**Team Members:** David Pace, Andy Tran, and Lisa Gilmore-Montero

| Data Models | Schema | Description |
| --- | --- | --- |
| Cart | {  “totalQuantity”: Number,  “dollarUS”: String,  “products”: [{ type: Schema.ObjectId, ref: 'Product’' }]  } | Properties:   * **objectID automatic** * totalQuantity * dollarUS * products: contains array of Product’s objectID |
| Product | {  “title”: String,  “des”: String,  “quantity”: Number,  “price:’ Number,  “image”: String,  “qualFeatures”: { type : Array , "default" : [] },  “quanFeatures”: { type : Array , "default" : []},  “category\_id”: Number  } | Properties:   * **ObjectId automatic** * title * des * quantity * price * image * qualFeatures (note: this is an array of Strings. If we search for it and the product does not have it then it is just not there so automatically it is false) * quanFeatures (contains an array of objects that look like this: {“quanName”: String, “quanNum”: Number}) * category\_id |
| Category | {  “type”: String,  “products” : [{ type: Schema.ObjectId, ref: 'Product' }]  } | Properties   * type * **ObjectID automatic** * products: contains array of Product’s ObjectID |
| Wishlist | {  “type”: String,  “products” :[{ type: Schema.ObjectId, ref: 'Product’' }]  } | Properties   * type * **objectID automatic** * products: contains array of Product’s ObjectID |
| User | {  “name”: String,  “email”: String,  “carts”: [{ type: Schema.ObjectId, ref: 'Cart' }],  “curCart”: Number,  “wishlists”: [{ type: Schema.ObjectId, ref: 'Cart' }],  “password”: String,  “htmlElement”: String  } | Properties   * **objectID automatic** * name * email * carts: contains array of Cart’s ObjectID * curCart: ObjectID of a Cart * wishlists: contains array of Wishlist’s ObjectID * password |
| Store | {  “Categories”:  [{ type: Schema.ObjectId, ref: 'Categories' }], “products”:  [{ type: Schema.ObjectId, ref: 'Product' }],  “Customer”:  [{ type: Schema.ObjectId, ref: 'Customer' }]  } | Properties   * **objectID automatic** * Array of category document references * Array of product document references * Array of customer document references |

| Query Logic | Description |
| --- | --- |
| Selecting | We can use find() and filter by ID to get information about a specific document. |
| Inserting | We can use insert() to add new documents to collections |
| Adding to Document | We can use updateOne() and $push/$pushAll/$each to add new elements/documents to a document’s array |
| Update Document | We can use updateOne() and $set to update key value pairs within a document |
| Filtering | We can use find() with conditions to apply filters based on category, price range, qualitative features, and quantitative features |
| Searching | We can use find({ “$text”: {“$search”: String} }) when trying to search a product by name and description using createIndex()   * <https://www.mongodb.com/docs/v4.4/core/index-text/#std-label-index-feature-text> * <https://www.mongodb.com/docs/v4.4/text-search/> * <https://www.geeksforgeeks.org/mongodb-text-indexes/>   Look at table below under the Read Section |
| Sorting | We can use sort() to sort by increasing/decreasing order of “title” or “price” of Products by using createIndex()   * <https://www.mongodb.com/docs/v4.0/tutorial/sort-results-with-indexes/> * <https://www.mongodb.com/docs/manual/tutorial/sort-results-with-indexes/> * <https://www.geeksforgeeks.org/mongodb-db-collection-createindex-method/#:~:text=CreateIndex()%20Method,-Last%20Updated%20%3A%2017&text=In%20MongoDB%2C%20indexes%20are%20special,field%20specified%20in%20the%20index>.   Look at table below under the Read Section |

| CRUD | Query | Description |
| --- | --- | --- |
| For Inserts | Note: remember to use ***{"ordered": false}*** | to disable the default ordered insert |
| CreateNewProduct | collection.insert(newProduct) | Create a new product to collection of Products |
| CreateCart | collection.insert(newCart) | Create a new cart to collection of Carts |
| CreateCate | collection.insert(newCate) | Create a new category to collection of Categories |
| CreateWish | collection.insert(newWish) | Create a new category to collection of Wishlists |
| CreateUser | collection.insert(newUser) | Create a new category to collection of Users |
| CreateProdQualFeat | collection.updateOne(  { “\_id”: String },  { “$push” : { “qualFeatures”: String }  )  collection.updateOne(  { “\_id”: String },  { “$push”: {“qualFeatures”: {“$each”: [String, …]} } }  )  ^ not 100% certain for multiple values | <https://www.mongodb.com/docs/manual/reference/operator/update/push/>   * We are appending a new qualitative feature to a product |
| CreateProdQuanFeat | collection.updateOne(  { “\_id”: String },  { “$push”: {“quanFeatures”: {“quanName”: String, “quanNum”: Number} }}  )  collection.updateOne(  { “\_id”: String },  { “$push”: {“quanFeatures”: {  “$each”: [{“quanName”: String, “quanNum”:Number}, …] } }  }  ) | <https://www.mongodb.com/docs/v4.0/reference/operator/update/push/>  <https://www.mongodb.com/docs/manual/reference/operator/update/push/>  Not 100% certain for the one where you add only one new quanFeature to a product  Another option is to add multiple quan features at a time |
| CreateCartProducts | Similar to CreateProdQualFeat just change “qualFeatures” with “products” | Add product to cart |
| CreateCateProducts | Similar to CreateProdQualFeat just change “qualFeatures” with “products” | Add product to category |
| CreateWishProducts | Similar to CreateProdQualFeat just change “qualFeatures” with “products” | Add product to wishlist |
| CreateUserCarts | Similar to CreateProdQualFeat just change “qualFeatures” with “carts” | Add cart to carts |
| CreateUserWishlists | Similar to CreateProdQualFeat just change “qualFeatures” with “wishlists” | Add wishlist to wishlists |
| READ NOTE | Include ***{ “quantity”: { {“$gt”: 0} }*** | If we do not want to see the products that are no longer in stock |
| ReadAllProducts | collection.find({ “quantity”: {“$gt”: 0} }) | * to display all products for product collections for carts, wishlist, or store page * Only show products that are in stock * Can also be used to reset the filtering/sorting |
| ReadSortProducts | a) collection.createIndex({ “title”: Number, “price”: Number })  b) collection.find().sort({ “title”: Number }) | Resource Links In Previous Table  -1 = descending order  1 = ascending order   1. CreateIndex based on title or price 2. Can sort based on 1 or -1   To display all products based on ascending or descending order of the product’s title or price |
| ReadProductByName/Description | a) collection.createIndex({ “title”: “text”, “des”: “text” })  b) collection.find({ “$text”: {“$search”: String} })  **NOTE: A collection can only have one text search index, but that index can cover multiple fields** | <https://www.mongodb.com/docs/v4.4/text-search/>  <https://www.mongodb.com/docs/manual/core/index-text/>  More Resource Link in Previous Table  To display products based on its name |
| ReadCategory | collection.find({ “category\_id”: insertNumber, “quantity” : {“$gt”: 0} }) | To display all products based on category and only show products that are in stock |
| ReadUser | collection.findById(insertUserObjId) | To access specific user |
| ReadProduct | collection.findById(insertNumber) | To display product page |
| ReadQualFeature | collection.find(  {  { “quantity”: { {“$gt”: 0} },  “qualFeatures”: {“$all”: [“insertString”,... ] }  }) | To display products that have specific qualitative features |
| ReadQuanFeature | collection.find({ “$and”:  [  { “quantity”: {{“$gt”: 0}},  { “quanFeatures.quanName”: {$gte”: insertMinVal} },  { “quanFeatures.quanName”: {“$lte”: insertMaxVal} }  ]  }) | To display products based on quantitative features by providing min and max value |
| ReadPrice | collection.find({ “$and”:  [  { “quantity”: { {“$gt”: 0} },  { “price”: {$gte”: insertMinVal} },  { “price”: {“$lte”: insertMaxVal} }  ]  }) | To display products based on price |
| UpdateUser | collection.updateOne({ “\_id”: String },  { “$set:  { “name”: String,  “email”: String,  “curCart”: Number,  “Password”: String,  “htmlElement”: String }  }  }) | * Update a property of user * Choose which ones you want to include in the update |
| UpdateUserCarts | collection.updateOne(  { “\_id”: String, “carts”: Number },  { “$set”: {“carts.$”: Number }  }) | <https://www.mongodb.com/docs/manual/reference/operator/update/positional/>   * Under Examples → Update Values in an Array * \_id = userID and specific cartID we want to update * $ is the first match to the specific cartID we want to change |
| UpdateUserWishlist | collection.updateOne(  { “\_id”: String, “wishlists”: Number },  { “$set”: {“wishlists.$”: Number} }  ) | <https://www.mongodb.com/docs/manual/reference/operator/update/positional/>   * Under Examples → Update Values in an Array * \_id = userID and specific wishlistID we want to update * $ is the first match to the specific wishlistID we want to change |
| UpdateCart | collection.updateOne({ “\_id”: Number},  { “$set:  { “totalQuantity”: String, “dollarUS”: String }  }  }) | Update a specific cart’s total quantity and total amount |
| UpdateCartProducts | collection.updateOne(  { “\_id”: String, “products”: Number },  { “$set”: {“products.$”: Number} }  }) | <https://www.mongodb.com/docs/manual/reference/operator/update/positional/>   * Under Examples → Update Values in an Array * \_id = cartID and specific productID we want to update * $ is the first match to the specific productID we want to change |
| UpdateWishlist | collection.updateOne({ “\_id”: String},  { “$set: {“type”: String} }  }) | Update a specific wish’s name |
| UpdateWishlistProducts | collection.updateOne(  { “\_id”: String, “products”: Number },  { “$set”: {“products.$”: Number} }  ) | <https://www.mongodb.com/docs/manual/reference/operator/update/positional/>   * Under Examples → Update Values in an Array * \_id = cartID and specific productID we want to update * $ is the first match to the specific productID we want to change |
| UpdateCate | collection.updateOne({ “\_id”: String},  { “$set: {“type”: String} }  ) | Update a specific category’s name |
| UpdateCateProducts | collection.updateOne(  { “\_id”: String, “products”: Number},  { “$set”: {“products.$”: Number} }  ) | <https://www.mongodb.com/docs/manual/reference/operator/update/positional/>   * Under Examples → Update Values in an Array * \_id = cartID and specific productID we want to update * $ is the first match to the specific productID we want to change |
| UpdateProduct | collection.updateOne({ “\_id”: String },  { “$set:  { “title”: String,  “des”: String,  “quantity”: Number,  “price”: Number,  “image” : String,  “category\_id”: Number }  }  }) | * Update a property of user * Choose which ones you want to include in the update |
| UpdateProductQualF | collection.updateOne(  { “\_id”: String, “qualFeatures”: String },  { “$set”: {“qualFeatures.$”: String} }  ) | <https://www.mongodb.com/docs/manual/reference/operator/update/positional/>   * Under Examples → Update Values in an Array * \_id = productID and specific qualitative feature we want to update * $ is the first match to the specific qualitative feature we want to change |
| UpdateProductQuanF | collection.updateOne(  { “\_id”: String, “quanFeatures.quanName”: String },  { “$set”: {“quanFeatures.$.quanNum”: Number} }  ) | The array is an array of documents that look like this:  ***{“quanName”: String, “quanNum”:Number}***  <https://www.mongodb.com/docs/manual/reference/operator/update/positional/>   * Under Examples → Update Documents in an Array * id\_ = productID and specific quantitative feature we want to change based on name * $ is the first match to the specific quantitative feature we want to change and changing its quanNum value |
| DeleteProductQualF | collection.updateOne(  { “\_id”: String },  { “$pull”: {“qualFeatures”: String} }  )  collection.updateOne(  { “\_id”: String },  { “$pullAll”: {“qualFeatures”: [String,...]} }  ) | <https://www.mongodb.com/docs/manual/reference/operator/update/pull/>   * \_id is a specific productID and we are removing a specific qualitative feature   <https://www.mongodb.com/docs/manual/reference/operator/update/pullAll/>   * pullAll - remove all instances of the specified values from an existing array |
| DeleteProductQuanF | collection.updateOne(  { “\_id”: String },  { “$pull”: {“quanFeatures”: {“quanName”: String}} }  )  collection.updateOne(  { “\_id”: String },  { “$pull”: {“quanFeatures”: {“quanName”: { “$in”: [String, …] ]}} } ) | <https://www.mongodb.com/docs/manual/reference/operator/update/pull/>   * \_id is a specific productID and we are removing a specific Document from the array of quanFeatures based on the quanName   <https://www.tutorialspoint.com/pull-multiple-objects-from-an-array-in-mongodb>  <https://www.mongodb.com/docs/manual/reference/operator/query/in/>   * Delete multiple quantitative features at a time |
| DeleteCartProducts | Similar to DeleteProductQualF just change “qualFeatures” to “products” | Remove product from cart |
| DeleteCateProducts | Similar to DeleteProductQualF just change “qualFeatures” to “products” | Remove product from category |
| DeleteWishProducts | Similar to DeleteProductQualF just change “qualFeatures” to “products” | Remove product from wishlist |
| DeleteUserCarts | Similar to DeleteProductQualF just change “qualFeatures” to “products” | Remove cart from cart |
| DeleteUserWishlists | Similar to DeleteProductQualF just change “qualFeatures” to “products” | Remove wishlist from wishlists |
| DeleteProduct | collection.updateOne({ “\_id”: String }) | Delete product |
| DeleteCart | collection.updateOne({ “\_id”: String }) | Delete cart |
| DeleteCategory | collection.updateOne({ “\_id”: String }) | Delete category |
| DeleteWishlist | collection.updateOne({ “\_id”: String }) | Delete wishlist |
| DeleteUser | collection.updateOne({ “\_id”: String }) | Delete user |

**Tools & Libs**

* We will be using MongoDB and Mongoose.
  + MongoDB is where we store data.
  + Mongoose is a object data model library for MongoDB and Node.js