**Anonymous Builders Client/Job Data Application**

**Design Document**

**Introduction:** Anonymous Builders (AB), provider of residential construction services, seeks a complete replacement of their existing data system, used to record details of customers and the jobs associated with them. The company currently relies on a process that is initiated by hand-filled paper forms and then transcribed into electronic spreadsheets. AB’s objective is to eliminate the need for any paper documentation in the initial job assessment process, secure storage of all customer, job, and job site photo data off site from the office and make data accessible only to authorized personnel.

Other primary requirements are that the interface must be easily utilized by non-technical personnel and that the software must be compatible with existing PC and mobile device browsers (Firefox and Chrome, specifically). Users will fall into three categories: Estimators, Clerical staff, and Administrator. Time constraint is the completion of the project within 16 weeks of the start date.

**Architecture:** This project will take the form of a service-based web application that will run in the user’s browser. The design architecture will be a combination of client-server model and Model-View-Controller (MVC) pattern; below is a textual description of the design followed by diagram representations:

The GUI View presented to the end users, primarily the Estimators and Clerical staff, will be a series of HTML pages: a Login page, a New Client page, a New Job page, and a Query page. The functionality and aesthetics of the pages will be supported by JavaScript code in tandem with the jQuery library and Bootstrap framework. The pages will allow authorized users the ability to save and retrieve client and jobs records to and from a remote database. Due to the size and format of the files, job site photographs will be sent to an AWS S3 Bucket for storage instead of to the database.

This is accomplished by connecting the client to an Amazon Web Services (AWS) cloud server. The entire backend of the application consists of a series of AWS services we will configure to share data, functions, and authorizations amongst one another. First, an AWS DynamoDB database, featuring a NoSQL, non-relational structure, will be created to store the data for both clients and jobs. The AWS Lambda service, configured for Node.js runtime, will host custom JavaScript functions we will write to provide the means of moving data to and from the database.

The bridge between this data and the front-end GUI will be the AWS API Gateway service. The API configured here will feature the endpoints for the browser to send GET, POST, PATCH, and DELETE HTTP requests, which will be passed through to the corresponding Lambda functions for execution. This means that API Gateway and Lambda combined effectively form the data Model representing “Clients” and “Jobs” while also serving as the Controller that handles their access and modification.

Finally, user login and authentication for Estimators and Clerical staff will be processed by the AWS Cognito service. We will create a separate User Pool for each user category, to which we will attached IAM Policies tailored to their specific duties. For AB’s Administrator user, our team will simply turn over ownership of the AWS account and its login credentials upon the settled completion of our project contract.

It should be noted these services are all based on AWS’s “serverless” technology, executing Lamdba functions and rendering responses to service calls only as needed, thus keeping virtual infrastructure costs to a minimum for AB.

*Client-Server Architecture Diagram*

AWS Cloud Servers

Web Browser GUI

REQUEST

*Model-View-Controller Architecture Diagram*

**SERVER**

**CLIENT**

RESPONSE

**CONTROLLER**

API Gateway  
& Lambda functions control data transfer  
to and from database

**MODEL**

**VIEW**

DynamoDB records for clients and jobs are modeled into usable data objects  
by mapping configured in API Gateway

Input interface and visual display provided by web pages in user’s browser

*AWS Services Flow Diagram*

**DYNAMODB**

**IAM POLICIES**

**USER POOL**

Stores tables that hold client and job records

Defines database operations permitted

Contains lists of users in separate categories

Authenticates users against defined pool, granting or denying access to API endpoints

**GUI**

Executes custom functions that move data in and out of the database

Send requests and display responses

Defines methods for accessing data and the mapping that defines how data will be modeled

**COGNITO**

**LAMBDA**

**API GATEWAY**

**Use Cases:** The following is a list of use cases describing fictional but representative users of the application. They include members of each user pool category and describe situations relevant to the requirements referenced in the Requirements Document.

1. Bob is a seasoned job site Estimator for AB. Having previously worked as a job foreman for years, he is an expert at assessing the requirements for a given construction project and also has great people skills that put potential customers at ease. However, he has, on occasion, misplaced customer/job forms or procrastinated when typing them up back at the office to submit to Clerical staff. With our application, Bob can simply arrive at a site, use his Android phone to enter and upload client/job data and photographs while he conducts his estimate, then move directly on to the next site. This eliminates the chance of lost or idle paper documents, while saving him time in his work day by making trips back to the office less necessary.
2. Jane is also an Estimator. She became nervous when hearing that AB was planning a switch to a digital system. She grew up without frequent access to a computer and is unsure of herself on the occasions when she has to use PC software. She also detests reading any kind of technical documentation. AB management became aware of these concerns and relayed them to our team. Upon delivery of the completed application, we held brief training with AB’s employees. Jane, who has been an iPhone user for years and is quite savvy with the device, realized that our application was no different than many of the native and web apps she has been using daily. After AB provided her with a new Android tablet, she was quickly able to use our simply application with ease.
3. Miguel is a member of the Clerical staff at AB’s office building. One of his primary duties is receiving client/job data forms from the Estimators, submitted to him via email in the form of Word documents. A considerable portion of his time is spent sifting through his email inbox, separating out new form email from old ones needing to be deleted, downloading the attached documents to his workstation, then manually entering the data into an Excel spreadsheet that holds a record of all the company’s clients and jobs. Over time, this already tedious process has become more cumbersome due to the size of the spreadsheet and the need to group jobs with repeat clients. Mistakes, including duplicate records and overwritten records, are common. Miguel is delighted by the new application since it automatically sorts and associates all new job records as soon as they are entered by Estimators in the field, reducing his task list and keeping his inbox far less cluttered.
4. Kacey is AB’s office manager. She receives a call at home one evening; it is AB’s owner and CEO informing her that a kitchen fire engulfed the restaurant next door to the AB office and spread before firefighters could stop the blaze. A large portion of AB’s office was destroyed as a result, including the destruction of all their computer equipment due to the extreme heat. Formerly, this would have created an IT disaster for the company, but thanks to our software, Kacey knew they could quickly bounce back. While awaiting insurance payout, Kacey was authorized to use company funds to purchase new workstations and rent temporary office space. Due to the entire application and database being stored in the cloud, Kacey was able to immediately get the office staff back to work with all their client/job records intact.

**UML:** The following is a UML representation of the two classes and their relationship that will be used in the OOP design of our software. Essentially, client data input and job data input by the user will be stored as instantiations of the “Client” class and “Job” class respectively. Each client object will have a one-to-many relationship with one of more objects jobs. These objects will pass through the application sequence (described above in Architecture) to eventually be stored as database records in their respective tables.

Diagram

Description automatically generated

**Test Cases:** The following functions must be tested to ensure the application meets all project requirements:

1. Test that user accounts of both Estimator and Clerical pools are able to successfully login to the application browser GUI.
2. Test that usernames not added to the pools by the AWS administrator cannot login to the application.
3. Test that Estimator user account type is able to successfully create new client and job records as well as retrieve and alter existing client and job records. Ensure that data inserted in the GUI matches exactly the resulting information that appears in the database.
4. Test to ensure that duplicate clients and jobs cannot be created.
5. Repeat the above test for Clerical user account type. Furthermore, ensure that this account type and ONLY this account type is able to successfully delete client and job records.
6. Test that the application exhibits proper validation of input data and correctly handles incompatible input.
7. Test that retrieval of existing records returns the correct record queried.
8. Test that input of a new job record is correctly associated with the desired client record.
9. Test that deletion of a client record also deletes all associated job records.
10. Conduct usage tests with actual end users selected by AB management to ensure these target users are able to operate the application successfully.
11. All previous tests must be conducted in both Firefox and Chrome browsers on both PC workstations and Android mobile devices.

The following Test Cases will validate the functionality outlined above. Note that some Test Cases will serve to meet the testing requirement for multiple benchmarks.

Links to Test Case Documents (included in parent folder):

[TC#1:Login](Test_Case_1.doc)

[TC#2:Invalid Login](Test_Case_2.doc)

[TC#3: Create New Client](Test_Case_3.doc)

[TC#4: Find Client & Jobs](Test_Case_4.doc)

[TC#5: Create New Job](Test_Case_5.doc)

[TC#6: Update Client](Test_Case_6.doc)

[TC#7: Update Job](Test_Case_7.doc)

[TC#8: Delete Client](Test_Case_8.doc)

[TC#9: Delete Job](Test_Case_9.doc)

**Summary:** The application, if designed and successfully tested as described in this document, will meet all the requirements specified by Anonymous Builders. The use of established and straightforward design architecture will simplify development and promote ease of maintenance. Furthermore, our choice of Amazon Web Services for hosting and backend functionality will greatly reduce the workload on our team, ensuring that we are able to deliver a robust suite of features in time to meet the project deadline. Likewise, our use of Bootstrap framework will allow us to produce a GUI that looks professional and aesthetically acceptable without drawing too much of our team’s focus away from implementing necessary functionality. Our expected result is to deliver, on time, a product that is secure, easy-to-use, and high-quality, while necessitating a relatively low ongoing cost to our client.