**High level Architecture**

**P05:HR MANAGEMENT SYSTEM**

**<team member names & ids>**

|  |  |
| --- | --- |
| **Student ID** | **Name** |
| **22100289** | **Mohammad Yousuf** |
| **22100260** | **Talha Nasir** |
| **22100230** | **Javeria Tariq** |
| **22100303** | **Ali Adnan Arif** |
| **22100062** | **Aamina Mariam** |

|  |  |  |
| --- | --- | --- |
| **Content** | **Totals** | **Obtained** |
| Architecture diagram | 30 | 8 |
| Architecture description | 10 | 10 |
| Architecture justification | 15 | 12 |
| Risk Management | 20 | 16 |
| Tools & Technologies | 10 | 10 |
| Hardware Requirements | 5 | 5 |
| Who did what | 5 | 5 |
| Review checklist | 2 | 2 |
| Overall formatting/template | 3 | 3 |
| Late submission penalty | -20 |  |
| **Total** | **100** | **71** |
| Review | 20 |  |
| **Grand Total** |  |  |

**Table of Contents**

[*1. Introduction 3*](#_3znysh7)

*[2. System Architecture 4](#_2et92p0)*

[*2.1 Architecture Diagram*](#_tyjcwt)

[*2.2 Architecture Description*](#_3dy6vkm)

*2.2.1 Web Application Servers*

*2.2.2 Database*

#### 2.2.3 Data Lake

#### 2.2.4 DNS - Domain Name System

#### 2.2.5 Load Balancer

[*2.3 Justification of the Architecture*](#_17dp8vu) *5*

*2.3.1 Why we chose this architecture*

*2.3.2 Pros and Cons*

*[3. Risk Management 7](#_3rdcrjn)*

[*3.1 Potential Risks and Mitigation Strategies*](#_26in1rg)

[*4. Tools and Technologies*](#_lnxbz9) *10*

[*5. Hardware Requirements*](#_35nkun2) *10*

[*6. Who Did What? 1*](#_1ksv4uv)*1*

[*7. Review checklist*](#_4i7ojhp) 11

# Introduction

The Human resource management system covers many HR aspects from application to management to promotions. The software keeps track of an organization’s employees and provides analytics of their performance using relevant KPIs. The software combines a number of systems and processes to ensure the ease of management in human resources and business processes. The HRMS software helps HR professionals manage the modern workforce.

Our aim is to assist companies in running effectively and efficiently. The system is a suite of software that companies can use to regulate their internal HR functions. Employee data management, recruitment, benefits, training, talent management, employee engagement, and employee attendance include some of the features our software will provide.

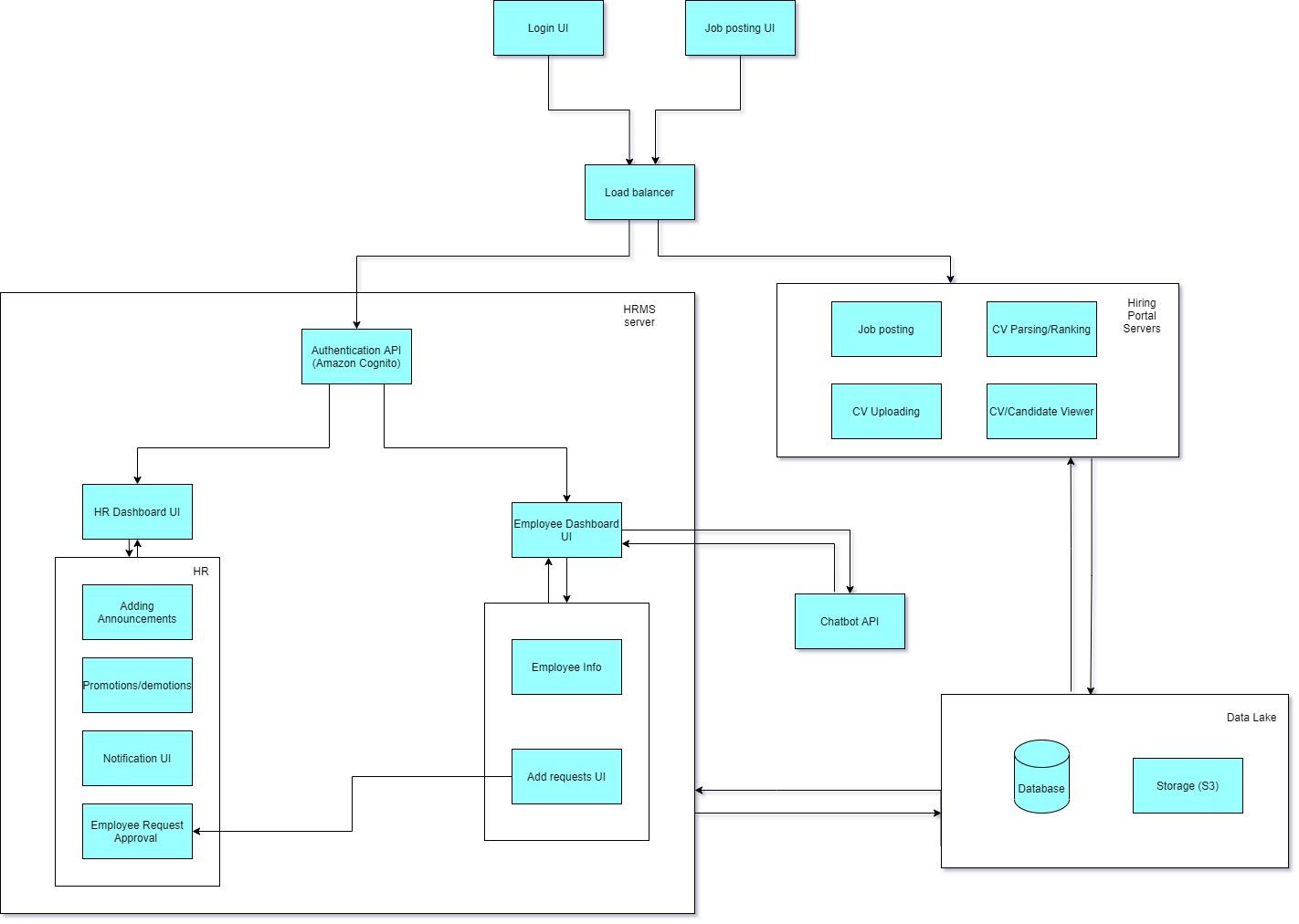
# System Architecture

## Architecture Diagram

[As discussed in the class, this is not an architecture diagram. Please look at architectural patterns and update it. You have discussed justification of client server and layered architecture but did not draw the diagram properly. ]

# 

Updated Architecture Diagram:



## Architecture Description

**2.2.1 HRMS Server**

This subsystem processes the user’s requests (the server connected to the load balancer is used to process the user’s requests.) and sends all of the required documents (based on the request) back to their browser. The required documents will usually consist of files such as JSON and XML. The way this subsystem works is that it calls the back-end infrastructures such as the database, etc. The server will be hosted on Amazon Web Services (AWS).

#### **2.2.2 Database**

The database subsystem contains information to be used for performing computations. The web application server can interact with the database to complete the information updates of the whole system such as adding, deleting, searching and organizing user data.

#### **2.2.3 Data Lake**

A data lake is a centralized repository that allows you to store all your structured and unstructured data. Since our system consists of data from two sources: employees and candidates, the data lake will act as a central repository for both of them. The application will interact with the data lake to provide analytics dashboards and for searching/querying data.

**2.2.4 Load Balancer**

The Load Balancer subsystem directs incoming requests to one of the multiple servers. After this, the load balancer sends an answer to a user. The load balancer distributes tasks among the servers to make sure that they do not get overcharged.

**2.2.5 Login (Amazon Cognito)**

This module will provide secure user authentication for the web application that will allow a user to login/signup.

**2.2.6 AI Chatbot (Amazon Lex)**

The AI Chatbot is a chatbot service designed to assist the user. The user interacts with the chatbot by clicking on the chat functionality and typing a query and the chatbot will respond by generating an automated response based on the user’s query.

**2.2.7 Cloud Storage(S3)**

S3 is a storage container for storing the CVs and resumes of applicants. When a job applicant uploads their resume/CV it gets stored in the S3 container.

## Justification of the Architecture

We have chosen a hybrid between client server architecture and layered architecture. This architecture is specifically useful for our application since we require a separation of concerns between the client and the server along with the layered architecture separating the system functionality into layers. Furthermore, this also allows the candidate applying for a job to add their CV/resume without having to login to the internal HR portal of the company. This architecture is best suited for our application as it can cater to a large number of users without being overwhelmed and store and process data according to the requirements of the system. This also allows our system to scale both horizontally and vertically while ensuring the modularity of the system. Our architecture also ensures the security of sensitive user data like passwords by using Amazon Cognito. Cognito will also ensure proper authorization. It controls permissions for different user groups in our web app. This ensures that users have appropriate access to backend resources, determined by the group they belong to.

Moreover, the architecture uses a combination of server and serverless protocols. This strategy optimizes performance and cost of operating the system while ensuring that there is no single point of failure. The use of serverless protocols for handling data intensive workloads (like AI chatbot and search queries) ensures efficient utilization of resources while providing fast and reliable performance. We have decided to use a DynamoDB as our database as it provides a serverless, key-value NoSQL database for high-performance applications at scale. DynamoDB provides an efficient and scalable solution for building ad-hoc and on-demand dashboards required by our system.

Lastly, we went with having an architecture with multiple servers and only one database. This model doesn’t store any data. When the client sends information to the web server, it is processed and written to the database, but managing this data takes place outside of the server. It’s called stateless architecture. It’s important to make our model reliable because if one server goes down, another one will take charge. So, in such a failure, all the requests will automatically go to the new server, without affecting the systems functioning. Thus, this model is more reliable than a single server. However, if something happens to the database, the app will crash.

The hybrid approach we have used also makes the system architecture more complex. This added complexity makes the system and its various functionalities difficult for a new developer to understand without interacting with the system.

[Also discuss how would your system architecture ensure authorization.]

# Risk Management

## Potential Risks and Mitigation Strategies

[All the risks listed here are related to security and privacy of the system. However, we also talked about product, project and business risks. You should address some of them as well.]

|  |  |  |
| --- | --- | --- |
| **Sr.** | **Risk Description** | **Mitigation Strategy** |
|  | Server failure | There are two ways to fix this issue: to store data in all the accessible databases or distribute it evenly among them |
|  | Broken Authentication | Configure multi-factor authentication whenever possible. The more hoops an attacker has to jump, the harder it is to get into your system. Also paying attention to session management and setting application timeouts correctly. When a user closes a browser, log them out of the system. Any time a user leaves a session while still logged in, the entire system remains vulnerable. |
|  | Sensitive data exposure | Sensitive data exposure can be prevented by   * Encrypting data in transit and at rest. * Using the latest encryption algorithms. * Disabling auto-complete on forms that collect data. * Disabling caching on forms that collect data. |
| 4 | Broken Access Control | It is important to control who has access to which portion of the website. Access control is only effective if it is enforced in trusted server-side code, where the attacker cannot modify the access control authentication.   * Deny access to standard functions. * Use access control lists and role-based authentication mechanisms. * Don't just hide features. |
| 5 | Security Misconfiguration | Security misconfiguration can be mitigated by not allowing default configuration in our website. It can be prevented by   * Disabling administration interfaces * Disabling use of default accounts/passwords. * Configuring the server to prevent unauthorized access, directory listing, etc. * Considering periodically performing scans and audits to help detect future configuration errors or missing fixes. |
| 6. | Cross Site Scripting | In general, effective prevention of cross site scripting vulnerabilities can involve a combination of the following measures:   * **Sanitize User Input**  1. Validate to detect potentially malicious input from users. 2. Encrypt the output to prevent potentially malicious user-supplied data from triggering the autoload and execute behavior by a browser  * **Limit use of user provided data**  1. Only use when it’s necessary 2. Utilize content security policy   By providing additional layers of protection against cross site scripting attacks. |
| 7. | Insufficient logging and monitoring | Depending on the risk of the data being stored or processed by the application:   * Ensure that all server-side login, access control, and access validation errors can be logged with sufficient user context to identify suspicious or malicious accounts and for a sufficient period of time to allow for deferred forensic analysis. * Ensure that high-value transactions have an audit trail with integrity checks to prevent tampering or deletion, such as add-only database tables and the like. * Provide effective monitoring and alerts to detect and address suspicious activity in a timely manner. |
| 8. | Insecure Direct Object References | Perform proper and consistent user authorization and whitelist the choices. More often than not, however, the whole problem can be avoided by storing the data internally. |
| 9. | Unvalidated Redirects And Forwards | This can be prevented by   * Avoiding redirects * Providing a static list of valid locations to redirect to. |
| 10. | Unrestricted File Upload | * Never accept a filename and its extension directly without an allow list filter. * The application should filter and check content for all files uploaded to the server. Files must be carefully analyzed and validated before being made available to other users. When in doubt, the file should be discarded. * It is necessary to have a list of the only allowed extensions on the web application. And the file extension can be selected from the list. |
| 11. | Poor Management | * The project schedule is known to every team member and they should have access to the project Gantt chart. * Have regular meetings with the project manager to discuss progress. |
| 12. | Communication Issues | * Team members should be in touch with each other through text messaging * All members should be able to contact the project manager through a digital workspace(in our case it is Slack) |
| 13. | Team members(coders) leave | * All employees will follow the same coding practices either old or new. * All important pieces of code specifically functions will be properly commented for easy understanding of all members |

# Tools and Technologies

* Backend: Django 3.8.2 or 4.0
* Frontend: React 17.0
* Databases: Amazon DynamoDB orAmazon DocumentDB 3.6
* Chatbot services: AWS Lex V2
* Servers/Serverless: AWS EC2, AWS lambda

# Hardware Requirements

* Windows Server x64 OS
* Mac Os X 10.8

# 20 GB free hard disk space

# Network interface software for network communications, and TCP/IP network protocol

* 3 GHz CoreTM 2 Duo processor with 2 GB RAM (*minimum*)
* DVD-ROM drive

# 6. Who Did What?

|  |  |
| --- | --- |
| **Name of the Team Member** | **Tasks done** |
| **Mohammad Yousuf** | Architecture Diagram, Architecture description, Justification of the architecture, Tools and technologies |
| **Talha Nasir** | Architecture Diagram, Architecture description, Justification of the architecture, Formatting |
| **Javeria Tariq** | Hardware requirements & Risk management |
| **Ali Adnan Arif** | Architecture description, Introduction & Formatting |
| **Aamina Mariam** | Architecture Diagram & Risk management |

# Review checklist

|  |  |
| --- | --- |
| **Section** **Title** | **Reviewer Name(s)** |
| Section 2 | **Mohammad Yousuf** |
| Section 3 | **Talha Nasir** |
| Section 2 | **Javeria Tariq** |
| Section 4&5 | **Ali Adnan Arif** |
| Section 2 | **Aamina Mariam** |