleteTop5List(req, res) {
    method: DELETE
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : {}
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
    }
}

getTop5ListById(req, res) {
    method: GET
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : { top5List: { name, items, ownerEmail } }
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by user
        payload : { errorMessage }
    }
```

SequelizeManager
<specialized implementation>
```
createTop5List(name, items, email) {
   const top5List = await Top5List.create({
       name: name,
       items: items,
       email: email
  });
   ...
   if (success) // whatever that is
       return newList
}


deleteTop5List(req, res) {
    method: DELETE
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : {}
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
    status 401 (Unauthorized) response : {
        // properly formatted but not owned by
user
        payload : { errorMessage }
    }
}

getTop5ListById(req, res) {
    method: GET
    route: /api/top5list/:id
    status 200 (Ok) response: {
        // properly formatted legal request
        payload : { top5List: { name, items,
ownerEmail } }
    }
    status 400 (Bad Request) response : {
        // improperly formatted request
        payload : { errorMessage }
    }
```

Top5List
<using Sequelize ORM>
```
const sequelize = new Sequelize('sqlite::memory:');

class Top5List extends Model {}

Top5List.init({
  name: DataTypes.STRING,
  items: DataTypes.ARRAY(DataTypes.STRING),
  email: DataTypes.STRING
}, { sequelize, modelName: 'top5List' });
```

UserInfo
<using Sequelize ORM>
```
const sequelize = new Sequelize('sqlite::memory:');

class UserInfo extends Model {}

UserInfo.init({
  email: DataTypes.STRING,
  likedLists: DataTypes.ARRAY(),
  dislikedLists: DataTypes.ARRAY()

}, { sequelize, modelName: 'userInfo' });
```

GlobalUserStoreController