writeup.pdf:

- which is your group's account of which parts of your submission you used Generative AI for (include a brief discussion of whether you modified the AI code and if so to what extent),

Following is the Code in OrderService.java that is AI generated,

```
private static HttpResponse<String> sendPostRequest(String endpoint, JsonNode
json) throws IOException, InterruptedException {
        HttpClient httpClient = HttpClient.newHttpClient();

        // Create the request with the target URL and JSON payload
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(endpoint))
                .header("Content-Type", "application/json")
                .POST(HttpRequest.BodyPublishers.ofString(json.toString()))
                .build();

        // Send the request and receive the response
        HttpResponse<String> response = httpClient.send(request,
HttpResponse.BodyHandlers.ofString());
        // System.out.println("SEND2");
        // System.out.println(response.statusCode());
        return response;

    }

    private static HttpResponse<String> sendGetRequest(String url) throws
IOException, InterruptedException {
        // Create an instance of HttpClient
        HttpClient client = HttpClient.newHttpClient();

        // Create a GET request
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(url))
                .GET()
                .build();

        // Send the request and handle the response
```

```
        HttpResponse<String> response = client.send(request,
HttpResponse.BodyHandlers.ofString());
        return response;
    }
```

Following is the Code in ProductService.java that is AI generated,

```
private static void createProductsTable(Connection connection) throws
SQLException {
        try (Statement stmt = connection.createStatement()) {
            // SQL query to create the "products" table with the specified schema
            String createTableQuery = "CREATE TABLE products (" +
                    "id INTEGER PRIMARY KEY," +
                    "name TEXT NOT NULL," +
                    "description TEXT," +
                    "price DOUBLE NOT NULL," +
                    "quantity INTEGER NOT NULL" +
                    ")";
            stmt.executeUpdate(createTableQuery);
        }
    }

    private static boolean tableExists(Connection connection, String tableName)
throws SQLException {
        try (Statement stmt = connection.createStatement()) {
            // Query to check if the table exists in the database
            String query = "SELECT name FROM sqlite_master WHERE type='table' AND
name='" + tableName + "'";
            return stmt.executeQuery(query).next();
        }
    }
private static boolean doesProductIdExist(int productId) {
        String query = "SELECT COUNT(*) FROM products WHERE id = ?";

        try (Connection connection = DriverManager.getConnection(url);
             PreparedStatement preparedStatement =
connection.prepareStatement(query)) {
```

```
            preparedStatement.setInt(1, productId);

            try (ResultSet resultSet = preparedStatement.executeQuery()) {
                if (resultSet.next()) {
                    int count = resultSet.getInt(1);
                    return count > 0; // If count is greater than 0, the ID
exists
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return false; // An error occurred or the ID was not found
    }
private static int createProduct(String url, int id, String name, String
description, Double price, int quantity) {
        try (Connection connection = DriverManager.getConnection(url)) {
            String insertQuery = "INSERT INTO products (id, name, description,
price, quantity) VALUES (?, ?, ?, ?, ?)";
            try (PreparedStatement pstmt =
connection.prepareStatement(insertQuery)) {
                pstmt.setInt(1, id);
                pstmt.setString(2, name);
                pstmt.setString(3, description);
                pstmt.setDouble(4, price);
                pstmt.setInt(5, quantity);
                // System.out.println("Product created successfully.");
                int res = pstmt.executeUpdate();
                if (res == 1) {
                    return 200;
                } else {
                    return 400;
                }

            }
        }  catch (SQLIntegrityConstraintViolationException e) {
            // Handle unique constraint violation (product with the same ID
already exists)
```

```
            e.printStackTrace();
            return 409; // Conflict status code
        } catch (SQLException e) {
            // Handle other SQL exceptions
            e.printStackTrace();
            return 500; // Internal Server Error status code
        } catch (NumberFormatException e) {
            // Handle number format exception
            e.printStackTrace();
            return 400; // Bad Request status code
        } catch (Exception e) {
            // Handle number format exception
            e.printStackTrace();
            return 400; // Bad Request status code
        }

    }

    private static int deleteProduct(String url, int id) {
        try (Connection connection = DriverManager.getConnection(url)) {
            String deleteQuery = "DELETE FROM products WHERE id = ?";
            try (PreparedStatement pstmt =
connection.prepareStatement(deleteQuery)) {
                pstmt.setInt(1, id);
                int res = pstmt.executeUpdate();

                if (res > 0) {
                    // Product deleted successfully
                    return 200; // OK status code
                } else {
                    // No product found with the given ID
                    return 404; // Not Found status code
                }
            }
        } catch (SQLIntegrityConstraintViolationException e) {
            // Handle unique constraint violation or other integrity constraints
            e.printStackTrace();
            return 409; // Conflict status code
        } catch (SQLException e) {
```

```
            // Handle other SQL exceptions
            e.printStackTrace();
            return 500; // Internal Server Error status code
        } catch (NumberFormatException e) {
            // Handle number format exception
            e.printStackTrace();
            return 400; // Bad Request status code
        }
    }

    private static int updateProduct(int id, Optional<String> name,
Optional<String> description, Optional<Double> price, Optional<Integer> quantity)
{
        try (Connection connection = DriverManager.getConnection(url)) {
            StringBuilder updateQuery = new StringBuilder("UPDATE products SET
");
            ArrayList<Object> values = new ArrayList<>();

            if (name.get() != null) {
                updateQuery.append("name = ?, ");
                values.add(name.get());
            }

            if (description.get() != null) {
                updateQuery.append("description = ?, ");
                values.add(description.get());
            }

            if (price.get() != null) {
                updateQuery.append("price = ?, ");
                values.add(price.get());
            }

            if (quantity.get() != null) {
                updateQuery.append("quantity = ?, ");
                values.add(quantity.get());
            }

            // Remove the trailing comma and space
```

```
            updateQuery.delete(updateQuery.length() - 2, updateQuery.length());
            updateQuery.append(" WHERE id = ?");

            try (PreparedStatement pstmt =
connection.prepareStatement(updateQuery.toString())) {
                int index = 1;

                for (Object value : values) {
                    pstmt.setObject(index++, value);
                }

                pstmt.setInt(index, id);

                int res = pstmt.executeUpdate();

                if (res > 0) {
                    // Product updated successfully
                    return 200; // OK status code
                } else {
                    // No product found with the given ID
                    return 404; // Not Found status code
                }
            }
        } catch (SQLIntegrityConstraintViolationException e) {
            // Handle unique constraint violation or other integrity constraints
            e.printStackTrace();
            return 409; // Conflict status code
        } catch (SQLException e) {
            // Handle other SQL exceptions
            e.printStackTrace();
            return 500; // Internal Server Error status code
        } catch (NumberFormatException e) {
            // Handle number format exception
            e.printStackTrace();
            return 400; // Bad Request status code
        }
    }
```

For UserService.java, it is a copy of productService.java with names tuned for userService needs.

For the workload_parser.py and iscs.py, AI was used as a google replacement with little to no copy pasted code from chatgpt.

In general, the use of generative AI assisted in my learning and in the sense of its much quicker than finding answers on google. Once I got past the learning curve needed for this assignment, I started using chatgpt as a time saver tool as most of the coding in this assignment was error handling and incredibly tedious. However this didn't affect my learning as I still had to understand the code and tweak the code itself or prompts to get it to work to my liking.

- as well as a discussion of where you took shortcuts and submitted a "this will do for a1, but won't scale and needs a rewrite for a2 but time is precious and it will likely pass test cases"

I started to run out of time so when I copied ProductService.java to UserService.java, I only changed the critical parts leaving the function names the same in most cases. When it comes to code that I don’t believe will scale, the orderService currently when handling an order request crafts a get to both user and product services, then a post request to product service, this assumes that the only post request being received at /order endpoint will be an order request. I'm assuming there will be more /order endpoint requests we will have to handle and thus this section will need a rewrite and is in fact just fine for my needs in A1. Another area that will definitely need a rewrite is the iscs. This is because it just forwards the requests from the orderService to the /user and /product endpoints. As the professor said, we will not be tested on if the iscs even exists but it is nice to have to make sure I won't have to deal with re-routing the orderservice later on. I actually underestimated my time for this assignment, so I did not have much time for testing. Thus I will likely have to have rewrites in A2 assuming that what I have is not sufficient.

- as well as which pieces of code you think will NOT require major rewrites for a2, include a WHY you believe this, justify your statements..

As mentioned, the orderservice just sends all its requests to the iscs and the iscs just forwards them to the /user and /product endpoints. This means that unless there are new requirements for the order service, everything in the architecture.png is currently in place and thus I won't have to reroute any request traffic. Another area that will not require major rewrites is the workload_parser.py since I wrote it with extensibility in mind. If new commands need to be parsed, then I just have to define their handling function in the API_COMMANDS dictionary I defined.