Lars Palombi

# Code Explanation

## ./server/app.py

The file *app.py* contains configuration information for the Flask application. Here, we define our blueprints (this is where the routes are stored) as well as code for database configuration.

## ./server/migrations.py

This directory is automatically generated by Alembic which I use with Flask-mySQL. More details can be found here.

## ./server/api/vehicle.py

This file contains the entry point for this application.

The main route will retrieve a byte array sent by a given client application.

Theoretically, the client application would also specify the endianness of the byte array. However, if no endianness is specified, it will be detected in processing.py. More details on this below.

Once the data is retrieved from the client application, the byte array data (and endianness if specified) will get passed to the process() function in the ProcessingService object.

This class is located in ./server/service/processing.py.

## ./server/service/processing.py

This file is a service layer to extract the binary data from the byte array and convert it to the necessary formats (i.e. strings and short integers) for the insertion/retrieval of data from the database.

**Function name:** process

**Parameters**:
- data (bytes): The byte array from the client application
- Endian (string): The endianness specified by the client application (optional)

**Return value:**
-Bytes: Error message, success response, or vehicle data

**Explanation:**
1. The endianness is stored as an attribute.
2. The data is cast to bytes if not type “bytes” already.
3. The request type is processed.
4. The specified length is processed and converted to an integer.
5. Returns error if error while processing length.
6. License plate is processed and converted to a string.
7. Returns error if error while processing license plate.
8. If insertion, convert axel / height data from bytes to integers.
9. Pass to Vehicle class to perform database insertion.
10. Return success or error response from database.
11. If retrieval, pass license plate to vehicle class to perform database retrieval.
12. Return vehicle data in bytes format or error response from database.
13. Return error if invalid request type.

**Function name:** retrieve_data

**Parameters:**
- data (bytes): The byte array from the client application

**Return value:**
- None

**Explanation:**
1. Detects type of data.
2. If data is not in bytes, cast to bytes.
3. Assign data to binary_data attribute.

**Function name:** getType

**Parameters:**
- None

**Return value:**
- None

**Explanation:**
1. Retrieves the request type from the third byte.
2. Assigns request type to type attribute.

Lars Palombi

**Function name:** processLength

**Parameters:** None

**Return value:**
- Bytes if error message, integer (specified length) if no error

**Explanation:**
1. Gets the length of the byte array and subtracts two to only get the length of the data portion.
2. If endianness is not specified, pass data_length to detectEndianness() function.
3. If an error is returned, return error message
4. Retrieve specified length from first two bytes of array and convert to integer
5. Assign length attribute to specified length
6. Return length

**Function name:** processPlate

**Parameters:**
- None

**Return value:**
- Bytes if error message, integer (0) if no error

**Explanation:**
1. License plate extracted from byte array and decoded to a string
2. If the license plate contains non-alphanumeric characters or it is empty, return an error.
3. Return 0 for success.

**Function name:** processDataForInsert

**Parameters:**
- None

**Return value:**
- None

**Explanation:**
1. Convert axel count from bytes to integer
2. Convert height from bytes to integer

Lars Palombi

**Function name:** detectEndianness

**Parameters:**
- data_length (integer): Length of data portion of byte array.

**Return value:**
- Bytes if error message, integer (specified length) if no error

**Explanation:**
1. Retrieves the first two bytes to get the length specified in the data message using big endian order.
2. If the specified length does not match the actual length of the data, get the length specified in the data message using little endian order.
3. If the specified length still does not match the actual length of the data, returns an error that the specified length of the data does not match the actual length.
4. If the lengths do match after using little endian, assign the endian attribute to 'little.'
5. If the lengths match after using big endian, assign the endian attribute to 'big.'
6. Return the specified length.

# ./server/service/error.py

This is a class used to convert error code and error message into bytes to return an error back to the client in the byte array format.

**Function name:** packageErrorResponse

**Parameters:**
- error_code (integer)
- error (string)

**Return value:**
- bytes

**Explanation:**
1. If the length of the error is >255 bytes, truncate it to contain only 255 bytes.
2. Return a byte array with the error code followed by the error message.

# ./server/models/db.py

Contains database instantiation code.

Lars Palombi

# ./server/models/vehicle.py

This class is used to represent the database tables for vehicles. It handles all database communication.

**Function name:** insert

**Parameters:**
- license_plate (string)
- axle_count (integer)
- height (integer)

**Return value:**
- bytes

**Explanation:**
1. Create a vehicle object using parameters
2. Add vehicle to database session
3. Insert into database
4. Return 0 for OK, empty field for error message
5. If there was a database error, convert error message to bytes. Return code 500 and error message.

**Function name:** retrieve

**Parameters:**
- license_plate (string)
- endian (string)

**Return value:**
- bytes

**Explanation:**
1. Retrieve the latest record based on the timestamp.
2. If no vehicle found, return error.
3. If vehicle found, return byte array with vehicle data.
4. If database error, return error.

Lars Palombi