Masters Module - Detailed Sequence Diagrams

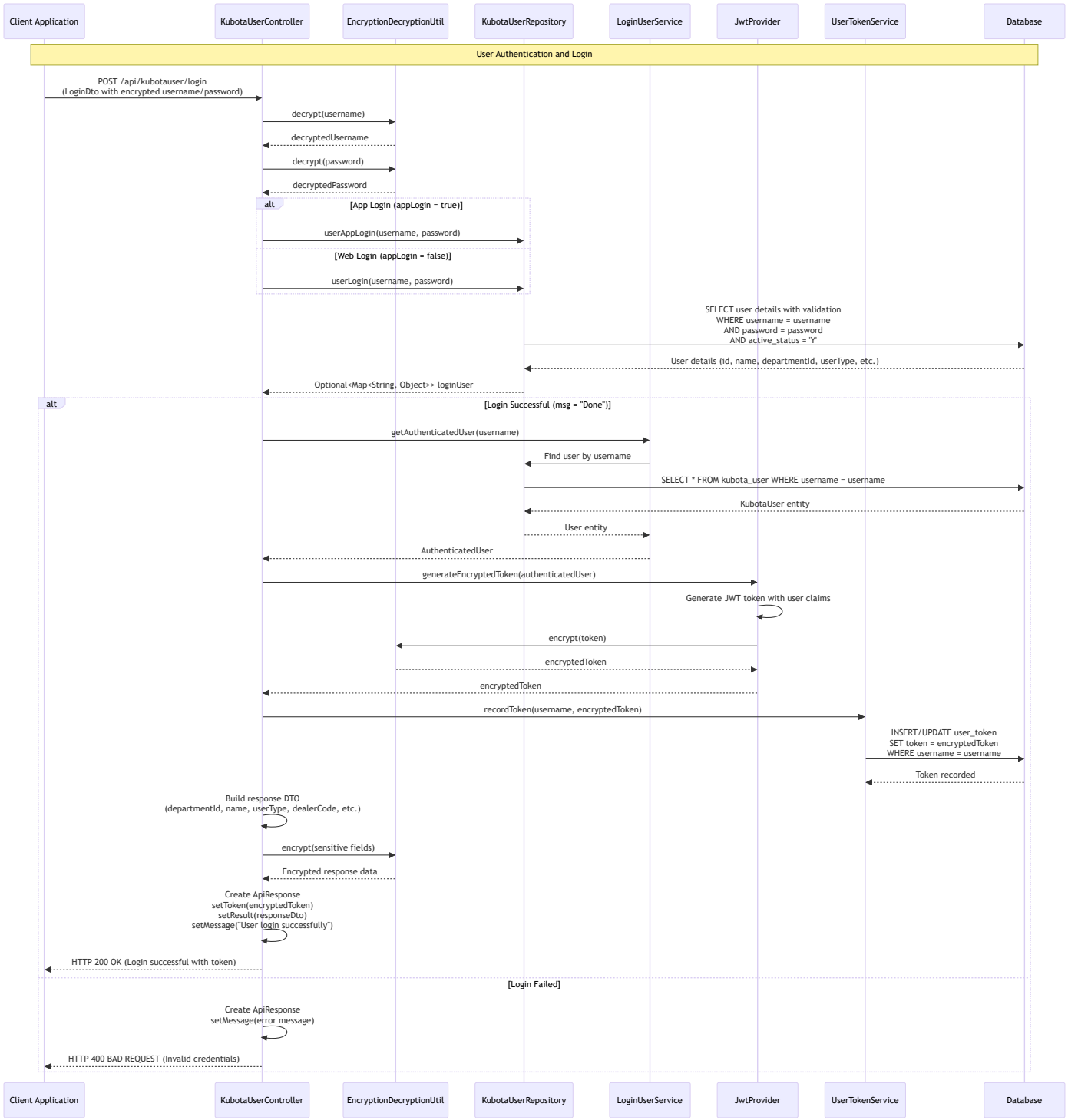
This document describes the **key technical flows** implemented in the `com.i4o.dms.kubota.masters` module:

- **User Authentication and Login** (Kubota user login with JWT token generation).
- **User Management** (Kubota user creation, role assignment, functionality mapping).
- **Customer Master Management** (Customer creation, search, change request with approval workflow).
- **Product Master Management** (Item/Machine master search, product hierarchy navigation).
- **Area Master Management** (Geographic hierarchy management: State, District, Tehsil, City, PinCode).
- **Dealer Master Management** (Dealer creation, search, territory management).
- **Employee Management** (Kubota employee and dealer employee management with organizational hierarchy).

All diagrams use Mermaid sequence diagrams and reflect the current implementation of the Masters module.

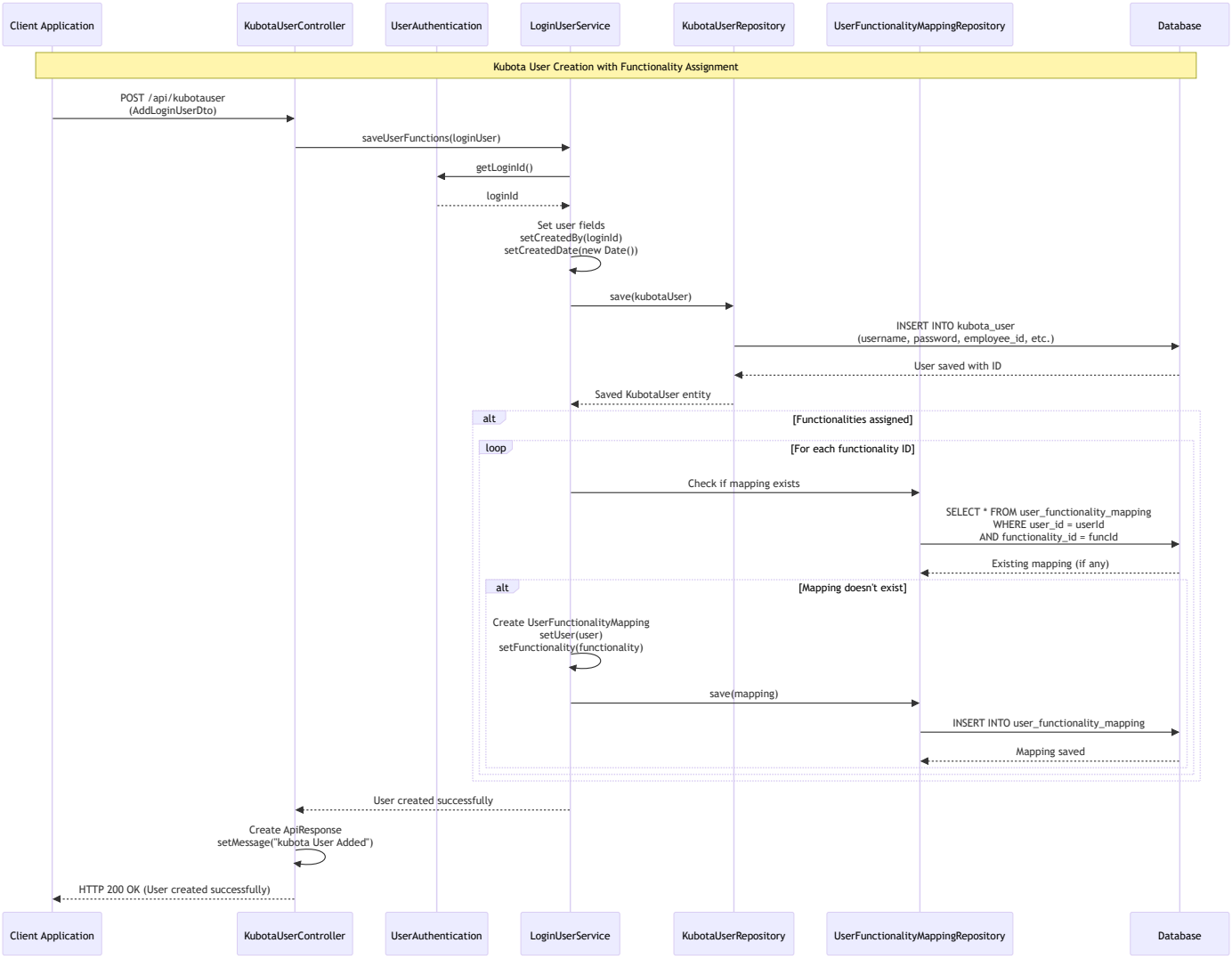
1. User Authentication and Login Flow

This flow shows how **Kubota users** authenticate and receive JWT tokens for accessing the system.



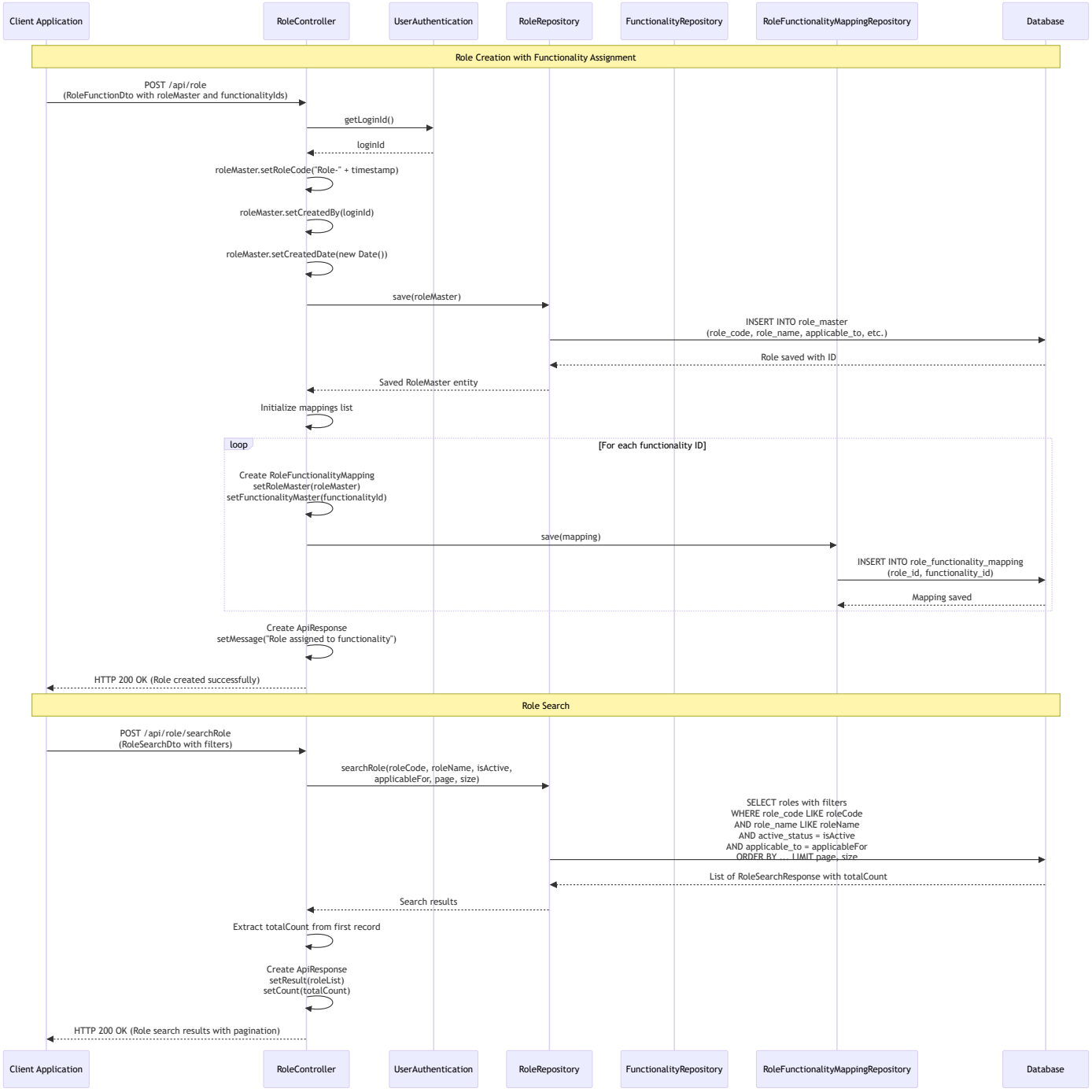
2. User Creation and Functionality Assignment Flow

This flow shows how **new Kubota users** are created and assigned roles and functionalities.



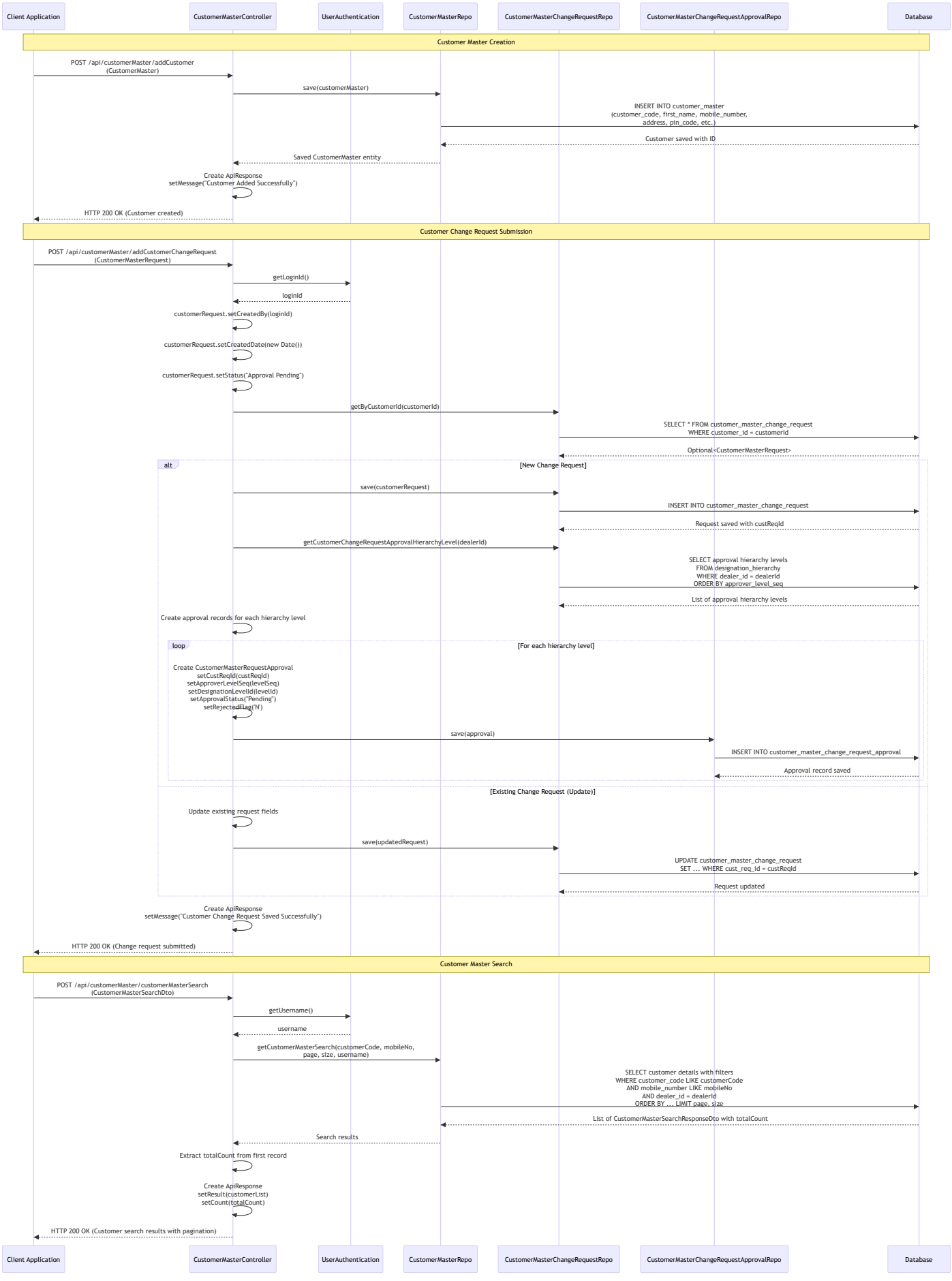
3. Role Creation and Functionality Assignment Flow

This flow shows how **roles** are created and assigned functionalities in a hierarchical structure.



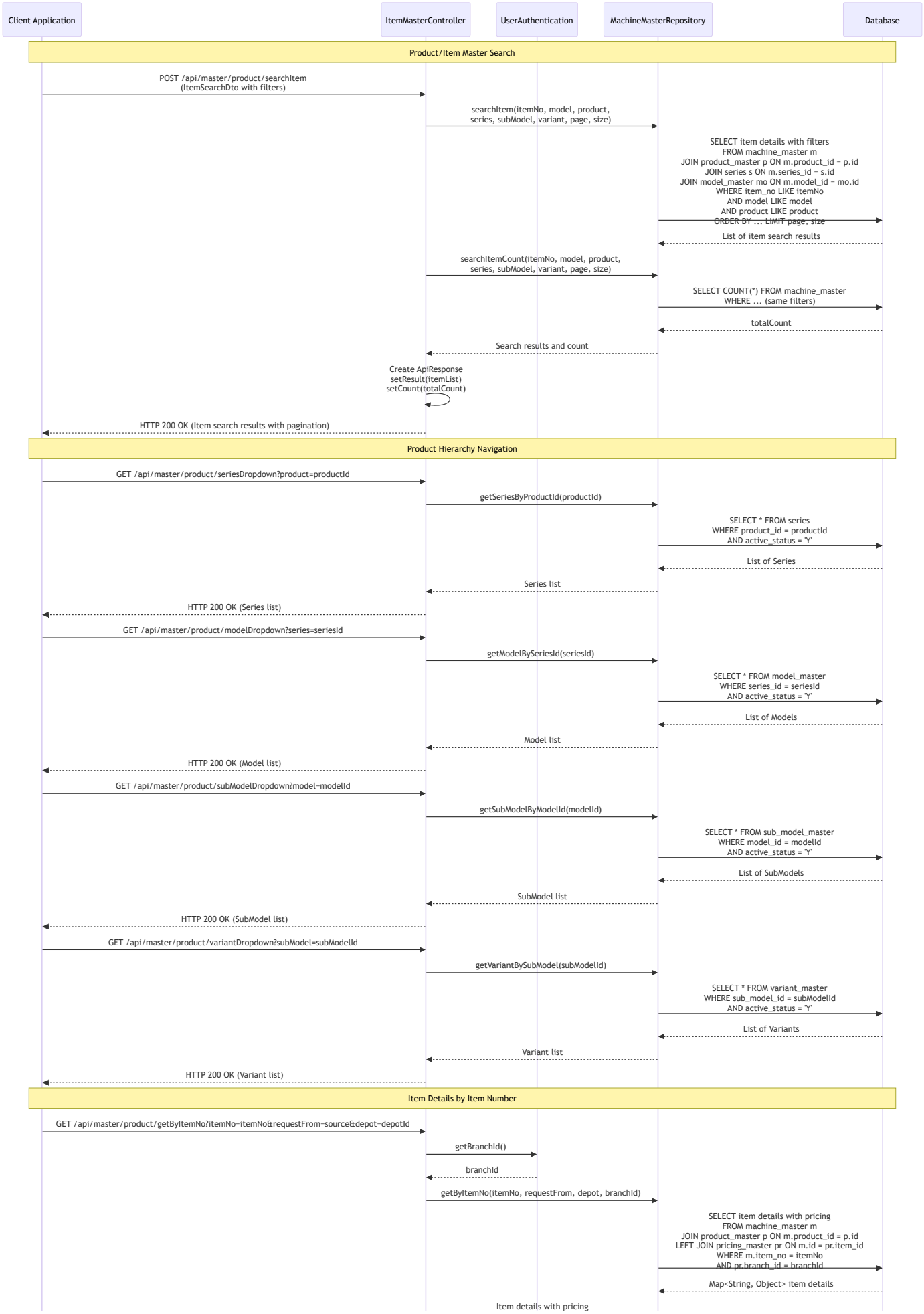
4. Customer Master Creation and Search Flow

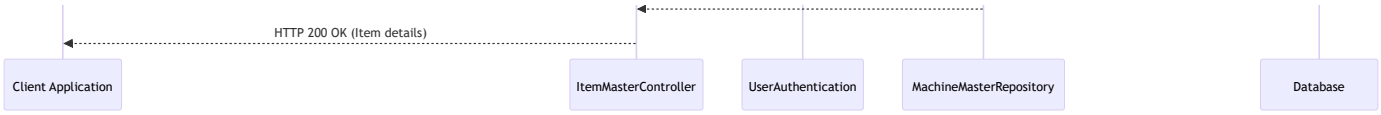
This flow shows how **customers** are created, searched, and change requests are submitted with approval workflow.


Client Application
CustomerMasterController
UserAuthentication
CustomerMasterRepo
CustomerMasterChangeRequestRepo
CustomerMasterChangeRequestApprovalRepo
Database
Customer Master Creation
POST /api/customerMaster/addCustomer (CustomerMaster)
save(customerMaster)
INSERT INTO customer_master (customer_code, first_name, mobile_number, address, pin_code, etc.)
Customer saved with ID
Saved CustomerMaster entity
Create ApiResponse setMessage("Customer Added Successfully")
HTTP 200 OK (Customer created)
Customer Change Request Submission
POST /api/customerMaster/addCustomerChangeRequest (CustomerMasterRequest)
getLoginId()
loginId
customerRequest.setCreatedBy(loginId)
customerRequest.setCreatedDate(new Date())
customerRequest.setStatus("Approval Pending")
getByCustomerId(customerId)
SELECT * FROM customer_master_change_request WHERE customer_id = customerId
Optional<CustomerMasterRequest>
alt
[New Change Request]
save(customerRequest)
INSERT INTO customer_master_change_request
Request saved with custReqId
getCustomerChangeRequestApprovalHierarchyLevel(dealerId)
SELECT approval hierarchy levels FROM designation_hierarchy WHERE dealer_id = dealerId ORDER BY approver_level_seq
List of approval hierarchy levels
Create approval records for each hierarchy level
loop
[For each hierarchy level]
Create CustomerMasterRequestApproval setCustReqId(custReqId) setApproverLevelSeq(levelSeq) setDesignationLevelId(levelId) setApprovalStatus("Pending") setRejectedFlag('N')
save(approval)
INSERT INTO customer_master_change_request_approval
Approval record saved
[Existing Change Request (Update)]
Update existing request fields
save(updatedRequest)
UPDATE customer_master_change_request SET ... WHERE cust_req_id = custReqId
Request updated
Create ApiResponse setMessage("Customer Change Request Saved Successfully")
HTTP 200 OK (Change request submitted)
Customer Master Search
POST /api/customerMaster/customerMasterSearch (CustomerMasterSearchDto)
getUsername()
username
getCustomerMasterSearch(customerCode, mobileNo, page, size, username)
SELECT customer details with filters WHERE customer_code LIKE customerCode AND mobile_number LIKE mobileNo AND dealer_id = dealerId ORDER BY ... LIMIT page, size
List of CustomerMasterSearchResponseDto with totalCount
Search results
Extract totalCount from first record
Create ApiResponse setResult(customerList) setCount(totalCount)
HTTP 200 OK (Customer search results with pagination)
Client Application
CustomerMasterController
UserAuthentication
CustomerMasterRepo
CustomerMasterChangeRequestRepo
CustomerMasterChangeRequestApprovalRepo
Database

5. Product Master Search and Hierarchy Navigation Flow

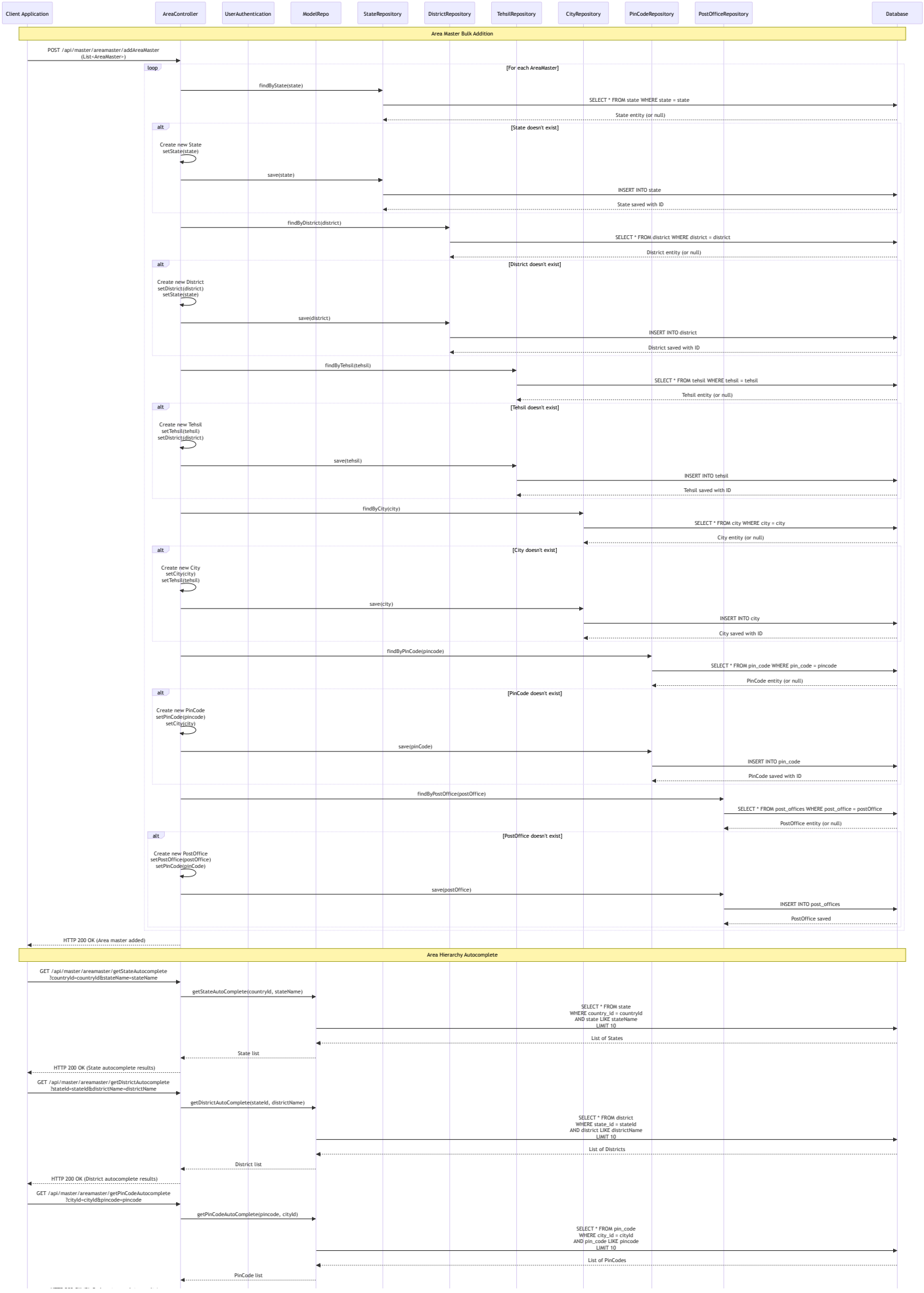
This flow shows how **product/item masters** are searched and product hierarchies (Series, Model, SubModel, Variant) are navigated.


Client Application
ItemMasterController
UserAuthentication
MachineMasterRepository
Database
Product/Item Master Search
POST /api/master/product/searchItem
(ItemSearchDto with filters)
searchItem(itemNo, model, product,
series, subModel, variant, page, size)
SELECT item details with filters
FROM machine_master m
JOIN product_master p ON m.product_id = p.id
JOIN series s ON m.series_id = s.id
JOIN model_master mo ON m.model_id = mo.id
WHERE item_no LIKE itemNo
AND model LIKE model
AND product LIKE product
ORDER BY ... LIMIT page, size
List of item search results
searchItemCount(itemNo, model, product,
series, subModel, variant, page, size)
SELECT COUNT(*) FROM machine_master
WHERE ... (same filters)
totalCount
Search results and count
Create ApiResponse
setResult(itemList)
setCount(totalCount)
HTTP 200 OK (Item search results with pagination)
Product Hierarchy Navigation
GET /api/master/product/seriesDropdown?product=productId
getSeriesByProductId(productId)
SELECT * FROM series
WHERE product_id = productId
AND active_status = 'Y'
List of Series
Series list
HTTP 200 OK (Series list)
GET /api/master/product/modelDropdown?series=seriesId
getModelBySeriesId(seriesId)
SELECT * FROM model_master
WHERE series_id = seriesId
AND active_status = 'Y'
List of Models
Model list
HTTP 200 OK (Model list)
GET /api/master/product/subModelDropdown?model=modelId
getSubModelByModelId(modelId)
SELECT * FROM sub_model_master
WHERE model_id = modelId
AND active_status = 'Y'
List of SubModels
SubModel list
HTTP 200 OK (SubModel list)
GET /api/master/product/variantDropdown?subModel=subModelId
getVariantBySubModel(subModelId)
SELECT * FROM variant_master
WHERE sub_model_id = subModelId
AND active_status = 'Y'
List of Variants
Variant list
HTTP 200 OK (Variant list)
Item Details by Item Number
GET /api/master/product/getByItemNo?itemNo=itemNo&requestFrom=source&depot=depotId
getBranchId()
branchId
getByItemNo(itemNo, requestFrom, depot, branchId)
SELECT item details with pricing
FROM machine_master m
JOIN product_master p ON m.product_id = p.id
LEFT JOIN pricing_master pr ON m.id = pr.item_id
WHERE m.item_no = itemNo
AND pr.branch_id = branchId
Map<String, Object> item details
Item details with pricing



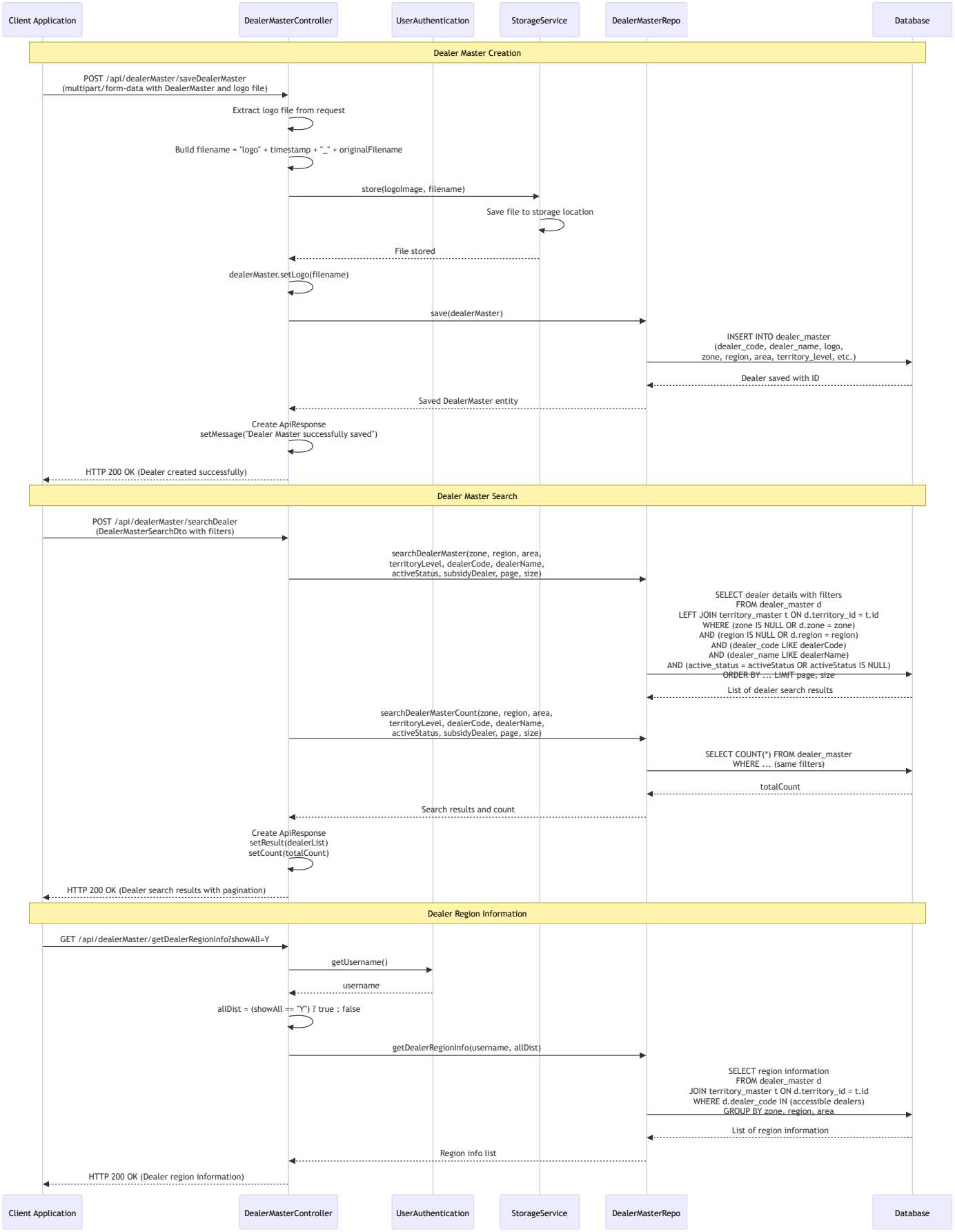
6. Area Master Management Flow

This flow shows how **geographic hierarchies** (State, District, Tehsil, City, PinCode, PostOffice) are managed and queried.


Client Application
AreaController
UserAuthentication
ModelRepo
StateRepository
DistrictRepository
TehsilRepository
CityRepository
PinCodeRepository
PostOfficeRepository
Database
Area Master Bulk Addition
POST /api/master/areamaster/addAreaMaster (List<AreaMaster>)
loop
[For each AreaMaster]
findByState(state)
SELECT * FROM state WHERE state = state
State entity (or null)
alt
[State doesn't exist]
Create new State setState(state)
save(state)
INSERT INTO state
State saved with ID
findByDistrict(district)
SELECT * FROM district WHERE district = district
District entity (or null)
alt
[District doesn't exist]
Create new District setDistrict(district) setState(state)
save(district)
INSERT INTO district
District saved with ID
findByTehsil(tehsil)
SELECT * FROM tehsil WHERE tehsil = tehsil
Tehsil entity (or null)
alt
[Tehsil doesn't exist]
Create new Tehsil setTehsil(tehsil) setDistrict(district)
save(tehsil)
INSERT INTO tehsil
Tehsil saved with ID
findByCity(city)
SELECT * FROM city WHERE city = city
City entity (or null)
alt
[City doesn't exist]
Create new City setCity(city) setTehsil(tehsil)
save(city)
INSERT INTO city
City saved with ID
findByPinCode(pincode)
SELECT * FROM pin_code WHERE pin_code = pincode
PinCode entity (or null)
alt
[PinCode doesn't exist]
Create new PinCode setPinCode(pincode) setCity(city)
save(pinCode)
INSERT INTO pin_code
PinCode saved with ID
findByPostOffice(postOffice)
SELECT * FROM post_offices WHERE post_office = postOffice
PostOffice entity (or null)
alt
[PostOffice doesn't exist]
Create new PostOffice setPostOffice(postOffice) setPinCode(pinCode)
save(postOffice)
INSERT INTO post_offices
PostOffice saved
HTTP 200 OK (Area master added)
Area Hierarchy Autocomplete
GET /api/master/areamaster/getStateAutocomplete ?countryId=countryId&stateName=stateName
getStateAutoComplete(countryId, stateName)
SELECT * FROM state WHERE country_id = countryId AND state LIKE stateName LIMIT 10
List of States
State list
HTTP 200 OK (State autocomplete results)
GET /api/master/areamaster/getDistrictAutocomplete ?stateId=stateId&districtName=districtName
getDistrictAutoComplete(stateId, districtName)
SELECT * FROM district WHERE state_id = stateId AND district LIKE districtName LIMIT 10
List of Districts
District list
HTTP 200 OK (District autocomplete results)
GET /api/master/areamaster/getPinCodeAutocomplete ?cityId=cityId&pincode=pincode
getPinCodeAutoComplete(pincode, cityId)
SELECT * FROM pin_code WHERE city_id = cityId AND pin_code LIKE pincode LIMIT 10
List of PinCodes
PinCode list

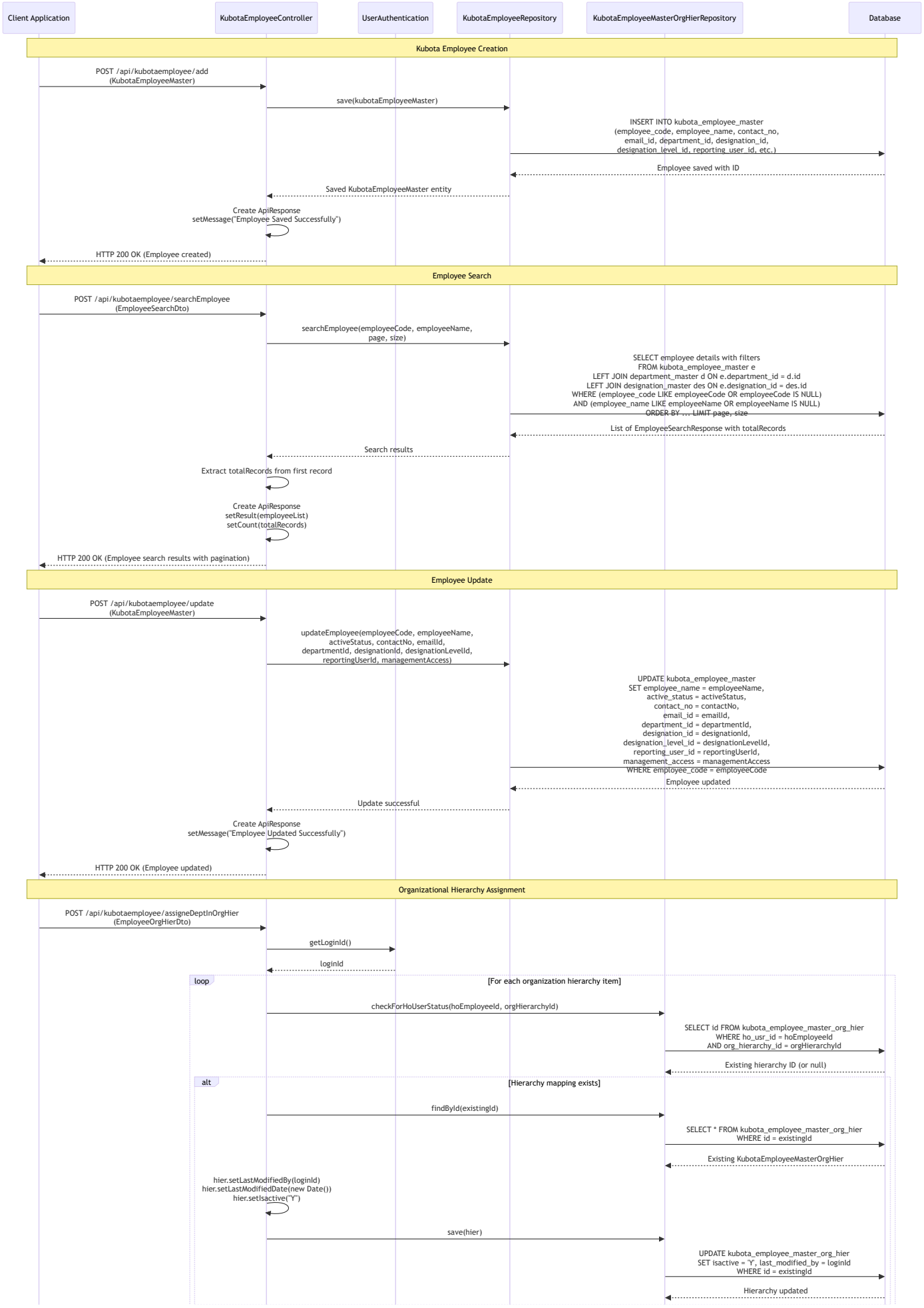
7. Dealer Master Management Flow

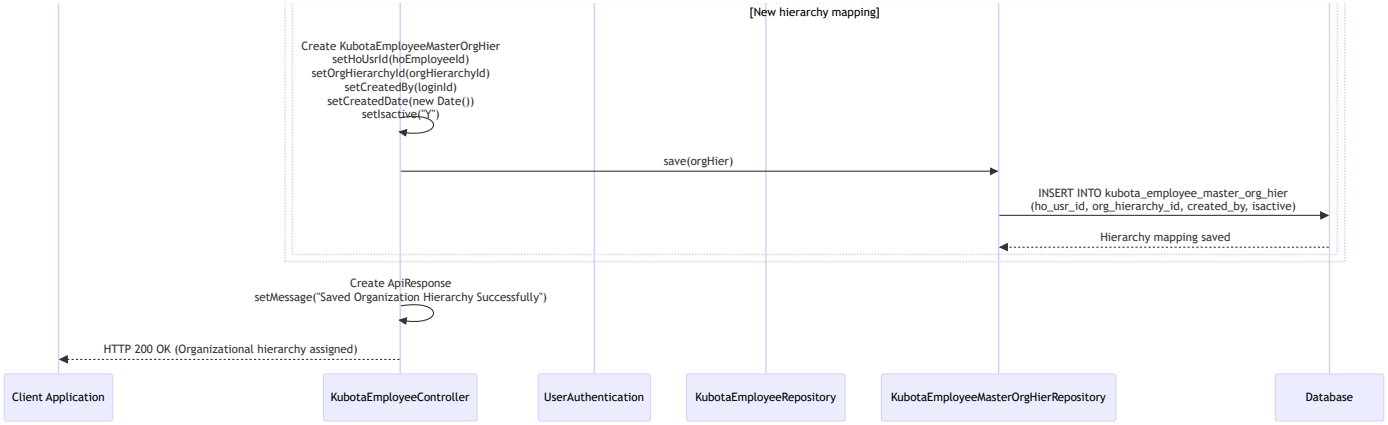
This flow shows how **dealers** are created, searched, and managed with territory and region information.


Client Application
DealerMasterController
UserAuthentication
StorageService
DealerMasterRepo
Database
Dealer Master Creation
POST /api/dealerMaster/saveDealerMaster
(multipart/form-data with DealerMaster and logo file)
Extract logo file from request
Build filename = "logo" + timestamp + "_" + originalFilename
store(logoImage, filename)
Save file to storage location
File stored
dealerMaster.setLogo(filename)
save(dealerMaster)
INSERT INTO dealer_master
(dealer_code, dealer_name, logo,
zone, region, area, territory_level, etc.)
Dealer saved with ID
Saved DealerMaster entity
Create ApiResponse
setMessage("Dealer Master successfully saved")
HTTP 200 OK (Dealer created successfully)
Dealer Master Search
POST /api/dealerMaster/searchDealer
(DealerMasterSearchDto with filters)
searchDealerMaster(zone, region, area,
territoryLevel, dealerCode, dealerName,
activeStatus, subsidyDealer, page, size)
SELECT dealer details with filters
FROM dealer_master d
LEFT JOIN territory_master t ON d.territory_id = t.id
WHERE (zone IS NULL OR d.zone = zone)
AND (region IS NULL OR d.region = region)
AND (dealer_code LIKE dealerCode)
AND (dealer_name LIKE dealerName)
AND (active_status = activeStatus OR activeStatus IS NULL)
ORDER BY ... LIMIT page, size
List of dealer search results
searchDealerMasterCount(zone, region, area,
territoryLevel, dealerCode, dealerName,
activeStatus, subsidyDealer, page, size)
SELECT COUNT(*) FROM dealer_master
WHERE ... (same filters)
totalCount
Search results and count
Create ApiResponse
setResult(dealerList)
setCount(totalCount)
HTTP 200 OK (Dealer search results with pagination)
Dealer Region Information
GET /api/dealerMaster/getDealerRegionInfo?showAll=Y
getUsername()
username
allDist = (showAll == "Y") ? true : false
getDealerRegionInfo(username, allDist)
SELECT region information
FROM dealer_master d
JOIN territory_master t ON d.territory_id = t.id
WHERE d.dealer_code IN (accessible dealers)
GROUP BY zone, region, area
List of region information
Region info list
HTTP 200 OK (Dealer region information)
Client Application
DealerMasterController
UserAuthentication
StorageService
DealerMasterRepo
Database

8. Employee Management Flow

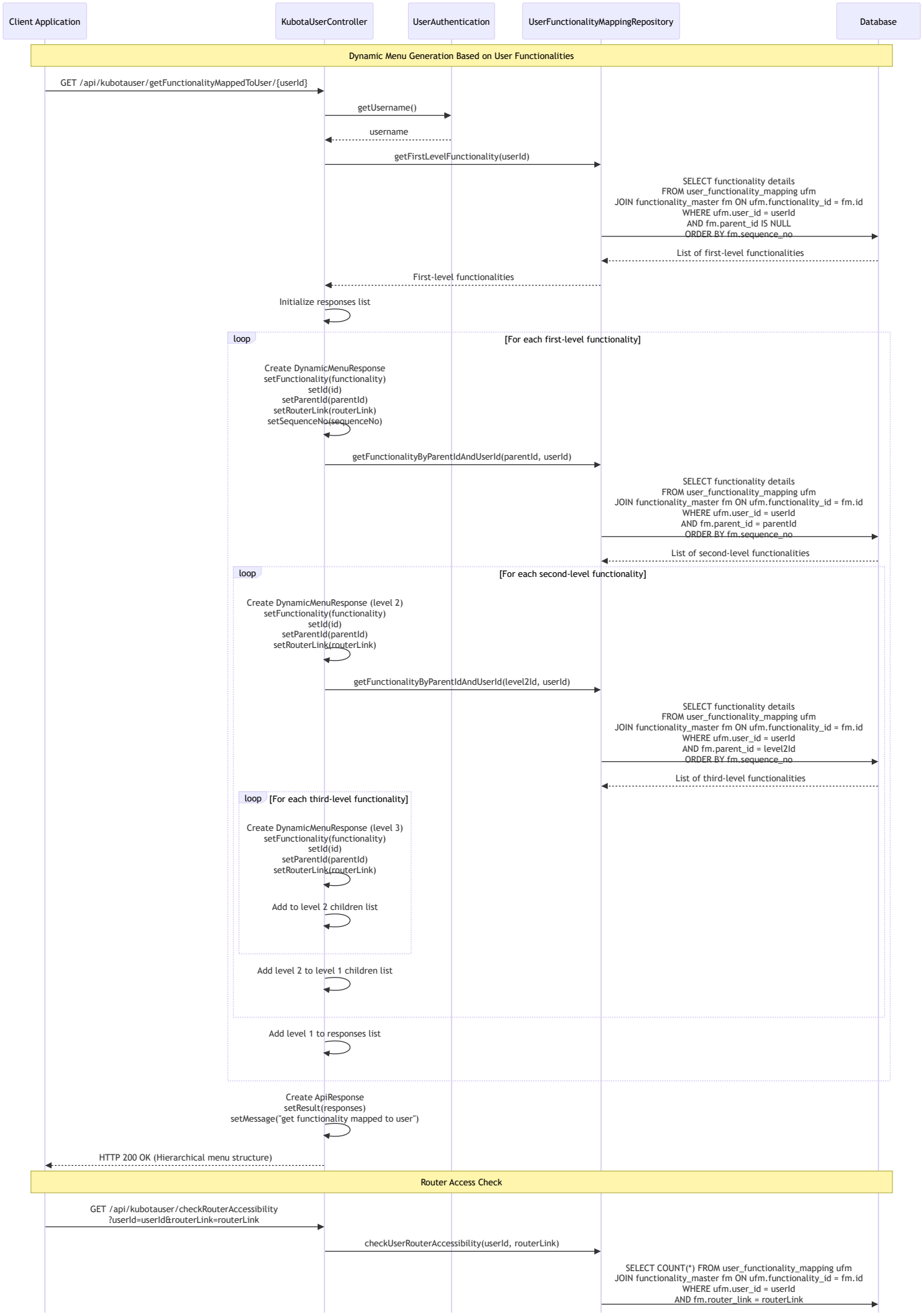
This flow shows how **Kubota employees** are created, updated, searched, and organizational hierarchy is managed.

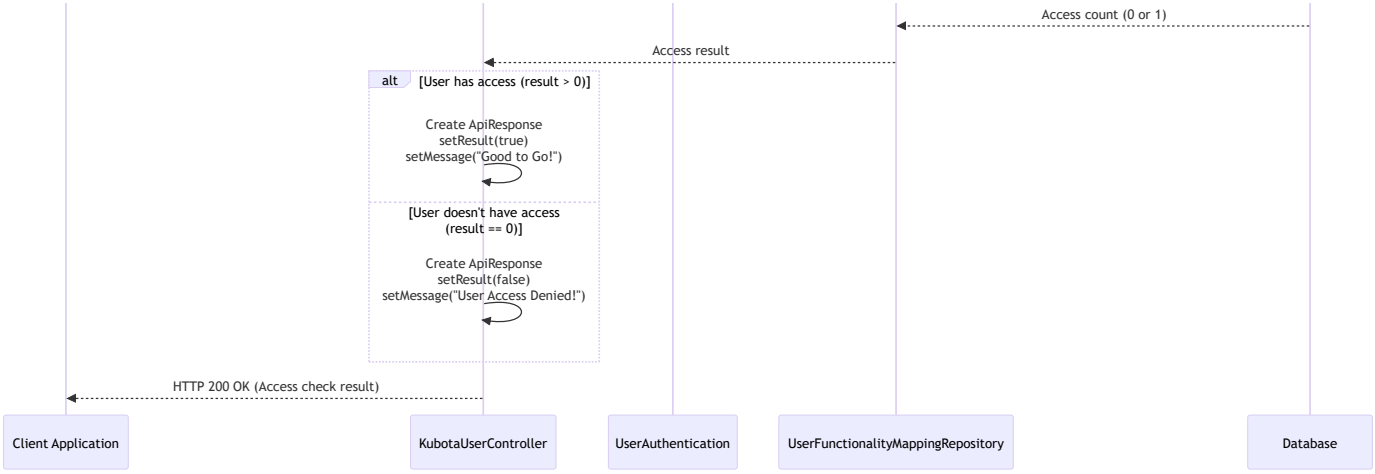

Client Application
KubotaEmployeeController
UserAuthentication
KubotaEmployeeRepository
KubotaEmployeeMasterOrgHierRepository
Database
Kubota Employee Creation
POST /api/kubotaemployee/add
(KubotaEmployeeMaster)
save(kubotaEmployeeMaster)
INSERT INTO kubota_employee_master
(employee_code, employee_name, contact_no,
email_id, department_id, designation_id,
designation_level_id, reporting_user_id, etc.)
Employee saved with ID
Saved KubotaEmployeeMaster entity
Create ApiResponse
setMessage("Employee Saved Successfully")
HTTP 200 OK (Employee created)
Employee Search
POST /api/kubotaemployee/searchEmployee
(EmployeeSearchDto)
searchEmployee(employeeCode, employeeName,
page, size)
SELECT employee details with filters
FROM kubota_employee_master e
LEFT JOIN department_master d ON e.department_id = d.id
LEFT JOIN designation_master des ON e.designation_id = des.id
WHERE (employee_code LIKE employeeCode OR employeeCode IS NULL)
AND (employee_name LIKE employeeName OR employeeName IS NULL)
ORDER BY ... LIMIT page, size
List of EmployeeSearchResponse with totalRecords
Search results
Extract totalRecords from first record
Create ApiResponse
setResult(employeeList)
setCount(totalRecords)
HTTP 200 OK (Employee search results with pagination)
Employee Update
POST /api/kubotaemployee/update
(KubotaEmployeeMaster)
updateEmployee(employeeCode, employeeName,
activeStatus, contactNo, emailId,
departmentId, designationId, designationLevelId,
reportingUserId, managementAccess)
UPDATE kubota_employee_master
SET employee_name = employeeName,
active_status = activeStatus,
contact_no = contactNo,
email_id = emailId,
department_id = departmentId,
designation_id = designationId,
designation_level_id = designationLevelId,
reporting_user_id = reportingUserId,
management_access = managementAccess
WHERE employee_code = employeeCode
Employee updated
Update successful
Create ApiResponse
setMessage("Employee Updated Successfully")
HTTP 200 OK (Employee updated)
Organizational Hierarchy Assignment
POST /api/kubotaemployee/assigneDeptInOrgHier
(EmployeeOrgHierDto)
getLoginId()
loginId
loop
[For each organization hierarchy item]
checkForHoUserStatus(hoEmployeeId, orgHierarchyId)
SELECT id FROM kubota_employee_master_org_hier
WHERE ho_usr_id = hoEmployeeId
AND org_hierarchy_id = orgHierarchyId
Existing hierarchy ID (or null)
alt
[Hierarchy mapping exists]
findById(existingId)
SELECT * FROM kubota_employee_master_org_hier
WHERE id = existingId
Existing KubotaEmployeeMasterOrgHier
hier.setLastModifiedBy(loginId)
hier.setLastModifiedDate(new Date())
hier.setIsactive("Y")
save(hier)
UPDATE kubota_employee_master_org_hier
SET isactive = 'Y', last_modified_by = loginId
WHERE id = existingId
Hierarchy updated



9. Dynamic Menu Generation Flow

This flow shows how **dynamic menus** are generated based on user functionality mappings and role assignments.


Client Application
KubotaUserController
UserAuthentication
UserFunctionalityMappingRepository
Database
Dynamic Menu Generation Based on User Functionalities
GET /api/kubotauser/getFunctionalityMappedToUser/{userId}
getUsername()
username
getFirstLevelFunctionality(userId)
SELECT functionality details
FROM user_functionality_mapping ufm
JOIN functionality_master fm ON ufm.functionality_id = fm.id
WHERE ufm.user_id = userId
AND fm.parent_id IS NULL
ORDER BY fm.sequence_no
List of first-level functionalities
First-level functionalities
Initialize responses list
loop
[For each first-level functionality]
Create DynamicMenuResponse
setFunctionality(functionality)
setId(id)
setParentId(parentId)
setRouterLink(routerLink)
setSequenceNo(sequenceNo)
getFunctionalityByParentIdAndUserId(parentId, userId)
SELECT functionality details
FROM user_functionality_mapping ufm
JOIN functionality_master fm ON ufm.functionality_id = fm.id
WHERE ufm.user_id = userId
AND fm.parent_id = parentId
ORDER BY fm.sequence_no
List of second-level functionalities
loop
[For each second-level functionality]
Create DynamicMenuResponse (level 2)
setFunctionality(functionality)
setId(id)
setParentId(parentId)
setRouterLink(routerLink)
getFunctionalityByParentIdAndUserId(level2Id, userId)
SELECT functionality details
FROM user_functionality_mapping ufm
JOIN functionality_master fm ON ufm.functionality_id = fm.id
WHERE ufm.user_id = userId
AND fm.parent_id = level2Id
ORDER BY fm.sequence_no
List of third-level functionalities
loop
[For each third-level functionality]
Create DynamicMenuResponse (level 3)
setFunctionality(functionality)
setId(id)
setParentId(parentId)
setRouterLink(routerLink)
Add to level 2 children list
Add level 2 to level 1 children list
Add level 1 to responses list
Create ApiResponse
setResult(responses)
setMessage("get functionality mapped to user")
HTTP 200 OK (Hierarchical menu structure)
Router Access Check
GET /api/kubotauser/checkRouterAccessibility
?userId=userId&routerLink=routerLink
checkUserRouterAccessibility(userId, routerLink)
SELECT COUNT(*) FROM user_functionality_mapping ufm
JOIN functionality_master fm ON ufm.functionality_id = fm.id
WHERE ufm.user_id = userId
AND fm.router_link = routerLink



Summary

The **Masters module** provides comprehensive **master data management** functionality for the KUBOTA DMS:

- **User Management**:
  - User authentication with JWT token generation and encryption
  - User creation with role and functionality assignment
  - Dynamic menu generation based on user permissions
  - Router access control and validation
  - Password reset and forgot password functionality
- **Role and Functionality Management**:
  - Role creation with hierarchical functionality assignment
  - Role search with filters (role code, name, active status, applicable to)
  - Functionality tree generation for role assignment
  - Role update with functionality remapping
- **Customer Master**:
  - Customer creation and search with multiple filters
  - Customer change request workflow with multi-level approval hierarchy
  - Customer validation (mobile number, customer code)
  - Vehicle details and chassis number tracking
  - Customer autocomplete functionality
- **Product Master**:
  - Product/item search with filters (item number, model, product, series, submodel, variant)
  - Product hierarchy navigation (Product → Series → Model → SubModel → Variant)
  - Item details retrieval with pricing information
  - Product autocomplete and dropdown population

- **Area Master**:
  - Geographic hierarchy management (Country → State → District → Tehsil → City → PinCode → PostOffice)
  - Bulk area master addition with automatic parent entity creation
  - Area hierarchy autocomplete for all levels
  - PinCode and locality lookup
- **Dealer Master**:
  - Dealer creation with logo upload
  - Dealer search with multiple filters (zone, region, area, territory, dealer code/name, status)
  - Dealer region information retrieval
  - Territory and area level management
- **Employee Management**:
  - Kubota employee creation, update, and search
  - Organizational hierarchy assignment
  - Employee status management (active/inactive)
  - Reporting hierarchy management
  - Department and designation management
- **Spares Master**:
  - Spare part master search and autocomplete
  - Spare part details retrieval for MRC and quotation
  - Item number validation and lookup

The Masters module serves as the foundation for all other modules in the KUBOTA DMS, providing essential master data that is referenced throughout the system for sales, service, warranty, CRM, and other business processes. All master data operations include proper authentication, authorization checks, and audit trail maintenance (created by, created date, last modified by, last modified date).