WORK INTEGRATED LEARNING PROGRAMEE

SOFTWARE ARCHITECTURE

ASSIGNMENT 2

Submitted By,
Pavithra.S
2022MT93172

CLOUD REPORTING APPLICATION

CONTENTS

PURPOSE OF THE SYSTEM

- The cloud reporting application is to centrally store all the testing data collected from a touch device and to automate the process of data collection by integrating directly with the touch device.
- The touch device integrates with scanners that perform network tests and collects relevant data. This data is synced directly to the cloud data stores.
- Our cloud platform will serve as a single serving platform for multiple type of end users having access to the platform to view, upload, configure and export test reports.

KEY FUNCTIONAL REQUIREMENTS

Authentication

- The user should be able to login into the application using an email/password combination.
- Based on the credentials, the user type is to be identified, and the corresponding admin or company view is to be displayed.

Company Side

- Add/Update company users and respective permissions.
- Create or map building to the company.
- Upload building test reports.
- View and export the building test reports in pdf or word formats.
- Create/Update test plan configurations.

Admin Side

- The super admin should be able to manage other admin users and their permissions.
- Create/Update companies and map subscriptions to them which will the allow companies to manage building test data.
- Provide help to the companies through the support module.

ARCHITECTURALLY SIGNIFICANT REQUIREMENTS(ASR'S)

The top 3 architecturally significant requirements for the cloud reporting application are,

Performance

- The cloud reporting application is going to be the cental store for all test data from the web and the touch device.
- So, the high performance becomes an important requirement.

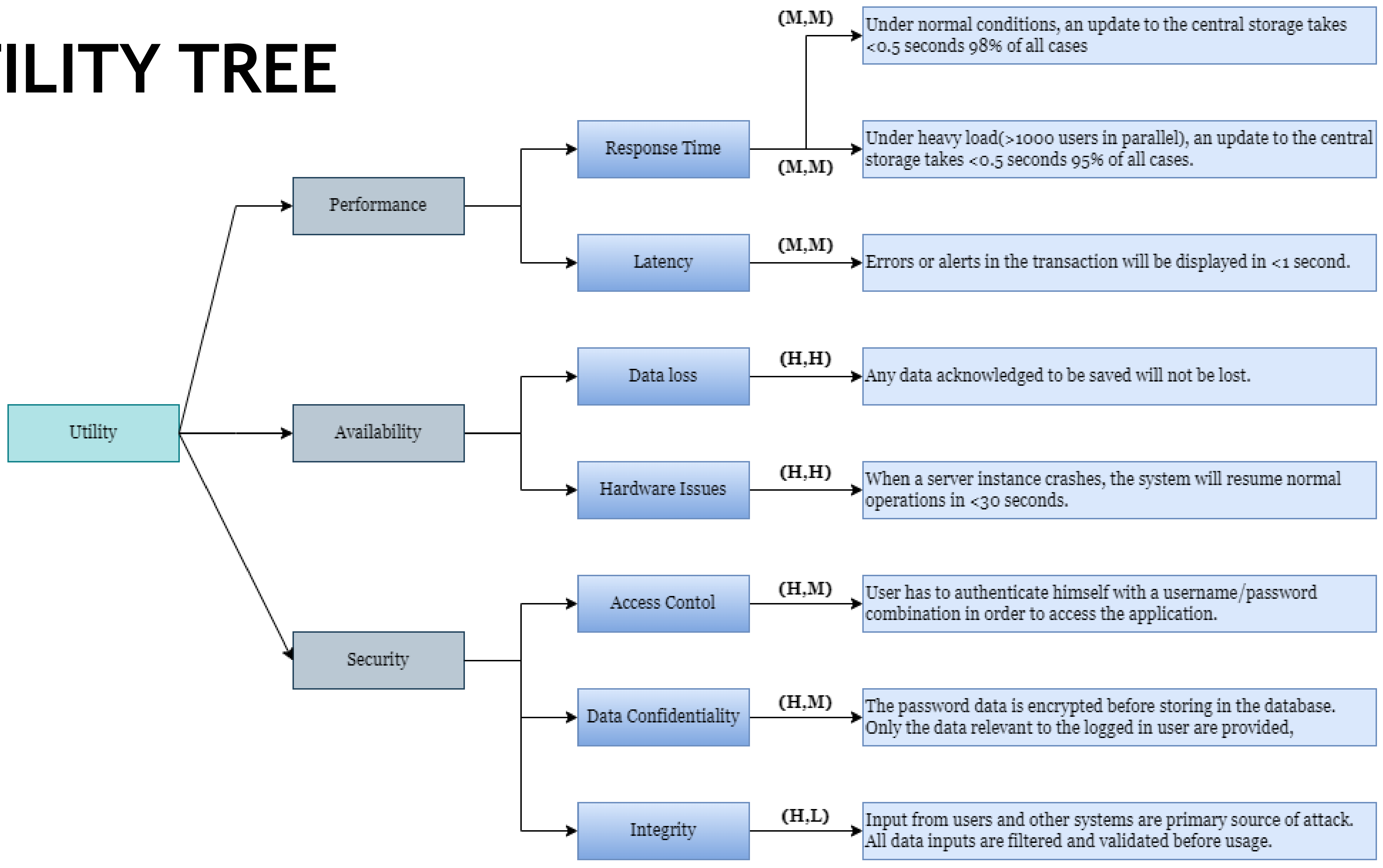
Availability

- The system should be available to share and update test data from different sources.
- High availability will make sure that the test data is not lost.

Security

- The test data files are going to be shared from various sources.
- So, the system should be secured enough to protect from attacks and security threats.

UTILITY TREE



Utility
Performance
Availability
Security
Response Time
Latency
Data loss
Hardware Issues
Access Contol
Data Confidentiality
Integrity
(M,M)
Under normal conditions, an update to the central storage takes <0.5 seconds 98% of all cases
(M,M)
Under heavy load(>1000 users in parallel), an update to the central storage takes <0.5 seconds 95% of all cases.
(M,M)
Errors or alerts in the transaction will be displayed in <1 second.
(H,H)
Any data acknowledged to be saved will not be lost.
(H,H)
When a server instance crashes, the system will resume normal operations in <30 seconds.
(H,M)
User has to authenticate himself with a username/password combination in order to access the application.
(H,M)
The password data is encrypted before storing in the database. Only the data relevant to the logged in user are provided,
(H,L)
Input from users and other systems are primary source of attack. All data inputs are filtered and validated before usage.

KEY: (Business Value, Architectural Value) -> High(H), Medium(M), Low(L)

TACTICS USED TO ACHIEVE TOP 3 ASR'S

Performance

- Caching – Caching the data from the database to avoid multiple calls.
- Avoid blocking calls and use asynchronous calls.
- The response time of the Api call is optimized as much as possible with an average of <0.5sec.

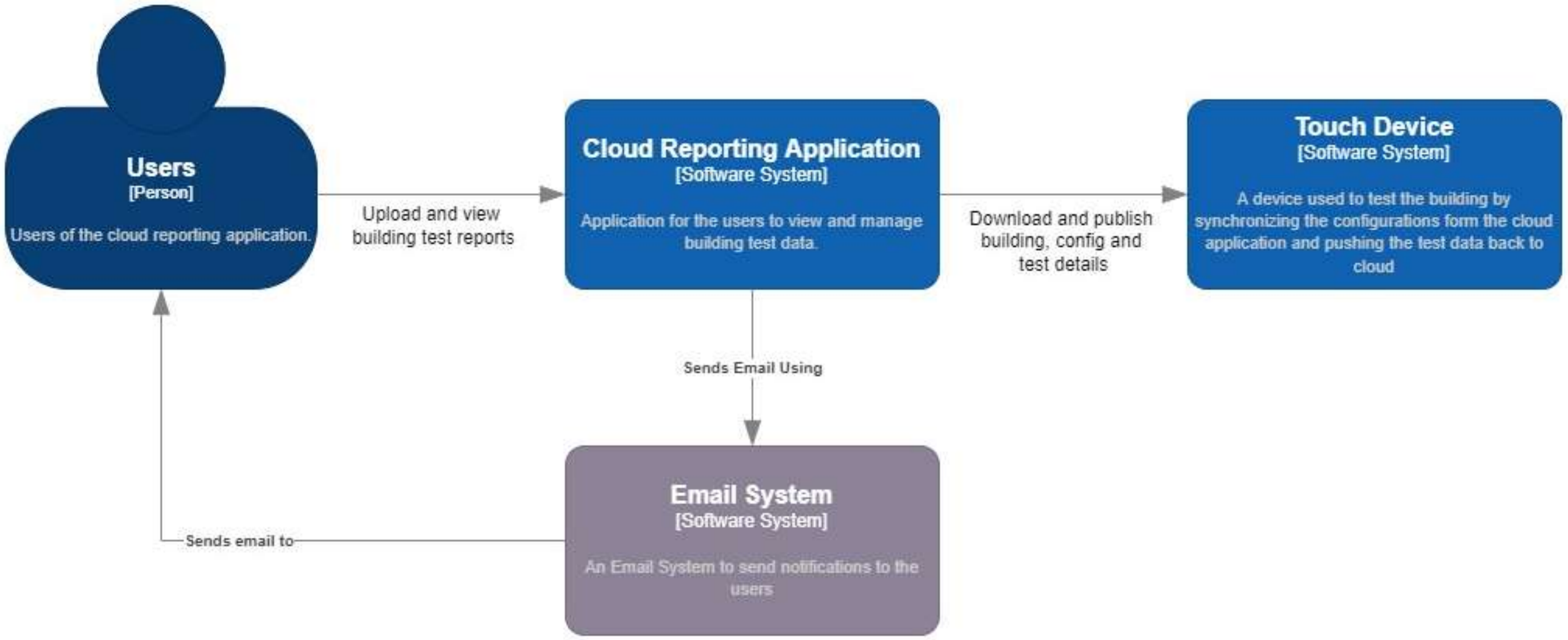
Availability

- Periodic health check to check the availability of the resources.
- Exception handling to handle unexpected errors and failures.
- Periodic backup of database.

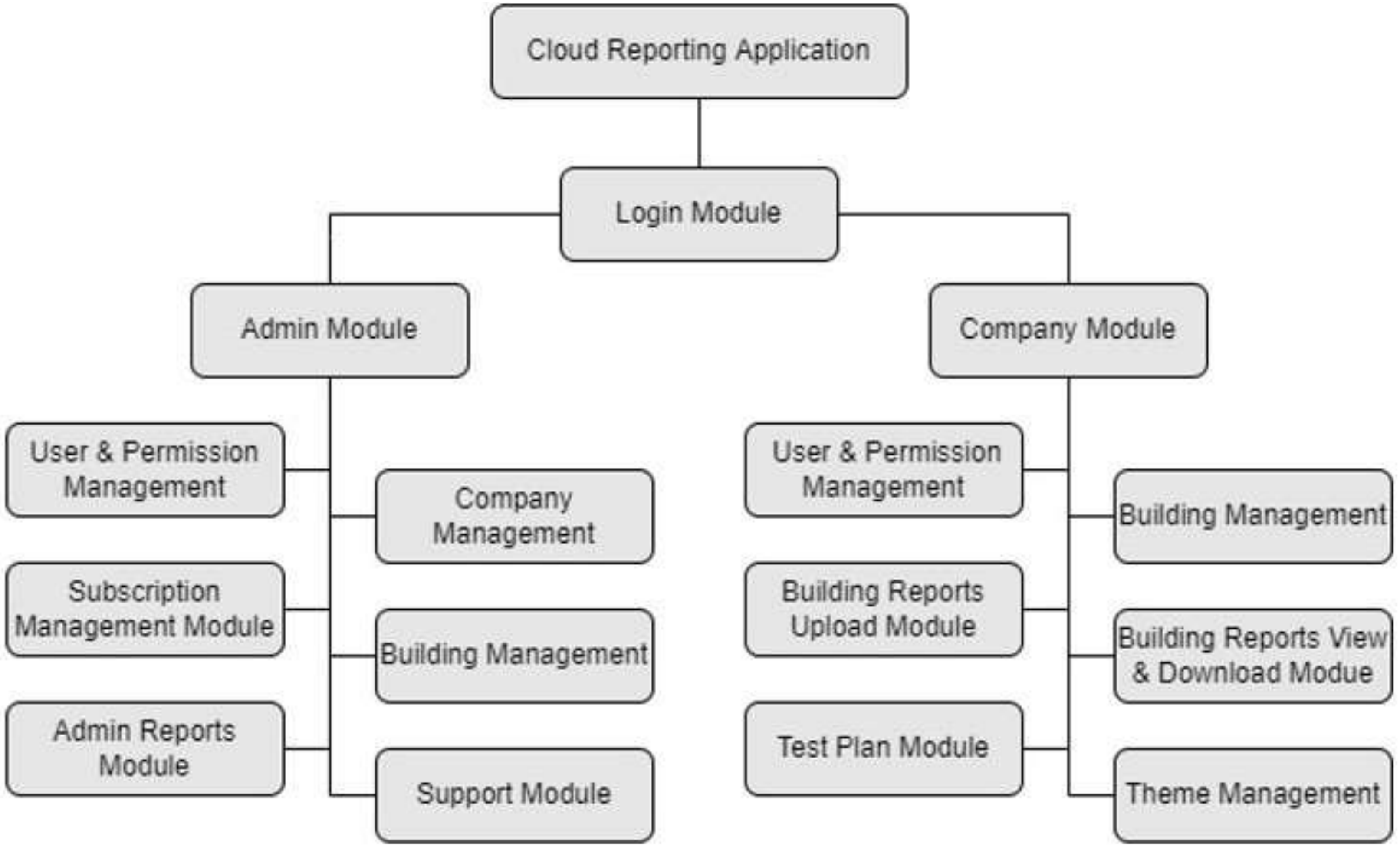
Security

- Authenticate the user before the user access the system and check authorization before each request. JWT token is to be sent with each request to validate the user.
- The resources will run with minimum privileges to user accounts according to least privilege, based on a “need to know” approach.
- All inputs from users and other sources are filtered and validated to avoid security attacks.

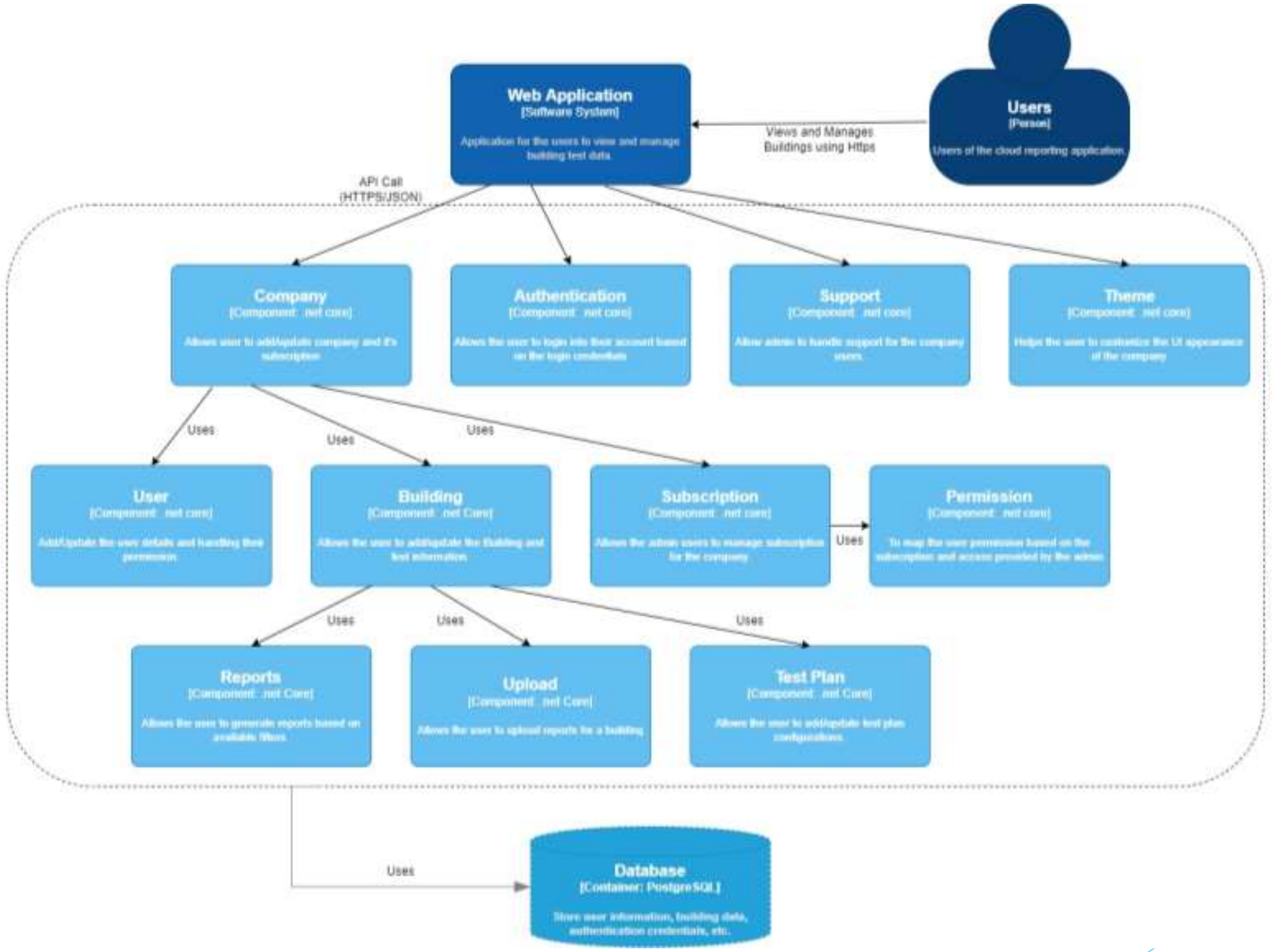
CONTEXT DIAGRAM


Users
[Person]
Users of the cloud reporting application.
Upload and view building test reports
Cloud Reporting Application
[Software System]
Application for the users to view and manage building test data.
Download and publish building, config and test details
Touch Device
[Software System]
A device used to test the building by synchronizing the configurations form the cloud application and pushing the test data back to cloud
Sends Email Using
Email System
[Software System]
An Email System to send notifications to the users
Sends email to

MODULE DECOMPOSITION DIAGRAM

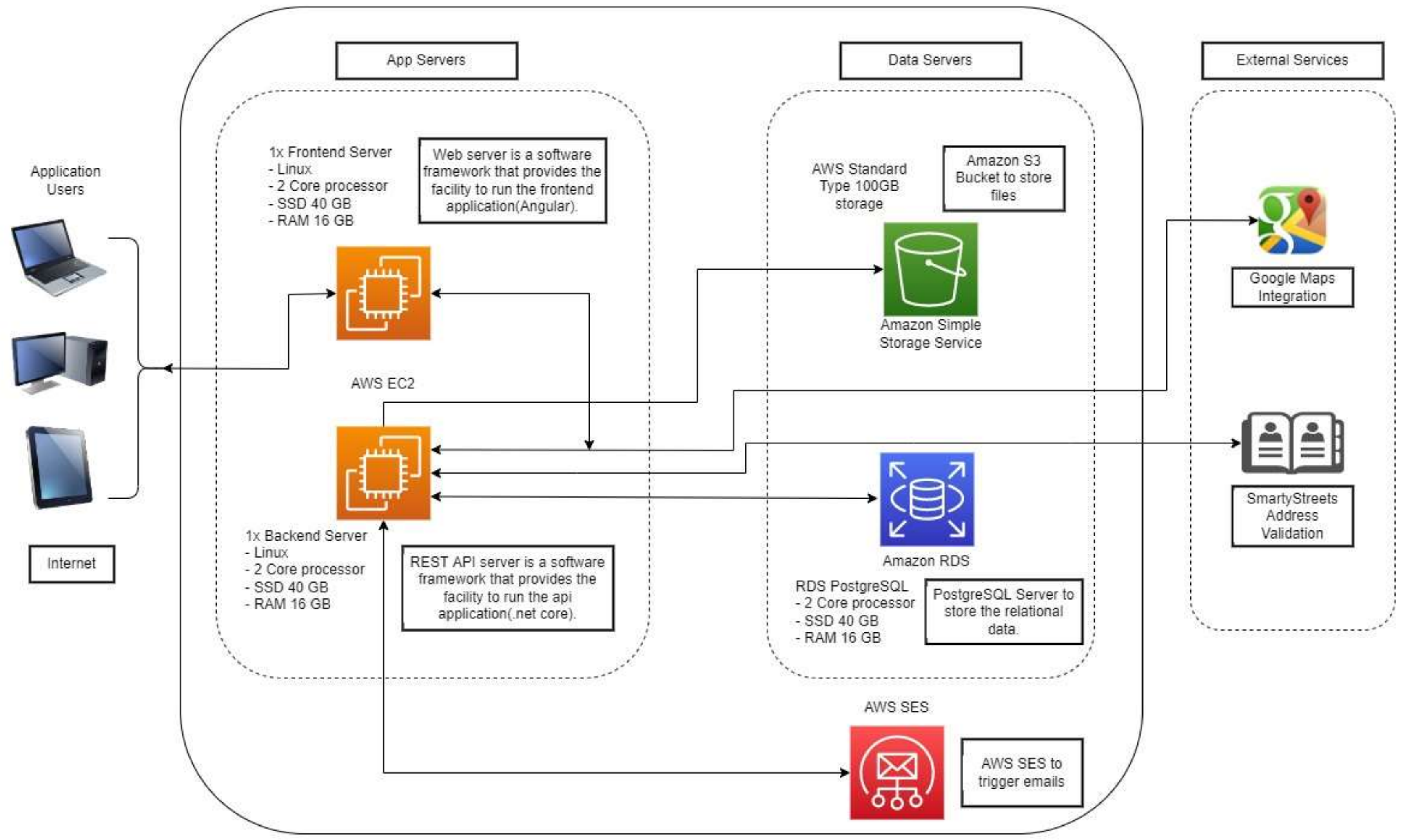

Cloud Reporting Application
Login Module
Admin Module
Company Module
User & Permission Management
Company Management
Subscription Management Module
Building Management
Admin Reports Module
Support Module
User & Permission Management
Building Management
Building Reports Upload Module
Building Reports View & Download Modue
Test Plan Module
Theme Management

COMPONENT & CONNECTION VIEW


Web Application
[Software System]
Application for the users to view and manage building test data
Users
[Person]
Users of the cloud reporting application.
Views and Manages Buildings using Https
API Call (HTTPS/JSON)
Company
[Component: .net core]
Allows user to add/update company and it's subscription
Authentication
[Component: .net core]
Allows the user to login into their account based on the login credentials
Support
[Component: .net core]
Allow admin to handle support for the company users.
Theme
[Component: .net core]
Helps the user to customize the UI appearance of the company
Uses
Uses
Uses
User
[Component: .net core]
Add/Update the user details and handling their permission
Building
[Component: .net Core]
Allows the user to add/update the Building and test information
Subscription
[Component: .net core]
Allows the admin users to manage subscription for the company
Uses
Permission
[Component: .net core]
To map the user permission based on the subscription and access provided by the admin
Uses
Uses
Uses
Reports
[Component: .net Core]
Allows the user to generate reports based on available filters
Upload
[Component: .net Core]
Allows the user to upload reports for a building
Test Plan
[Component: .net Core]
Allows the user to add/update test plan configurations
Uses
Database
[Container: PostgreSQL]
Store user information, building data, authentication credentials, etc.

- The previous slide shows the component – connector view with the high-level components within the system.
- The users will view and manage the test reports from the web application through the web(client).
- The web application interacts with the API application(server) through HTTP requests.
- The API application includes multiple modules that interacts with each other to provide the required response to the client.
- The API application also interacts with the postgres database to read and write the necessary data.

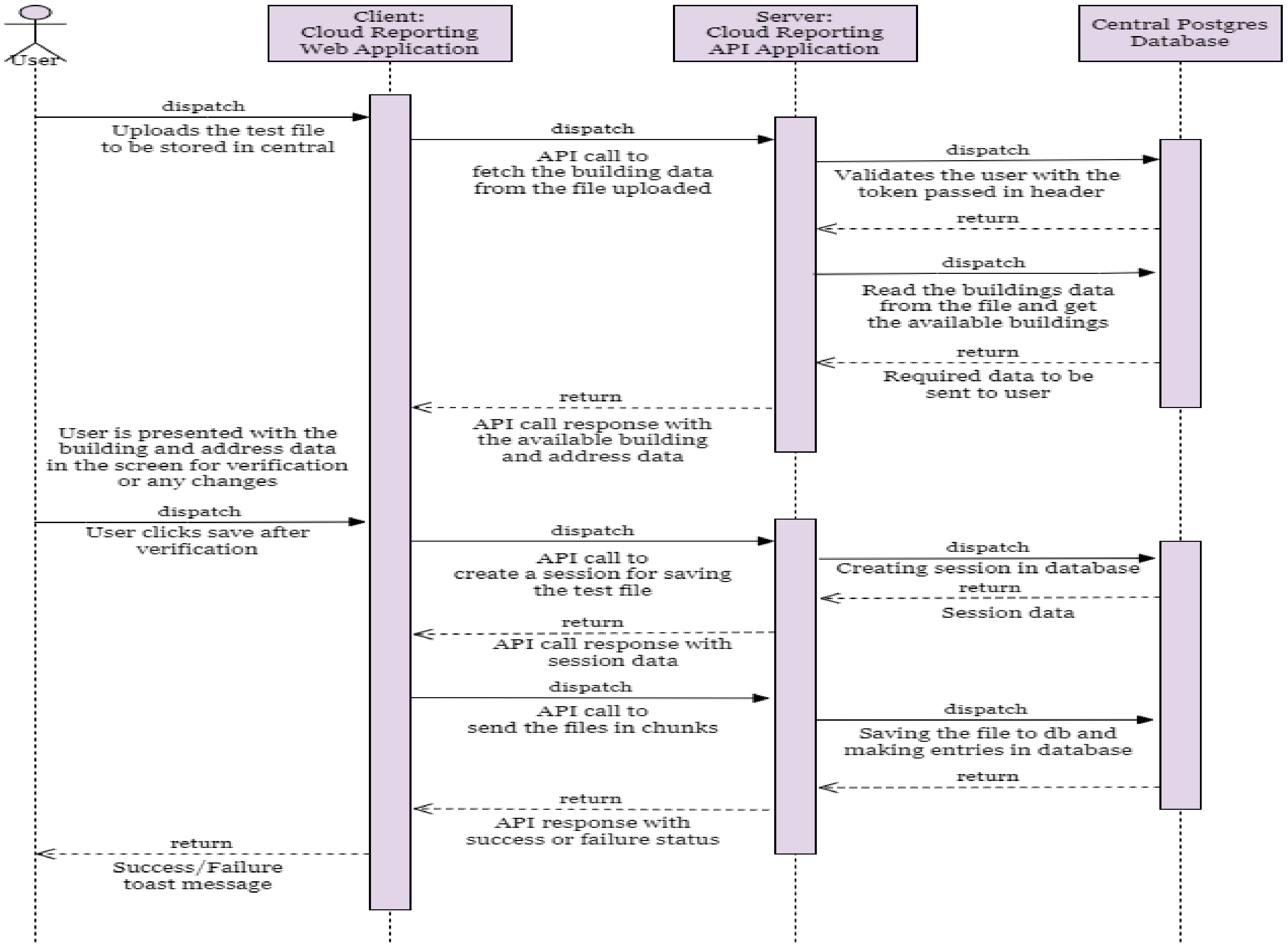
DEPLOYMENT VIEW


App Servers
Data Servers
External Services
Application Users
Internet
1x Frontend Server
- Linux
- 2 Core processor
- SSD 40 GB
- RAM 16 GB
Web server is a software framework that provides the facility to run the frontend application(Angular).
AWS EC2
1x Backend Server
- Linux
- 2 Core processor
- SSD 40 GB
- RAM 16 GB
REST API server is a software framework that provides the facility to run the api application(.net core).
AWS Standard Type 100GB storage
Amazon S3 Bucket to store files
Amazon Simple Storage Service
Amazon RDS
RDS PostgreSQL
- 2 Core processor
- SSD 40 GB
- RAM 16 GB
PostgreSQL Server to store the relational data.
AWS SES
AWS SES to trigger emails
Google Maps Integration
SmartyStreets Address Validation

- The previous slide shows the deployment view with the high-level components and how they are deployed.
- The application users can access the web application through web from various devices such as laptop, pc, tablets etc.
- The application is deployed using a variety of services from amazon.
- The frontend web application is deployed as web server using EC2.
- The backend API application is deployed as rest API server using EC2.
- S3 bucket is used for storing the test files and other files related to it.
- Postgres database is used for storing all the other data.
- Other external services that our server interacts with includes,
  - Google Maps API
  - SmartyStreets API

SEQUENCE DIAGRAM

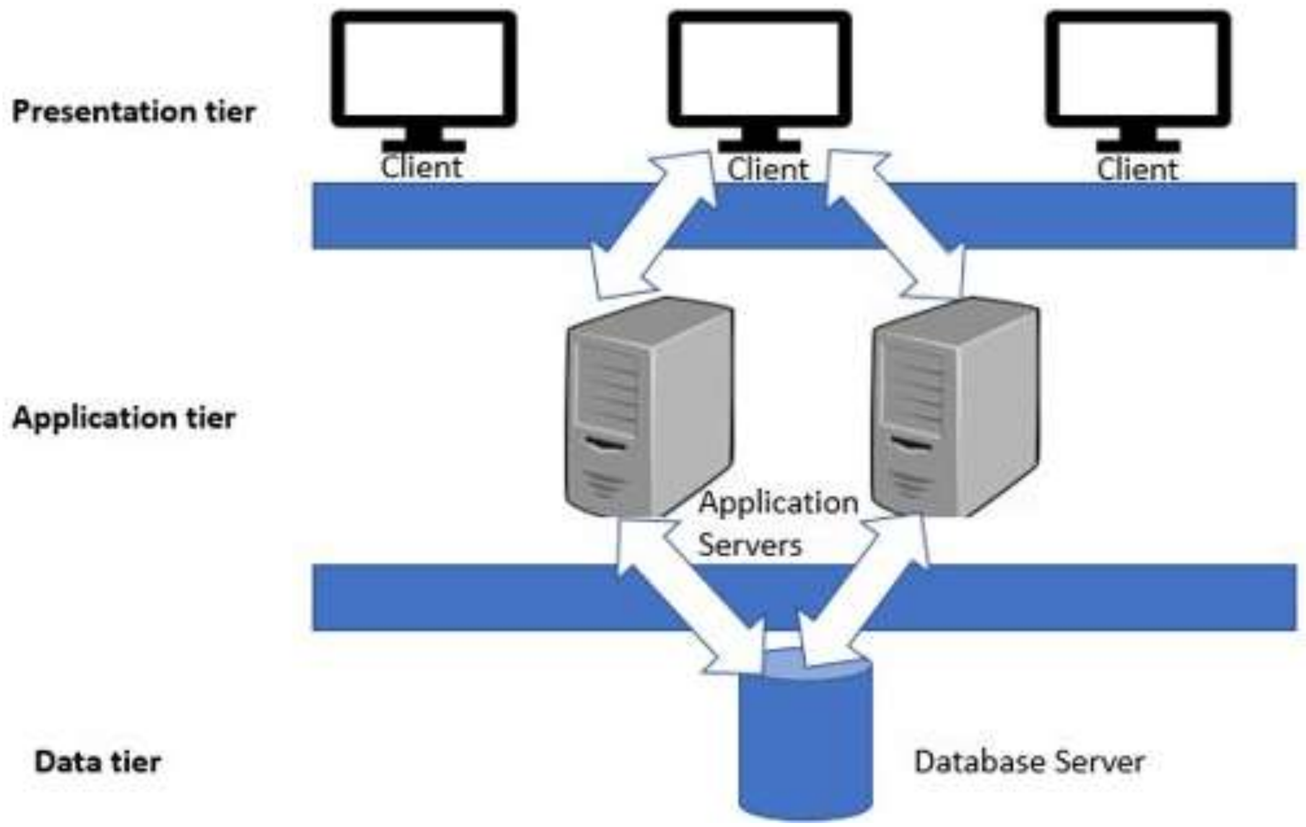
The sequence diagram of the process of uploading a test report from the cloud reporting web application to the central data store by a logged in company user.


User
Client: Cloud Reporting Web Application
Server: Cloud Reporting API Application
Central Postgres Database
dispatch
Uploads the test file to be stored in central
dispatch
API call to fetch the building data from the file uploaded
dispatch
Validates the user with the token passed in header
return
dispatch
Read the buildings data from the file and get the available buildings
return
Required data to be sent to user
return
API call response with the available building and address data
User is presented with the building and address data in the screen for verification or any changes
dispatch
User clicks save after verification
dispatch
API call to create a session for saving the test file
dispatch
Creating session in database
return
Session data
return
API call response with session data
dispatch
API call to send the files in chunks
dispatch
Saving the file to db and making entries in database
return
return
API response with success or failure status
return
Success/Failure toast message

- The previous slide shows the sequence diagram for uploading a test report manually from the cloud reporting web application.
- It's assumed that the user is logged into the application using his/her credentials and authenticated.
- Steps involved in uploading the test file are,
  - The user navigates to the test upload page, selects the test file to be uploaded and clicks next.
  - An API call is triggered to the server which will read the file and get the building details. The response will contain the building details which will be presented to the user as the second step.
  - The user can now verify the building details and update the details if required.
  - After confirmation, the user hits save which will trigger an API call to create a session for saving the test file details in the database for the corresponding buildings.
  - The server responds with a session id which will be used in the subsequent upload API calls.
  - The test file with the building details will now be uploaded as chunks of maximum 1MB size through the upload API one by one.
  - After all the chunks are uploaded, the server will combine them by identifying through the session ID. The final response will include the upload status.
  - A success message will display if all the API calls were successful. Else, a corresponding error message will be displayed.

ARCHITECTURAL PATTERN

- The system follows a "**Client-Server"** Architecture.
- Communication between the client and the server happens with REST services.



Client server architecture is a computing model in which the server hosts, delivers, and manages most of the resources and services requested by the client.

The application users can access the web application through client devices such as laptop, pc, tablets etc.

The server hosts the web application, API application and also interacts with the database to read and write data.

The web application is an angular single page application, and the API application is a .net core application.

The data is synced between the cloud application and the touch device by having a common data store and using rest services to fetch/store data.

The Amazon SES is used to send email notifications to the users.

KEY LEARNINGS

- Understood the different software architecture diagrams and their need to describe the requirements.
- Learnt that the software architecture diagrams should be detailed enough to explain maximum requirements.
- Understood the various architecturally significant requirements for the application and the tactics that can be used to implement them.
- Got to know about the importance of application security and how it can be implemented.
- Understood the architecture patterns and their usage in an application.

THANK YOU