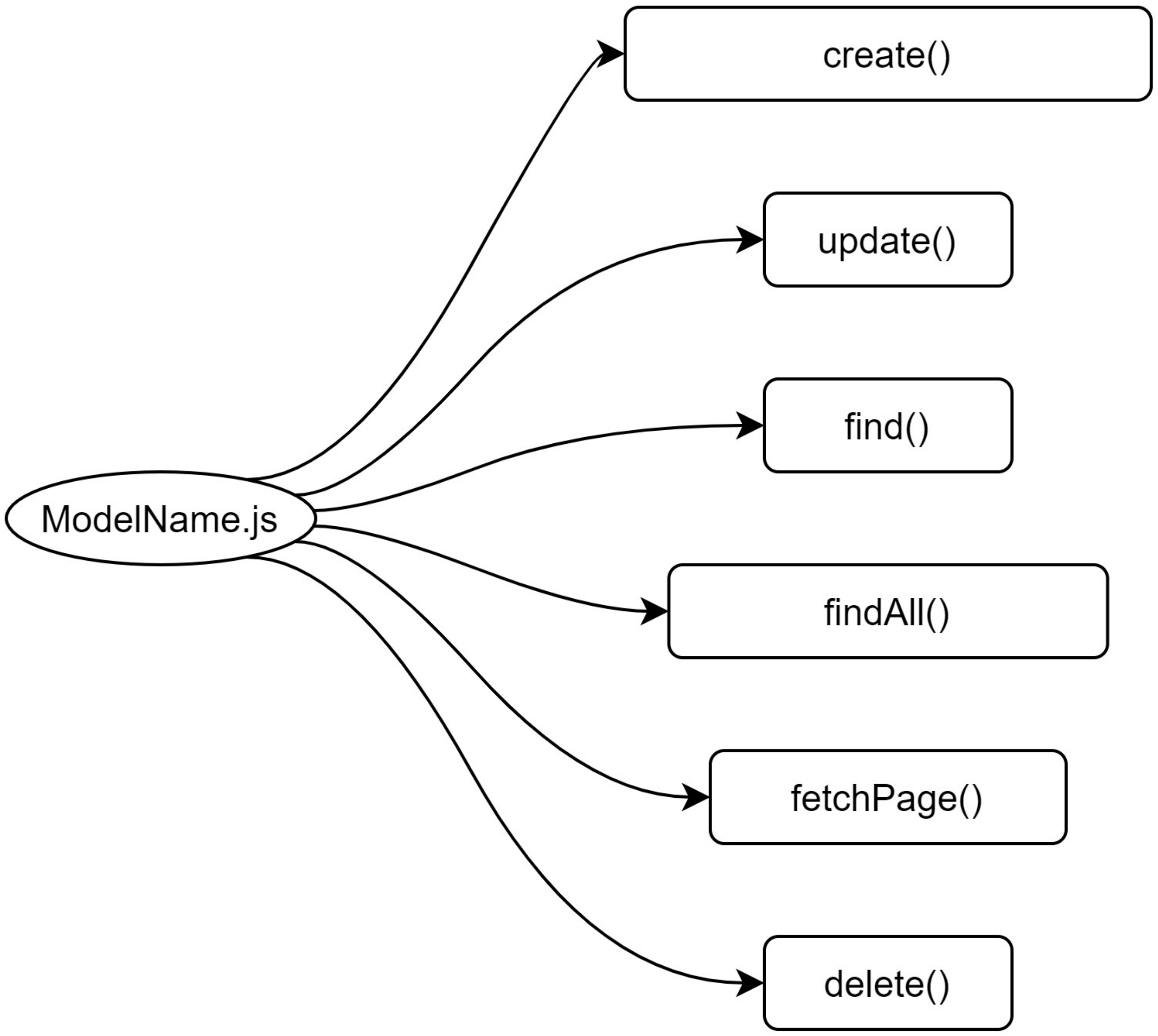
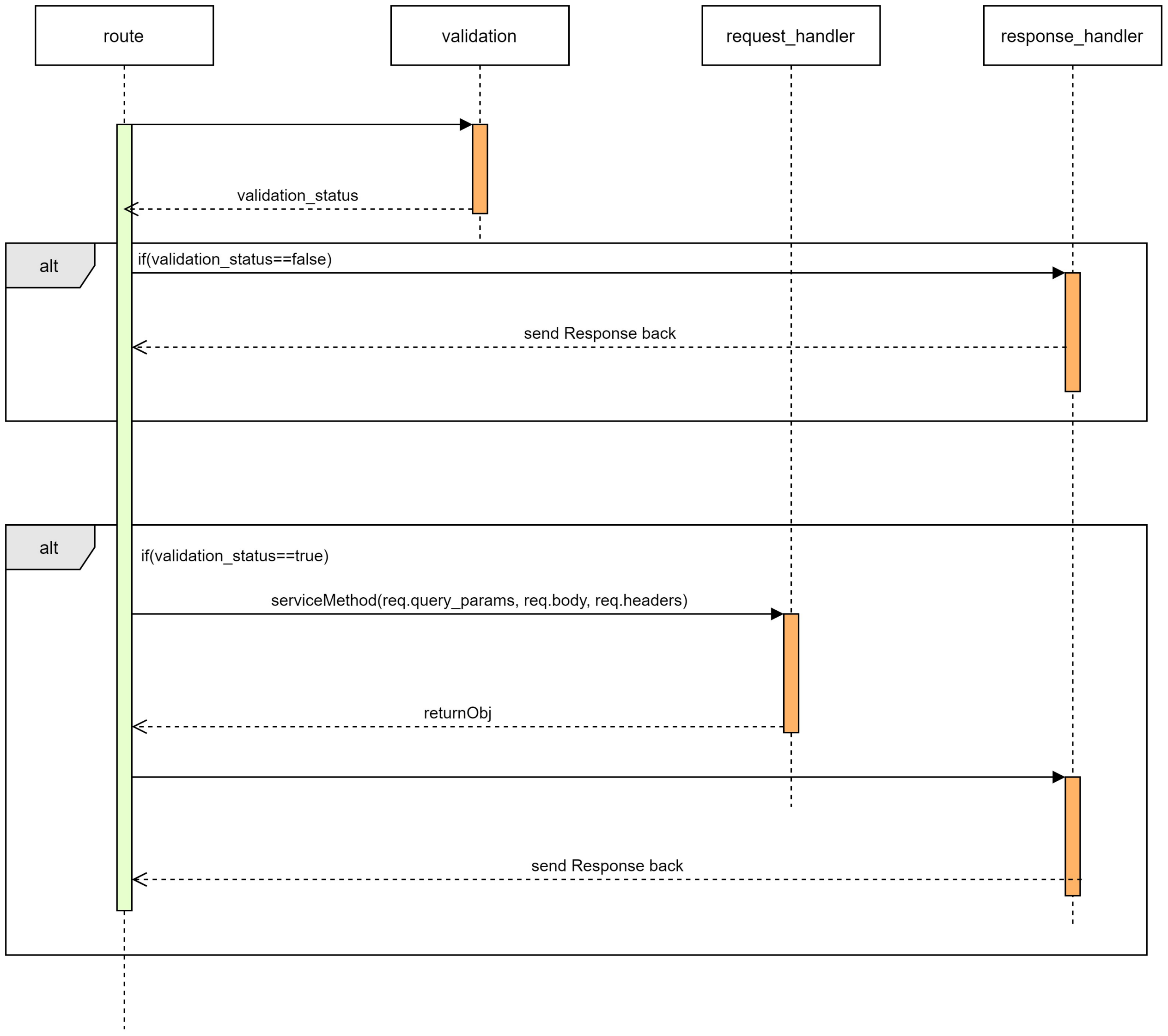
Model Implimentation



- Based on microservice and bussiness logic defined there will be atleast 2-5 or more tables or collections.
- FOR EACH collection or table there should be a model file and should must have collection or table name as model name in captial camil case conventions.
    - example: If table name is "login" then model name will be "Login.js"
    - example: If collection name is "order_header" then model name will be "OrderHeader.js"
- "ModelName.js" should expose create(), update(), find(), findAll(), fetchPage() , delete(). So, when these model is imported these exposed methods should be used to communicate with data layer and shouldn't communicate with direct meathods of "bookshelf" or "mongoose".
    - **create()**: This method impliments creating new entity in the database and then it saves it and then sends new_data and old_data as empty to s3_bucket.

- **update():** This method impiments updating the existing entity and sends new_data and old_data along with requestObj to s3_bucket.
- **find():** Gets single document/row based on query options.
- findAll(): Get All documents/rows based on query options.
- **fetchPage()**:Impliments pagination concept and returns the results with pagination object.
- **delete()**: This delete doesn't delete the acutal document/row but updates "is_archive" to "0" -> "1" then sends new_data and old_data along with requestObj to s3_bucket.

Route Implementation



- Incomming request is send for validation where it will validate the query_params, body and headers.
- If validation fails then it will immediatly sends response back with relative status_code.
- If the request passed the validation it will farward "req_object" to request_handler and then it will farward whatever the data is returned from requst_handler to response_handler.

Service Implimenation

- It will impliment the bussiness logic or application logic.

- This serviceName.js will import the modelName.js and performs the bussiness_logic/application_logic
- Some of the coding standards are given below:
    - **Limited use of globals:**
        - These rules tell about which types of data that can be declared global and the data that can't be.
    - **Naming conventions for local variables, global variables, constants and functions:**
        - Some of the naming conventions are given below:
            - Meaningful and understandable variables name helps anyone to understand the reason of using it.
            - Local variables should be named using camel case lettering starting with small letter (e.g. **localData**) whereas Global variables names should start with a capital letter (e.g. **GlobalData**). Constant names should be formed using capital letters only (e.g. **CONSDATA**).
            - It is better to avoid the use of digits in variable names.
            - The names of the function should be written in camel case starting with small letters.
            - The name of the function must describe the reason of using the function clearly and briefly.
    - **Indentation:**
        - Proper indentation is very important to increase the readability of the code. For making the code readable, programmers should use White spaces properly. Some of the spacing conventions are given below:
            - There must be a space after giving a comma between two function arguments.
            - Each nested block should be properly indented and spaced.
            - Proper Indentation should be there at the beginning and at the end of each block in the program.
            - All braces should start from a new line and the code following the end of braces also start from a new line.
    - **Error return values and exception handling conventions:**
        - All functions that encountering an error condition should either return a true or false with well defined error_code.
        - All the the exceptions should be handled by **app-error** class.
        - Al the errors should be logged along with stack trace

    - **Avoid using a coding style that is too difficult to understand:**
        - Code should be easily understandable. The complex code makes maintenance and debugging difficult and expensive.
    - **Avoid using an identifier for multiple purposes:**

        - Each variable should be given a descriptive and meaningful name indicating the reason behind using it. This is not possible if an identifier is used for multiple purposes and thus it can lead to confusion to the reader. Moreover, it leads to more difficulty during future enhancements.
- **Code should be well documented:**
    - The code should be properly commented for understanding easily. Comments regarding the statements increase the understandability of the code.
- **Length of functions should not be very large:**
    - Lengthy functions are very difficult to understand. That's why functions should be small enough to carry out small work and lengthy functions should be broken into small ones for completing small tasks.
    - A function should impliment a specific task or logic.