**PROJECT SPECIFICATION**

**Serverless Application**

1) Functionality -- DONE

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| The application allows users to create, update, delete TODO items  DONE | A user of the web application can use the interface to create, delete and complete a TODO item. |
| The application allows users to upload a file.  DONE- Student note: generateUrl service implemented. Service returns a signed url for file upload and update todo item and adds attechmentUrl which is diffirent from generated url. | A user of the web interface can click on a "pencil" button, then select and upload a file. A file should appear in the list of TODO items on the home page. |
| The application only displays TODO items for a logged in user.  DONE - Student note:Table,all services and queries are designed according to userId. System gets UserId information from token. | If you log out from a current user and log in as a different user, the application should not show TODO items created by the first account. |
| Authentication is implemented and does not allow unauthenticated access.  DONE - Student note:oauth integration completed. | A user needs to authenticate in order to use an application. |

2) Code Base --DONE

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| The code is split into multiple layers separating business logic from I/O related code.  DONE - Student note: database access located in dataLayer folder, business logic codes located in bussinessLogic folder, file storege codes are located in fileStorage folder | Code of Lambda functions is split into multiple files/classes. The business logic of an application is separated from code for database access, file storage, and code related to AWS Lambda. |
| Code is implemented using async/await and Promises without using callbacks.  DONE | To get results of asynchronous operations, a student is using async/await constructs instead of passing callbacks. |

3) Best Practices-- DONE

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| All resources in the application are defined in the "serverless.yml" file  DONE | All resources needed by an application are defined in the "serverless.yml". A developer does not need to create them manually using AWS console. |
| Each function has its own set of permissions.  DONE | Instead of defining all permissions under **provider/iamRoleStatements**, permissions are defined per function in the **functions** section of the "serverless.yml". |
| Application has sufficient monitoring.  DONE Student note:loging has been made. | Application has at least some of the following:   * Distributed tracing is enabled * It has a sufficient amount of log statements * It generates application level metrics |
| HTTP requests are validated  DONE - Student note: validation applied for create todo and update todo | Incoming HTTP requests are validated either in Lambda handlers or using request validation in API Gateway. The latter can be done either using the **serverless-reqvalidator-plugin** or by providing request schemas in function definitions. |

4) Architecture -- DONE

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| Data is stored in a table with a composite key.  DONE | 1:M (1 to many) relationship between users and TODO items is modeled using a DynamoDB table that has a composite key with both partition and sort keys. Should be defined similar to this:  KeySchema:  - AttributeName: partitionKey  KeyType: HASH  - AttributeName: sortKey  KeyType: RANGE |
| Scan operation is not used to read data from a database.  DONE | TODO items are fetched using the "query()" method and not "scan()" method (which is less efficient on large datasets) |

**Suggestions to Make Your Project Stand Out!**

1. Fetch a certificate from Auth0 instead of hard coding it in an authorizer. **Student note: not implemented**
2. Implement pagination support to work around a DynamoDB limitation that allows up to 1MB of data using a query method. **Student note: not implemented**
3. Add your own domain name to the service. **Student note: not implemented**
4. Add an ability to sort TODOs by due date or priority (this will require adding new indexes). **Student note: not implemented**
5. Implement a new endpoint that allows sending full-text search requests to Elasticsearch (this would require copying data from DynamoDB to Elasticsearch as we did in lesson 4). **Student note: not implemented**