**PROJECT SPECIFICATION**

**Cloud Capstone Project**

**Option 2 implemented for capstone project**

(Option 1): CI/CD, Github & Code Quality. **Student Note: Not Implemented**

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| The project demonstrates an understanding of CI and Github. | All project code is stored in a GitHub repository and a link to the repository has been provided for reviewers. The student uses a CI tool to build the application |
| The project has a proper documentation. | The README file includes introduction how to setup and deploy the project. It explains the main building blocks and has comments in the important files. |
| The project use continuous deployments (CD) | A CD tool is in place to deploy new version of the app automatically to production. The way is described and easy to follow. |

(Option 1): Container **Student Note: Not Implemented**

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| The app is containerized | There is a Dockerfile in repo and the docker image can be build |
| The project have public docker images | On DockerHub images for the application are available |
| The applications runs in a container without errors | Starting the app as a container on a local system |

(Option 1): Deployment **Student Note: Not Implemented**

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| The application runs on a cluster in the cloud | The project can be deployed to a kubernetes cluster |
| The app can be upgraded via rolling-update | The students can deploy a new version of the application without downtime |
| A/B deployment of the application | Two versions of the same app can run at the same and service traffic |
| Monitoring | The application is monitored by Amazon CloudWatch |

(Option 2): Functionality **Student Note: DONE**

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| The application allows users to create, update, delete items  **Student Note: DONE** | A user of the web application can use the interface to create, delete and complete an item. |
| The application allows users to upload a file.  **Student Note: DONE** | 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 items on the home page. |
| The application only displays items for a logged in user.  **Student Note: DONE** | If you log out from a current user and log in as a different user, the application should not show items created by the first account. |
| Authentication is implemented and does not allow unauthenticated access.  **Student Note: DONE** | A user needs to authenticate in order to use an application. |

(Option 2):Codebase **Student Note: DONE**

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| The code is split into multiple layers separating business logic from I/O related code.  **Student Note: DONE** | 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.  **Student Note: DONE** | To get results of asynchronous operations, a student is using async/await constructs instead of passing callbacks. |

(Option 2):Best practices **Student Note: DONE**

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| All resources in the application are defined in the "serverless.yml" file  **Student Note: 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.  **Student Note: 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.  **Student Note: DONE** | 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  **Student Note: DONE** | 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. |

(Option 2):Architecture **Student Note: DONE**

| CRITERIA | MEETS SPECIFICATIONS |
| --- | --- |
| Data is stored in a table with a composite key.  **Student Note: DONE** | 1:M (1 to many) relationship between users and 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.  **Student Note: DONE** | Items are fetched using the "query()" method and not "scan()" method (which is less efficient on large datasets) |