Chapter 5
Implementation and Testing

5.1 Implementation Approaches

The project is developed using Iterative development methodology which delivers the software in successive iterations. Each iteration promises a working version of the software and adds new features to the previous iteration. User feedback is a key part of this approach which ensures active user interaction which allows for better understanding of the requirements, improvements or any change in requirements. Hence any change in the requirements is incorporated and both the developer and the user get a better clarity of the requirements given and the end product that will be developed. Various cycles of testing ensure that the particular iteration works well with other iterations as well.

Initially the database was designed, hosted on “clever-cloud.com” and the database connectivity was performed. For both the frontend and backend, Visual Studio Code was used which ensured a consistent developer experience. This was followed by the developing the user interfaces for a particular iteration and then communication with the backend was established. This ensured that a working part of the software was developed.

For the frontend written in Flutter, MVVM (Model- View- View Model) approach was taken to structure the code which guarantees modularity and separation of concerns. The View is responsible for interacting with the user, the View Model holds all the business logic and handles the state management and the model interacts with the backend. The project follows a structured file structure with the aim to avoid repeating code, encouraging reuse to speed up the development process and to ensuring code readability and maintainable by making changes at one place and there are reflected wherever necessary. Shared Preferences is used to persist data locally on the device and is one of many classes that follows a singleton pattern where only one instance of that class is required.

For the backend written in JavaScript enhanced by the Express.js library, a modular approach to structure the code by separating the concerns into routers, controllers and additional middle-wares. The controllers hold the core logic while the routers route the incoming HTTP request to the functions defined in these controllers. Additionally, JWT (JSON Web Tokens) are used for authenticating client requests and Socket.IO is used for incorporating the chatting feature.

Both the frontend and backend code also contain comments which ensures that the code is readable and maintainable.

5.2 Coding details and Code Efficiency

5.2.1 Coding details

auth_controller.js

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//----------------------------------STUDENT REGISTRATION---------------------------------------

//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

//CHECK STUDENT DETAILS AND SEND OTP

const checkStudentDetailsAndSendOTP = async (

  req,

  res,

  { shouldSendOTP = true }

) => {

  //const pool = connectDB();

  const conn = await pool.getConnection();


  let {

    control_faculty_id,

    user_firstname,

    user_middlename,

    user_lastname,

    user_phone,

    user_email,

  } = req.body;

  //THE MIDDLE NAME AND LAST NAME CAN BE EMPTY WHEN RECEIVED FROM THE FRONTEND

  //Make the names to lowercase to compare it in the query

  var user_middlename_lowercase = "";

  var user_lastname_lowercase = "";

  if (user_middlename.length != 0) {
```

```
    user_middlename_lowercase = user_middlename.toLowerCase();
  }
  if (user_lastname.length != 0) {
    user_lastname_lowercase = user_lastname.toLowerCase();
  }
  var user_firstname_lowercase = user_firstname.toLowerCase();


/*----------------------------------
VALIDATE THE REQUIRED DETAILS
-----------------------------------*/
  //validate the details
   if (control_faculty_id.length == 0) {
    isValidStudent = false;
    throw new BadRequestError("Student's control ID required");
  } else if (user_email.length == 0) {
    isValidStudent = false;
    throw new BadRequestError("Student's email required");
  } else if (user_firstname.length == 0) {
    isValidStudent = false;
    throw new BadRequestError("Student's firstname required");
  } else if (user_phone.length == 0) {
    isValidStudent = false;
    throw new BadRequestError("Student's phone number required");
  }
  /*----------------------------------
CHECK WHETHER A SIMILAR USER EXISTS
-----------------------------------*/
  //check whether the student with the SAME EMAIL has already registered
  const [email_rows, metadata_about_fields1] = await conn.query(
```

```
  "select user_email from users where user_email=?;",
  [user_email]
 );
 if (email_rows.length > 0) {
  isValidStudent = false;
  throw new BadRequestError("Student with this email is already registered");
 }
 //check whether the student with the SAME CONTROL ID has already registered
 const [control_id_rows, metadata_about_fields2] = await conn.query(
  "select * from users,current_students_info where (control_faculty_id=? ) and (control_id=control_faculty_id and registration_status='registered');",
  [control_faculty_id]);
 if (control_id_rows.length > 0) {
  isValidStudent = false;
  throw new BadRequestError(
   "Student with this Control ID is already registered"
  );
 }
 /*-------------------------------------------------------------------------------
CHECK THE DETAILS OF NEW STUDENT
we reach to this stage only if the student isnt registered
If the user_middlename and user_lastname is empty then check for NULL
-------------------------------------------------------------------------------*/
 var student_middlename_query_part = `LOWER(middlename)='${user_middlename_lowercase}'`;
 var student_lastname_query_part = `LOWER(lastname)='${user_lastname_lowercase}'`;

 if (user_middlename == "") {
  student_middlename_query_part = `middlename IS NULL`;
```

```
  }


  if (user_lastname == "") {
    student_lastname_query_part = `lastname IS NULL`;
  }
  //check whether the student is a student of VAZE college
  const [row3, metadata_about_fields3] = await conn.query(
    `SELECT * from current_students_info where control_id=? and LOWER(firstname)=? AND ${student_middlename_query_part} AND ${student_lastname_query_part};`,
    [control_faculty_id, user_firstname_lowercase]
  );
  if (row3.length != 1) {
    isValidStudent = false;
    //If firstname is missing
    const [row5, metadata_about_fields6] = await conn.query(
      `SELECT * FROM current_students_info where control_id=? AND ${student_lastname_query_part} AND ${student_middlename_query_part};`,
      [control_faculty_id]
    );
    if (row5.length == 1) {
      throw new BadRequestError(
        "Please check the entered First Name. Pleae fill it in if not done so"
      );
    } else {
      //check if the middlename is missing
      const [row6, metadata_about_fields7] = await conn.query(
        `SELECT * FROM current_students_info where control_id=? AND LOWER(firstname)=? AND ${student_lastname_query_part} AND middlename IS NOT NULL;`,
        [control_faculty_id, user_firstname_lowercase]
```

```
    );
    if (row6.length == 1) {
      throw new BadRequestError(
        "Please check the entered Middle Name. Pleae fill it in if not done so"
      );
    } else {
      //check if the lastname is missing
      const [row7, metadata_about_fields8] = await conn.query(
        `SELECT * FROM current_students_info where control_id=? and LOWER(firstname)=? AND ${student_middlename_query_part} AND lastname IS NOT NULL;`,
        [control_faculty_id, user_firstname_lowercase]
      );
      if (row7.length == 1) {
        throw new BadRequestError(
          "Please check the entered Last Name. Pleae fill it in if not done so"
        );
      } else {
        //check if the lastname is missing
        throw new BadRequestError("Not a current student of Vaze college");
      }}}}
  isValidStudent = true;
  if (shouldSendOTP) {
    //send the email first then store the details if the email is sent successfully
    var name = `${user_firstname} ${user_lastname}`;
    var otp = generateOTP();

    var add_name_to_html_content =
      HTMLEmailContents.registration_verify_email_html.replace(
```

```
        "[NAME]",
        name
      );
    var final_edited_html_content = add_name_to_html_content.replace(
      "[OTP]",
      otp
    );
    //SEND THE OTP
    var isEmailSentObject = await sendEmail(
      user_email,
      "Find My Stuff: Verify your Email",
      final_edited_html_content
    );
    if (isEmailSentObject.email_sent_status) {
      conn.release();
      return res.status(200).json({ msg: "Valid Student and OTP sent successfully", otp: otp });
} else {
      isValidStudent = false;
      conn.release();
      //await pool.end()
      throw new EmailNotSentError(isEmailSentObject.email_err);
    }
};

//2.REGISTER STUDENT USER
const registerStudentUser = async (req, res) => {
  // register the user
  //const pool_1 = connectDB();
  const conn_1 = await pool.getConnection();
```

```
  const {
    control_faculty_id,
    user_firstname,
    user_middlename,
    user_lastname,
    user_phone,
    user_email,
    user_password,
  } = req.body;
  await checkStudentDetailsAndSendOTP(req, res, false);
  if (isValidStudent) {
    /*---------------------------------------
       HASH THE PASSWORD
       ---------------------------------------*/
    const salt = await bcryptjs.genSalt(10);
    const hashedPassword = await bcryptjs.hash(user_password, salt);

    /*---------------------------------------
       STORE DETAILS IN THE DATABASE
       ---------------------------------------*/
    var full_name = `${user_lastname} ${user_firstname} ${user_middlename}`;
    var student_user_type = "s"; //'s' denotes that the user is a student
    var account_status = "a"; //'a' denotes that the user's account is 'active' and not 'suspended'
    const [registerResult, metadata_about_fields1] = await conn_1.query(
      "INSERT INTO users(control_faculty_id, user_email,user_phone,
user_fullname,user_password,user_type,user_account_status) values (?,?,?,?,?,?,?);",
      [control_faculty_id, user_email, user_phone, full_name, hashedPassword,
student_user_type, account_status,]);
    if (registerResult.affectedRows == 1) {
```

```
    //UPDATE THE REGISTRATION STATUS
    const [update_reg_status_result, metadata_about_fields21] =
      await conn.query(
        "UPDATE current_faculty_info set registration_status='registered' WHERE control_id=?;",
        [control_faculty_id]);
    if (update_reg_status_result.affectedRows == 1) {
      res.status(StatusCodes.OK).json({ msg: "Student Registration Successful" });
    } else {
      conn.release();
      throw new BadRequestError(
        "Registration Status couldn't be updated.Please try again later"
      );
    }
   } else {
    throw new BadRequestError("Registration Unsuccessful");
   }
  } else {
   throw new BadRequestError("Invalid Student Details");
  }
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//----------------------------USER(STUDENT AND FACULTY) LOGIN ---------------------------
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
//.LOGIN USER
const loginUser = async (req, res) => {
  /*---------------------------------
   DB CONNECTION
   ----------------------------------*/
```

```
//const pool = connectDB();
const conn = await pool.getConnection();

/*------------------------------------------------
  EXTRACT EMAIL AND PASSWORD FROM THE REQUEST BODY
  ------------------------------------------------*/
const { user_email, user_password } = req.body;

/*------------------------------------------------
 CHECK IF THE EMAIL AND PASSWORDS FIELDS ARE EMPTY
  ------------------------------------------------*/
if (user_email.length == 0) {
  throw new BadRequestError("Please provide your registered email id");
} else if (user_password.length == 0) {
  throw new BadRequestError("Please fill in your password");
}
/*------------------------------------------------------------
  CHECK WHETHER USER EXISTS WITH THE EMAIL FROM THE REQUEST BODY
  ------------------------------------------------------------*/
const [email_exists_result, metadata_about_fields6] = await conn.query(
  "SELECT * FROM users where user_email=?",
  [user_email]
);
if (email_exists_result.length == 1) {
  // res.status(200).send("the email exists");
  const [password_fetch_result, metadata_about_fields7] = await conn.query(
    "SELECT user_password,control_faculty_id,user_type from users where user_email=?",
    [user_email]
  );
```

```
    if (password_fetch_result.length == 1) {
      let actual_password = password_fetch_result[0].user_password;
      const isPasswordValid = await bcryptjs.compare(user_password, actual_password);
      if (isPasswordValid) {
        //ALSO SEND THE JWT
        var user_id = password_fetch_result[0].control_faculty_id;
        var token = jwt.sign({ user_id }, process.env.JWT_SECRET);
        //GET THE USER TYPE AND SEND IT
        let userType= password_fetch_result[0].user_type;
        res.status(StatusCodes.OK).json({ msg: "Correct Password", token: token , user_type:userType});
      } else {
        throw new BadRequestError("Incorrect Password");
      }
    } else {
      throw new BadRequestError(
        "Either multiple users with the same email exist OR no one exists with this email"
      );
    }
  } else {
    throw new BadRequestError("User doesn't exist");
  }
};
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
-------------------------------COMMON TO ADMIN, STUDENT AND FACULTY-----------------
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

//1.RESEND THE EMAIL
const resendEmail = async (req, res) => {
```

```
  const { user_firstname, user_lastname, user_email } = req.body;
  var otp = generateOTP();
  var name = `${user_firstname} ${user_lastname}`;

  var add_name_to_html_content =
    HTMLEmailContents.registration_verify_email_html.replace("[NAME]", name);
  var final_edited_html_content = add_name_to_html_content.replace(
    "[OTP]",
    otp
  );
  //SEND THE OTP
  var isEmailSentObject = await sendEmail(
    user_email,
    "Find My Stuff: OTP Resent",
    final_edited_html_content
  );

  if (isEmailSentObject.email_sent_status) {
    res
      .status(StatusCodes.OK)
      .json({ msg: "OTP verification email resent", otp: otp });
  } else {
    throw new EmailNotSentError(isEmailSentObject.email_err);
  }
};

//2. Verify Email
const checkWhetherTheUserWithEmailExistsForgotPassword = async (req, res) => {
  /*---------------------------------
```

```
  DB CONNECTION
  ----------------------------------*/
const conn = await pool.getConnection();

/*-----------------------------------------------
  EXTRACT EMAIL AND PASSWORD FROM THE REQUEST BODY
  -----------------------------------------------*/
const { email, user_type } = req.body;

if (email.length == 0) {
  throw new BadRequestError("Please provide the email");
} else if (user_type.length == 0) {
  throw new BadRequestError("Please provide the user type");
}

/*-----------------------------------------------
CHECK THE USER TYPE AND HANDLE THE REQUEST ACCORDINGLY
  -----------------------------------------------*/
//THE USER
if (user_type == "UserTypeEnum.user") {
  const [check_user_email_result, metadata_about_fields22] = await conn.query(
    "SELECT * FROM users WHERE user_email=?",
    [email]
  );
  if (check_user_email_result.length == 1) {
    //TODO:FETCH THE USER'S FIRST NAME AND LAST NAME
    var firstname = check_user_email_result[0].user_firstname;
    var lastname = check_user_email_result[0].user_lastname;
    var name = `${firstname} ${lastname}`;
```

```
    await beforeSendingEmailFormattingAndSendEmail(
      req,
      res,
      conn,
      email,
      name
    );
  } else {
    throw new BadRequestError("No such user exists");
  }
}
//IF USER IS ADMIN
else if (user_type == "UserTypeEnum.admin") {
  const [check_admin_email_result, metadata_about_fields23] =
    await conn.query("SELECT * FROM admin_users WHERE admin_email=?", [
      email,
    ]);
  if (check_admin_email_result.length == 1) {
    //TODO:FETCH THE USER'S FIRST NAME AND LAST NAME
    var firstname = check_admin_email_result[0].admin_firstname;
    var lastname = check_admin_email_result[0].admin_lastname;
    var name = `${firstname} ${lastname}`;

    await beforeSendingEmailFormattingAndSendEmail(
      req,
      res,
      conn,
      email,
```

```
      name
    );
   } else {
    throw new BadRequestError("No such admin exists");
   }
  }
};

//3. Reset Password
const resetPassword = async (req, res) => {
 //reset the password of the user
 /*----------------------------------
 DB CONNECTION
 ----------------------------------*/
 const conn = await pool.getConnection();
 try {
  /*----------------------------------
  EXTRACT DATA FROM REQUEST
  ----------------------------------*/
  const { email, user_type, password } = req.body;

  if (email.length == 0) {
   throw new BadRequestError("Please provide the email");
  } else if (user_type.length == 0) {
   throw new BadRequestError("Please provide the user type");
  } else if (password.length == 0) {
   throw new BadRequestError("Please provide a password");
  }
```

```
 /*---------------------------------
IF ITS A USER
----------------------------------*/
 if (user_type == "UserTypeEnum.user") {
   //Check whether the password is similar to the one already present
   const [fetch_users_password_result, metadata_about_fields24] =
     await conn.query("SELECT * FROM users WHERE user_email=?", [email]);

   //var testinghashedPassword= await bcryptjs.hash("Aa1234567",10);
   if (fetch_users_password_result.length == 1) {
     var compareResult = await bcryptjs.compare(
       password,
       fetch_users_password_result[0].user_password
       //testinghashedPassword
     );
     var hashedPassword = await bcryptjs.hash(password, 10);
     //if the password is same as the previous one
     if (compareResult) {
       throw new BadRequestError(
         "Please set a different password than the previous one"
       );
     }

     //If the password is not the same then,
     //Hash the password
     var salt = await bcryptjs.genSalt(10);
     var hashedPassword = await bcryptjs.hash(password, salt);

     //Store the password
```

```
      const [reset_password_result, metadata_about_fields25] =
        await conn.query(
          "UPDATE users set user_password=? WHERE user_email=?",
          [hashedPassword, email]
        );

      if (reset_password_result.affectedRows == 1) {
        return res
          .status(StatusCodes.OK)
          .json({ msg: "User's password reset successfully" });
      } else {
        throw new BadRequestError("Password couldn't be reset");
      }
    } else {
      throw new BadRequestError("User does not exist with such an email");
    }
  } else if (user_type == "UserTypeEnum.admin") {
    /*---------------------------------
 IF ADMIN
-----------------------------------*/
    const [fetch_admins_password_result, metadata_about_fields26] =
      await conn.query("SELECT * FROM admin_users WHERE admin_email=?", [
        email,
      ]);

    if (fetch_admins_password_result.length == 1) {
      var compareResult = await bcryptjs.compare(
        password,
        fetch_admins_password_result[0].admin_password
```

```
  );

  //if the password is same as the previous one
  if (compareResult) {
    throw new BadRequestError(
      "Please set a different password than the previous one"
    );
  }

  //If the password is not the same then,
  //Hash the password
  var salt = await bcryptjs.genSalt(10);
  var hashedPassword = await bcryptjs.hash(password, salt);

  //Store the password
  const [reset_password_result, metadata_about_fields25] =
    await conn.query(
      "UPDATE admin_users set admin_password=? WHERE admin_email=?",
      [hashedPassword, email]
    );

  if (reset_password_result.affectedRows == 1) {
    return res
      .status(StatusCodes.OK)
      .json({ msg: "Admin's password reset successfully" });
  } else {
    throw new BadRequestError("Password couldn't be reset");
  }
} else {
```

```
        throw new BadRequestError("Admin does not exist with such an email");
      }
    }
  } catch (e) {
    throw new BadRequestError(e.toString());
  } finally {
    conn.release();
  }
};



//4.FETCH LOGGED IN USER DETAILS
const fetchLoggedInUserDetails = async (req, res) => {
  /*----------------------------------
    EXTRACT EMAIL FROM REQUEST BODY
    ----------------------------------*/

  const { email_of_user_to_fetch, user_type } = req.body;

  /*----------------------------------
    DB CONNECTION
    ----------------------------------*/

  const conn = await pool.getConnection();

  /*----------------------------------
    IF NO EMAIL IS ENTERED
    ----------------------------------*/
```

```
  if (email_of_user_to_fetch.length == 0) {
    conn.release();
    throw new BadRequestError(
      "No email provided. Couldn't fetch the user details"
    );
  }

  /*-------------------------------------------------------------
    CHECK THE TYPE OF USER AND FETCH THE DETAILS ACCORDINGLY
    -------------------------------------------------------------*/
  if (user_type == "UserTypeEnum.user" || user_type == "UserTypeEnum.student" || user_type
== "UserTypeEnum.faculty") {
    const [fetch_details_result] = await conn.query(
      `SELECT control_faculty_id, user_fullname from users where user_email=?`,
      [email_of_user_to_fetch]
    );
    if (fetch_details_result.length != 1) {
      conn.release();
      throw new BadRequestError(
        "Something's not right. Couldn't fetch the user's details"
      );
    } else {
      var control_faculty_id = fetch_details_result[0].control_faculty_id;
      var user_fullname = fetch_details_result[0].user_fullname;
      var details_map = {
        control_faculty_id: control_faculty_id,
        user_fullname: user_fullname,
      };
```

```
    conn.release();
    return res.status(StatusCodes.OK).json(details_map);
   }
  }

  //IF THE USER IS AN ADMIN
  else if (user_type == "UserTypeEnum.admin") {
   const [fetch_details_result, metadata_about_fields91] = await conn.query(
    `SELECT admin_id, admin_fullname from admin_users where admin_email=?`,
    [email_of_user_to_fetch]
   );
   if (fetch_details_result.length != 1) {
    conn.release();
    throw new BadRequestError(
     "Something's not right. Couldn't fetch the user's details"
    );
   } else {
    var admin_id = fetch_details_result[0].admin_id;
    var admin_fullname = fetch_details_result[0].admin_fullname;
    var details_map = {
     admin_id: admin_id,
     admin_fullname: admin_fullname,
    };
    conn.release();
    return res.status(StatusCodes.OK).json(details_map);
   }
  }

};
```

```
//5.STORE USER DEVICE DETAILS
const storeUserDeviceDetails = async (req, res) => {
  /*----------------------------------
    DB CONNECTION
    ----------------------------------*/
  //const pool = connectDB();
  const conn = await pool.getConnection();

  /*----------------------------------
    EXTRACT THE INFO TO STORE
    ----------------------------------*/
  const { device_id, device_token, user_or_admin_id, user_type } = req.body;

  /*----------------------------------------------------------------------------------------------------------
------
    CHECK WHETHER THE DEVICE ID IS ALREADY PRESENT IN THE DB.IF YES,
THEN UPDATE THE INFO, ELSE STORE THE NEW DEVICE INFO
    ----------------------------------------------------------------------------------------------------------
-------*/
  const [device_id_already_present_fetch_result, metadata_about_fields12] =
    await conn.query("SELECT * FROM user_device_details where device_id=?", [
      device_id,
    ]);
  if (device_id_already_present_fetch_result.length != 0) {
    await updateDeviceDetailsInDB(
      conn,
      device_id,
      device_token,
```

```
      user_or_admin_id,
      user_type,
      req,
      res
    );
  } else {
    await insertDeviceDetailsIntoDB(conn, device_id, device_token, user_or_admin_id, user_type, req, res); }
};
```

chats_and_messages_controller.js

```
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//------------------------------------EXPORTED FUNCTIONS -----------------------------------------
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

/*////////////////////////////////////////
//1. FETCH  ALL CHATS
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*/

const fetchAllChats = async (req, res) => {
  /*----------------------------
EXTRACT PARAMETERS FROM THE QUERY
-------------------------------*/
  const control_faculty_id = req.params.loggedInUserId;
  const user_type = req.query.user_type;
  /*----------------------------
PARAMETER VALIDATION
this step is important as it prevents against SQL injection attacks
-------------------------------*/
  if (control_faculty_id == undefined) {
```

```
    throw new BadRequestError(
      "REQUIRED: control faculty id of sender is not defined"
    );
  } else if (control_faculty_id.toString().length < 10) {
    throw new BadRequestError(
      "INVALID: control faculty id of sender  is not valid"
    );
  } else if (user_type == undefined) {
    throw new BadRequestError("REQUIRED: no user type defined");
  } else if (user_type.length == 0) {
    throw new BadRequestError("REQUIRED: no user type defined");
  }
  /*----------------------------
DB CONNECTION
----------------------------------*/
  const conn = await pool.getConnection();
  /*----------------------------
VARIABLES
----------------------------------*/
  let queryMap = {
    query_to_execute: "",
    query_params: "",
  };
  /*----------------------------
MAIN LOGIC
----------------------------------*/
  if (user_type == "user") {
    queryMap.query_to_execute = `Select c.chat_id,m.max_m_datetime as
latest_msg_datetime, m.m_content as latest_msg, u.control_faculty_id as
```

```
receipient_user_id,u.user_fullname AS receipient_user_name , u.user_phone AS receipient_user_phone,a.admin_id AS receipient_admin_id,a.admin_fullname AS receipient_admin_name,a.admin_phone AS receipient_admin_phone

  from chats AS c join participates_in on c.chat_id = participates_in.p_chat_id and participates_in.p_user_id= ?

  join participates_in as p on p.p_chat_id= c.chat_id and (p.p_user_id!=? or p.p_user_id IS NULL)

  LEFT join messages on messages.chat_id= c.chat_id

  left join users as u on u.control_faculty_id=p.p_user_id

  left join admin_users as a on a.admin_id=p.p_admin_id

  LEFT JOIN (

    SELECT ch.chat_id as new_chat_id,MAX(m1.m_datetime) AS max_m_datetime,m1.m_content,m1.m_id

    FROM messages as m1, chats AS ch, messages as m2 where ch.chat_id= m1.chat_id GROUP by ch.chat_id

  ) AS m

  ON c.chat_id = m.new_chat_id and messages.m_id= m.m_id

  GROUP BY c.chat_id

   `;


  queryMap.query_params = [control_faculty_id, control_faculty_id];


 }
 const [fetch_chats_result] = await conn.query(
  queryMap.query_to_execute,
  queryMap.query_params
 );
 if (fetch_chats_result) {
  let response = {
   chats: fetch_chats_result,
```

```
      total_chats: fetch_chats_result.length,
    };
    conn.release();
    res.status(StatusCodes.OK).json(response);
  } else {
    throw new BadRequestError("No chats found")}};
//2. Fetch messages of between logged in user and other user
const fetchMessagesBetweenLoggedInUserAndOtherUser = async (req, res) => {
  /*----------------------------
EXTRACT PARAMETERS FROM THE QUERY
---------------------------------*/
  const control_faculty_id = req.params.loggedInUserId;
  //const other_user_id = req.query.other_user_id;
  const chat_id = req.query.chat_id;
  /*----------------------------
PARAMETER VALIDATION
this step is important as it prevents against SQL injection attacks
---------------------------------*/
  if (control_faculty_id == undefined) {
    throw new BadRequestError(
      "REQUIRED: control faculty id of sender is not defined"
    );
  } else if (control_faculty_id.toString().length < 10) {
    throw new BadRequestError(
      "INVALID: control faculty id of sender  is not valid"
    );
  }
  else if (chat_id == undefined) {
    throw new BadRequestError("REQUIRED: no chat_id defined");
```

```
  } else if (chat_id.toString().length < 1) {
    throw new BadRequestError("INVALID: no chat_id provided");
  }
  /*----------------------------
DB CONNECTION
-------------------------------*/
  const conn = await pool.getConnection();
  /*----------------------------
MAIN LOGIC
-------------------------------*/
  const [fetch_msg_result] = await conn.query(
    "SELECT * FROM messages WHERE chat_id=?",
    [chat_id]
  );

  if (fetch_msg_result) {
    let jsonResponse = {};
    jsonResponse.messages = fetch_msg_result;
    jsonResponse.total_messages = fetch_msg_result.length;
    conn.release();
    return res.status(StatusCodes.OK).json(jsonResponse);
  } else {
    throw new BadRequestError("No messages found");
  }
};
```

google_drive_api_controller.js

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//-------------------------------------------EXPORTED FUNCTIONS ------------------------------
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
```

```
//1. upload images to google drive
const uploadImagesToGoogleDrive=async(base64Data, fileName)=>{
    try {
      // Create a Google Drive instance
      const drive = google.drive({ version: 'v3', auth:googleAuth });
      // Decode base64 to binary
      const binaryData = Buffer.from(base64Data, 'base64');

      // Create a readable stream from binary data
      const stream = new require('stream').PassThrough();
      stream.end(binaryData);
      // Upload the file to Google Drive
      const response = await drive.files.create({
        requestBody: {
          name: fileName,
          parents:["1xxY-zRaVvh3LOGpzL9vZ5O_m3DWX1MZ4"] //folder id where to
upload
          // user_id:user_id,
          // date_uploaded:date_uploaded
        },
        media: {
          mimeType: 'image/webp',
          body: stream,
        },
      });
      return response.data;
    } catch (error) {
      console.error('Error uploading file to Google Drive:', error.message);
```

```
        throw new BadRequestError(`Error uploading file to Google Drive: ${error.message}`);
      }
}


//2. Delete images from google drive of that user and that item temporarily ie put in trash
const deleteImageOfReportedItemFromgDrive= async(imageFileIdToDelete)=>{
  try {
    const gdrive= google.drive({version: 'v3', auth:googleAuth });
    const response= await gdrive.files.update(
      {fileId: imageFileIdToDelete, requestBody:{trashed:true}}
    );
  } catch (error) {
    console.error('Error deleting file from Google Drive:', error.message);
    throw new BadRequestError(`Error deleting file from Google Drive: ${error.message}`);


  }
```

Reported_items_controller.js

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//-----------------------------------EXPORTED FUNCTIONS ---------------------------------------
//\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
//1. Fetch all items
const fetchAllItems = async (req, res) => {
  /*----------------------------------
DB CONNECTION
----------------------------------*/
  const conn = await pool.getConnection();
```

```
  /*---------------------------------
EXTRACT  OFFSET FROM THE REQUEST FOR LAZY LOADING
-----------------------------------*/
  //const {offset} = req.headers.offset;
  let { offset, sort, categories, toDate, fromDate, searchQuery } = req.query;
  //const { offset } = req.body;
  offset = parseInt(offset);

  //offset is used to skip a certain number of rows and fetch from the nth row
  //eg: offset is 20, so the query will fetch from the 21st row

  /*---------------------------------
VARIABLES
-----------------------------------*/
  var categoriesList;
  var sort_part_of_query;
  let fetch_filtered_items_result; //the result of the query
  //let totalFilteredItemsCountWithoutLimitOffset; //the result of the count query
  const limitToFetchItems = 20;
  let totalLostReportedItems = 0;
  let totalFoundReportedItems = 0;
  let fetch_fitered_found_items_count_result;
  let fetch_fitered_lost_items_count_result;

  //Map/Object of the current query to execute along with its count query
  let queryToExecute = {
    query: "",
    params: "",
    lost_items_count_query: "",
```

```
  found_items_count_query: "",
  count_query_params: "",
 };

 //POSSIBLE SORT PARAMETERS
 // ReportedItemSortingOptionsEnum.newestfirst
 // ReportedItemSortingOptionsEnum.oldestfirst,
 // ReportedItemSortingOptionsEnum.atoz,
 // ReportedItemSortingOptionsEnum.ztoa

 /*---------------------------------
QUERY PARAMS VALIDATION
----------------------------------*/
 if (offset.length == 0) {
  conn.release();
  throw new BadRequestError("Please provide the offset");
 }

 if (
  (fromDate != undefined && toDate == undefined) ||
  (fromDate != undefined && toDate == undefined)
 ) {
  conn.releaes();
  throw new BadRequestError(
   "Please provide both  the fromDate and toDate filter. "
  );
 }

 if (sort == undefined) {
```

```
    conn.release();
    throw new BadRequestError("Please provide the sort parameter");
  }
  //get the categories id from the query parameters
  if (categories != undefined) {
    categoriesList = categories.split(",");
  }
  /*----------------------------------
SORT LOGIC
----------------------------------*/
  switch (sort) {
    case "ReportedItemSortingOptionsEnum.newestfirst":
      sort_part_of_query = "ORDER BY reported_item.r_date_of_loss DESC";

      break;
    case "ReportedItemSortingOptionsEnum.oldestfirst":
      sort_part_of_query = "ORDER BY reported_item.r_date_of_loss ASC";
      break;
    case "ReportedItemSortingOptionsEnum.atoz":
      sort_part_of_query = "ORDER BY reported_item.r_name ASC";
      break;
    case "ReportedItemSortingOptionsEnum.ztoa":
      sort_part_of_query = "ORDER BY reported_item.r_name DESC";
      break;
    default:
      sort_part_of_query = "ORDER BY reported_item.r_date_of_loss DESC";
      break;
  }
  /*----------------------------------------------------------------------
```

```
SEARCH LOGIC
----------------------------------------------------------------------*/
 var charsInSearchQuery = [];
 var lowerCaseSearchQuery = "";
 var searchQueryRegex = "";
 if (searchQuery != undefined) {
  if (searchQuery.trim().length > 0) {
   charsInSearchQuery = searchQuery.toLowerCase().split("");
   lowerCaseSearchQuery = searchQuery.toLowerCase();

   searchQueryRegex = `^(?i)${lowerCaseSearchQuery}`;
   // searchQueryRegexDynamic = `^(?i)(?=.*${charsInSearchQuery.join(
   //   ")(?=.*"
   // )})`;
  }
 }
 /*----------------------------------------------------------------------
DETERMINE THE QUERY TO USER TO FETCH  20 ITEMS of applying filters
----------------------------------------------------------------------*/
 if (
  searchQuery != undefined &&
  categoriesList != undefined &&
  toDate != undefined &&
  fromDate != undefined
 ) {
  //ALL PARAMETERS ARE SET
  //Main query
  queryToExecute.query = `SELECT reported_item.*, GROUP_CONCAT(DISTINCT
reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT
```

```
security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers

   FROM reported_item

   JOIN reported_image ON reported_item.r_id = reported_image.r_id

   JOIN category ON reported_item.category_id = category.cat_id

    JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id

   WHERE reported_item.category_id IN (?) AND reported_item.r_date_of_loss BETWEEN ? AND ?

 AND reported_item.r_name REGEXP ?

   GROUP BY reported_item.r_id  ${sort_part_of_query}

 LIMIT ? OFFSET ?;`;

   //Main query parameters

   queryToExecute.params = [categoriesList, fromDate, toDate, searchQueryRegex, limitToFetchItems, offset,];

   //Lost items Count query

   queryToExecute.lost_items_count_query = `WITH common_reported_item AS (

    SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers

     FROM reported_item

     JOIN reported_image ON reported_item.r_id = reported_image.r_id

     JOIN category ON reported_item.category_id = category.cat_id

      JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id

     WHERE reported_item.category_id IN (?)

     AND reported_item.r_date_of_loss BETWEEN ? AND ?

     AND reported_item.r_name REGEXP ?

   AND reported_item.r_reported_as='l'

     GROUP BY reported_item.r_id  ${sort_part_of_query}
```

```
    )


    SELECT COUNT(*)  AS total_lost_filtered_items FROM (
     SELECT * FROM common_reported_item
     ) as all_searched_reported_item;`;
   //Found items query
   queryToExecute.found_items_count_query = `WITH common_reported_item AS (
    SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers
    FROM reported_item
    JOIN reported_image ON reported_item.r_id = reported_image.r_id
    JOIN category ON reported_item.category_id = category.cat_id
     JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id
    WHERE reported_item.category_id IN (?)
    AND reported_item.r_date_of_loss BETWEEN ? AND ?
    AND reported_item.r_name REGEXP ?
   AND reported_item.r_reported_as='f'
    GROUP BY reported_item.r_id  ${sort_part_of_query}
    )
    SELECT COUNT(*)  AS total_found_filtered_items FROM (
     SELECT * FROM common_reported_item
     ) as all_searched_reported_item;`;
   //Lost and found query parameters
   queryToExecute.count_query_params = [categoriesList, fromDate, toDate, searchQueryRegex,];
  } else if (searchQuery != undefined && categoriesList != undefined) {
   //NEVER OCCURS
```

```
//Main query
queryToExecute.query = `SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers
FROM reported_item
JOIN reported_image ON reported_item.r_id = reported_image.r_id
JOIN category ON reported_item.category_id = category.cat_id
JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id
WHERE reported_item.category_id IN (?)
AND r_name REGEXP ?
GROUP BY reported_item.r_id ${sort_part_of_query}
LIMIT ? OFFSET ?;`;
//Main query parameters
queryToExecute.params = [categoriesList, searchQueryRegex, limitToFetchItems, offset,];
//Lost items Count query
queryToExecute.lost_items_count_query = `WITH common_reported_item AS (
SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers
FROM reported_item
JOIN reported_image ON reported_item.r_id = reported_image.r_id
JOIN category ON reported_item.category_id = category.cat_id
JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id
WHERE reported_item.category_id IN (?)
AND reported_item.r_name REGEXP ?
AND reported_item.r_reported_as='l'
GROUP BY reported_item.r_id ${sort_part_of_query}
```

```
    )
      SELECT COUNT(*)  AS total_lost_filtered_items FROM (
     SELECT * FROM common_reported_item
     ) as all_searched_reported_item;`;
   //Found items query
   queryToExecute.found_items_count_query = `WITH common_reported_item AS (
    SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers
    FROM reported_item
    JOIN reported_image ON reported_item.r_id = reported_image.r_id
    JOIN category ON reported_item.category_id = category.cat_id
     JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id
   WHERE reported_item.category_id IN (?)
    AND reported_item.r_name REGEXP ?
   AND reported_item.r_reported_as='f'
    GROUP BY reported_item.r_id  ${sort_part_of_query}
    )
    SELECT COUNT(*)  AS total_found_filtered_items FROM (
     SELECT * FROM common_reported_item
     ) as all_searched_reported_item ;`;
   //Lost and found query parameters
   queryToExecute.count_query_params = [categoriesList, searchQueryRegex];
  } else if (
   searchQuery != undefined &&
   toDate != undefined &&
   fromDate != undefined
  ) {
```

//Occurs when the user searches and fromDate and toDate is selected without the categories being selected.

//Main query

queryToExecute.query = `SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_answers

FROM reported_item

JOIN reported_image ON reported_item.r_id = reported_image.r_id

JOIN category ON reported_item.category_id = category.cat_id

JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id

WHERE reported_item.r_date_of_loss BETWEEN ? AND ?

AND r_name REGEXP ?

GROUP BY reported_item.r_id ${sort_part_of_query}

LIMIT ? OFFSET ?;`;

//Main query parameters

queryToExecute.params = [fromDate, toDate, searchQueryRegex, limitToFetchItems,offset,];

//Lost items Count query

queryToExecute.lost_items_count_query = `WITH common_reported_item AS (

SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_answers

FROM reported_item

JOIN reported_image ON reported_item.r_id = reported_image.r_id

JOIN category ON reported_item.category_id = category.cat_id

JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id

WHERE reported_item.r_date_of_loss BETWEEN ? AND ?

```
AND r_name REGEXP ?
AND reported_item.r_reported_as='l'
  GROUP BY reported_item.r_id  ${sort_part_of_query}
  )
  SELECT COUNT(*)  AS total_lost_filtered_items FROM (
   SELECT * FROM common_reported_item
   ) as all_searched_reported_item
  ;`;
  //Found items query
  queryToExecute.found_items_count_query = `WITH common_reported_item AS (
  SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers
  FROM reported_item
  JOIN reported_image ON reported_item.r_id = reported_image.r_id
  JOIN category ON reported_item.category_id = category.cat_id
   JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id
  WHERE reported_item.r_date_of_loss BETWEEN ? AND ?
AND r_name REGEXP ?
AND reported_item.r_reported_as='f'
  GROUP BY reported_item.r_id  ${sort_part_of_query}
  )

  SELECT COUNT(*)  AS total_found_filtered_items FROM (
   SELECT * FROM common_reported_item
   ) as all_searched_reported_item
  ;`;
  //Lost and found query parameters
```

```
    queryToExecute.count_query_params = [fromDate, toDate, searchQueryRegex,
    ];
  } else if (searchQuery != undefined) {
    //NEVER OCCURS
    //Main query
    queryToExecute.query = `SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers
    FROM reported_item
    JOIN reported_image ON reported_item.r_id = reported_image.r_id
    JOIN category ON reported_item.category_id = category.cat_id
     JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id
    WHERE r_name REGEXP ?
    GROUP BY reported_item.r_id  ${sort_part_of_query}
  LIMIT ? OFFSET ?;`;
    //Main query parameters
    queryToExecute.params = [searchQueryRegex, limitToFetchItems, offset];
    //Lost items Count query
    queryToExecute.lost_items_count_query = `WITH common_reported_item AS (
      SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers
      FROM reported_item
      JOIN reported_image ON reported_item.r_id = reported_image.r_id
      JOIN category ON reported_item.category_id = category.cat_id
       JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id
      WHERE reported_item.r_name REGEXP ?
```

```
  AND reported_item.r_reported_as='l'
    GROUP BY reported_item.r_id  ${sort_part_of_query}
    )
    SELECT COUNT(*)  AS total_lost_filtered_items FROM (
     SELECT * FROM common_reported_item
     ) as all_searched_reported_item;`;
  //Found items query
  queryToExecute.found_items_count_query = `WITH common_reported_item AS (
    SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers
    FROM reported_item
    JOIN reported_image ON reported_item.r_id = reported_image.r_id
    JOIN category ON reported_item.category_id = category.cat_id
     JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id
    WHERE reported_item.r_name REGEXP ?
  AND reported_item.r_reported_as='f'
    GROUP BY reported_item.r_id  ${sort_part_of_query}
    )
    SELECT COUNT(*)  AS total_found_filtered_items FROM (
     SELECT * FROM common_reported_item
     ) as all_searched_reported_item

  //Lost and found query parameters
  queryToExecute.count_query_params = [searchQueryRegex];
} else if (
  categoriesList != undefined &&
  toDate != undefined &&
```

```
    fromDate != undefined
  ) {
    //Main query
    queryToExecute.query = `SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers
    FROM reported_item
    JOIN reported_image ON reported_item.r_id = reported_image.r_id
    JOIN category ON reported_item.category_id = category.cat_id
    JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id
    WHERE reported_item.category_id IN (?) AND reported_item.r_date_of_loss BETWEEN ? AND ? GROUP BY reported_item.r_id ${sort_part_of_query}
    LIMIT ? OFFSET ?`;
    //Main query params
    queryToExecute.params = [categoriesList, fromDate, toDate, limitToFetchItems, offset,];
    //Lost filtered items Query (to count all the lost items in the query)
    queryToExecute.lost_items_count_query = `SELECT COUNT( DISTINCT reported_item.r_id) AS total_lost_filtered_items FROM reported_item,reported_image,category WHERE reported_item.r_id= reported_image.r_id AND reported_item.category_id=category.cat_id AND reported_item.category_id IN (?) AND reported_item.r_date_of_loss BETWEEN ? AND ? AND reported_item.r_reported_as='l' ${sort_part_of_query}; `;
    //Found filtered items Query (to count all the found items in the query)
    queryToExecute.found_items_count_query = `SELECT COUNT( DISTINCT reported_item.r_id) AS total_found_filtered_items FROM reported_item,reported_image,category WHERE reported_item.r_id= reported_image.r_id AND reported_item.category_id=category.cat_id AND reported_item.category_id IN (?) AND reported_item.r_date_of_loss BETWEEN ? AND ? AND reported_item.r_reported_as='f' ${sort_part_of_query}; `;
    //Count query params
```

```
    queryToExecute.count_query_params = [categoriesList, fromDate, toDate];
  } else if (categories != undefined) {
    //no dates given in query
    //this condition should never occur
    //Main query
    queryToExecute.query = `SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers
    FROM reported_item
    JOIN reported_image ON reported_item.r_id = reported_image.r_id
    JOIN category ON reported_item.category_id = category.cat_id AND reported_item.category_id IN (?)
    JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id
    GROUP BY reported_item.r_id ${sort_part_of_query}
    LIMIT ? OFFSET ?`;
    //params
    queryToExecute.params = [categoriesList, limitToFetchItems, offset];
    //Lost filtered items Query (to count all the lost items in the query)
    queryToExecute.lost_items_count_query = `SELECT COUNT( DISTINCT reported_item.r_id) AS total_lost_filtered_items FROM reported_item,reported_image,category WHERE reported_item.r_id= reported_image.r_id AND reported_item.category_id=category.cat_id AND reported_item.category_id IN (?) AND reported_item.r_reported_as='l' ${sort_part_of_query} ;`;
    //Found filtered items Query (to count all the lost items in the query)
    queryToExecute.found_items_count_query = `SELECT COUNT( DISTINCT reported_item.r_id) AS total_found_filtered_items FROM reported_item,reported_image,category WHERE reported_item.r_id= reported_image.r_id AND reported_item.category_id=category.cat_id AND reported_item.category_id IN (?) AND reported_item.r_reported_as='f' ${sort_part_of_query} ;`;
    //count query params
```

```
    queryToExecute.count_query_params = [categoriesList];
  } else if (toDate != undefined && fromDate != undefined) {
    //no categories given in query
    //Main QUERY
    queryToExecute.query = `SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers
    FROM reported_item
    JOIN reported_image ON reported_item.r_id = reported_image.r_id
    JOIN category ON reported_item.category_id = category.cat_id AND reported_item.r_date_of_loss BETWEEN ? AND ?
    JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id
    GROUP BY reported_item.r_id ${sort_part_of_query}
    LIMIT ? OFFSET ?`;
    //Main query parars
    queryToExecute.params = [fromDate, toDate, limitToFetchItems, offset];

    //Lost filtered items Query (to count all the lost items in the query)
    queryToExecute.lost_items_count_query = `SELECT COUNT(DISTINCT reported_item.r_id) as total_lost_filtered_items FROM reported_item JOIN reported_image ON reported_item.r_id = reported_image.r_id JOIN category ON reported_item.category_id = category.cat_id AND reported_item.r_date_of_loss BETWEEN ? AND ? AND reported_item.r_reported_as='l' ${sort_part_of_query};`;
    //Lost filtered items Query (to count all the lost items in the query)
    queryToExecute.found_items_count_query = `SELECT COUNT(DISTINCT reported_item.r_id) as total_found_filtered_items FROM reported_item,reported_image,category WHERE reported_item.r_id= reported_image.r_id AND reported_item.category_id=category.cat_id AND reported_item.r_date_of_loss BETWEEN ? AND ?  AND reported_item.r_reported_as='f' ${sort_part_of_query};`;
    //Count query params
```

```
    queryToExecute.count_query_params = [fromDate, toDate];
  } else {
    //default fetch query..However this condition should never occur as the dates are always received in the query params from the frontend
    //It fetches all the queries
    //Main query
    queryToExecute.query = `SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name ,GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|' ) as sqa_answers
    FROM reported_item
    JOIN reported_image ON reported_item.r_id = reported_image.r_id
    JOIN category ON reported_item.category_id = category.cat_id
    JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id
    GROUP BY reported_item.r_id ${sort_part_of_query} LIMIT ? OFFSET ?`;

    //Main query params
    queryToExecute.params = [limitToFetchItems, offset];

    //Lost filtered items Query (to count all the lost items in the query)
    queryToExecute.lost_items_count_query = `SELECT COUNT( DISTINCT reported_item.r_id) as total_lost_filtered_items FROM reported_item,reported_image,category WHERE reported_item.r_id= reported_image.r_id AND reported_item.category_id=category.cat_id AND reported_item.r_reported_as='l' ${sort_part_of_query};`;
    //Lost filtered items Query (to count all the lost items in the query)
    queryToExecute.found_items_count_query = `SELECT COUNT( DISTINCT reported_item.r_id) as total_found_filtered_items FROM reported_item,reported_image,category WHERE reported_item.r_id= reported_image.r_id AND reported_item.category_id=category.cat_id AND reported_item.r_reported_as='f' ${sort_part_of_query};`;
```

```
  //Count query params
  queryToExecute.count_query_params = null;
 }
 /*----------------------------------------------------------------------
FETCH  20 ITEMS of applying filters AFTER DETERMINING THE QUERY FROM
ABOVE
----------------------------------------------------------------------*/
 if (
  queryToExecute.query.length > 0 &&
  queryToExecute.params.length > 0 &&
  queryToExecute.lost_items_count_query.length > 0 &&
  queryToExecute.found_items_count_query.length > 0 &&
  (queryToExecute.count_query_params == null ||
   queryToExecute.count_query_params.length > 0)
 ) {
  //fetch the filtered items
  [fetch_filtered_items_result] = await conn.query(
   queryToExecute.query,
   queryToExecute.params
  );

  //fetch the filtered lost items count
  [fetch_fitered_lost_items_count_result] = await conn.query(
   queryToExecute.lost_items_count_query,
   queryToExecute.count_query_params
  );

  //fetch the filtered found items count
  [fetch_fitered_found_items_count_result] = await conn.query(
```

```
    queryToExecute.found_items_count_query,
    queryToExecute.count_query_params
  );
 } else {
  conn.release();
  throw new BadRequestError(
    "No query provided. Please check the queryToExecute variable"
  ); }
 //CHECK IF THE FETCH FAILED
 if (fetch_filtered_items_result == undefined) {
  conn.release();
  throw new BadRequestError(
    "No results after filtering. Please check filtering logic"
  );
 }
 //EXTRACT THE COUNT OF LOST FILTERED ITEMS AND FOUND FILTERED ITEMS
 if (fetch_fitered_lost_items_count_result.length > 0) {
  totalLostReportedItems =
    fetch_fitered_lost_items_count_result[0].total_lost_filtered_items;
 }
 if (fetch_fitered_found_items_count_result.length > 0) {
  totalFoundReportedItems =
    fetch_fitered_found_items_count_result[0].total_found_filtered_items;
 }
 //RETURN THE RESPONSE
 if (fetch_filtered_items_result.length > 0) {
   //sort the items according to lost and found and get a single map/object of lost items and found items
   var lostAndFoundItemsObject = sortReportedItemsIntoLostAndFound(
```

```
    fetch_filtered_items_result
  );
  //get the total number of items that are to be sent in the request to check at the flutter's frontend whether all the items in the database for that filter are exhausted or should it fetch more items when scrolled to the bottom of the grid view list
  var totalNoofItemsToBeSentInResponse =
    lostAndFoundItemsObject.lostItems.length +
    lostAndFoundItemsObject.foundItems.length;
  //release the connection
  conn.release();
  //return the response
  return res.status(StatusCodes.OK).json({
    ...lostAndFoundItemsObject,
    no_of_items: totalNoofItemsToBeSentInResponse,
    total_lost_reported_items: totalLostReportedItems,
    total_found_reported_items: totalFoundReportedItems,
  });
 } else {
  conn.release();
  return res
    .status(StatusCodes.OK)
    .json({ msg: "No reported items present" });
  //throw new BadRequestError("No reported items present");
 }};
```

5.2.2 Coding Efficiency

On the frontend for Flutter, I have used the MVVM architecture pattern to structure my code. MVVM is an architectural design pattern that separates an application into three interconnected components: Model, View, and ViewModel. the Model represents the data, the View corresponds to the user

interface, and the ViewModel acts as a bridge between the Model and View, managing the business and presentation logic. This separation enhances code organization, testability, and maintainability. In Flutter, providers like Provider can be used to implement the MVVM pattern efficiently. Also, I use a well-defined file structure while represents the MVVM pattern that I have used. The file structure includes:-

- Assets- contains all the animations and images
- Config- contains the configuration files for the app.
- Constants – the various constants that are used throughout the app.
- Models- It provides the structure for the data coming from the database.
- Services – It contains the functions that interact with the database
- Utils and singletons – all the common functions such as date formatting, text formatting along with Shared Preferences management,etc reside here.
- View – contains subfolders such as common_widgets, user views,admin views,etc. It holds the UI files.
- ViewModels- These files handle the core business and presentation logic.

On the backend, a similar modular approach is followed again with a well-defined file structure. The files are modularized into routers which handle the routing of the incoming request to the function that is to be executed and controllers that hold the functions to be executed. The file structure is as follows:-

- Assets- holds all the image files
- Controllers – contains the functions that hold the core logic
- Custom errors- it contains the custom errors.
- Db_socket_connection – contains the database connectivity files and the Socket.IO configuration files.
- Middlewares- contains all the functions that are executed between the router functions and the controller functions. These include authentication middleware which authenticates each incoming request before executing its associated controller function.
- Routes – these are files that route the incoming HTTP requests with the function to be executed that process the incoming request.
- Utils – contain files such as email templates and functions that are used for formatting, generating OTP,etc

- .env file- it includes the passwords of the database. The main purpose of this file is to never enter the password directly in the code. Rather this file is accessed globally and assigns a variable to the password, hence the variable is accessed but the plaintext password is never exposed. This file is never shared with anyone and is rather uploaded directly on the server.

Additionally, Git and GitHub are used to backup the code and provide version management. This means that if something goes wrong, we can always revert back to a previous version of the code. Multiple branches are used namely “beta-1” branch that holds all the recent changes that have been made, whereas the “main” branch contains the code that is tested and stable. Once the code in “beta-1” branch is tested successfully, it is merged with the “main” branch. This helps to effectively manage various versions of the code and track the changes over time.

5.3 Testing approaches

Here I have used manual testing technique. Manual testing is a type of software testing in which testers manually execute test cases without the use of automation tools. This involves a human tester performing a set of predefined steps and observations to evaluate the functionality, usability, and performance of a software application. Manual testing is often used in the early stages of the development cycle and is an effective way to identify issues and defects that may have been missed during automated testing or development. It can also provide valuable feedback on the user experience and overall quality of the application

5.3.1 Unit testing

Unit testing is a type of software testing in which individual units or components of a software application are tested in isolation from the rest of the system to ensure they are functioning as intended. This involves writing and executing test cases for each unit of code to validate that it meets its specifications and produces the expected output. It is an important practice for ensuring the reliability, maintainability, and overall quality of software applications

TEST CASES:

Test case no.	Test case	Expected outcome	Actual outcome	Remarks
1.	**Student registration** First name: Aaditya Middle name: Sheelkumar Last name: Pal Control id: 1111111111 Phone: 9999888877 Email: chocolateassignment68@gmail.com	Not a current student of Vaze college.	Not a current student of Vaze college.	Pass
2.	**Student registration** First name: Aaditya Middle name: Sheelkumar Last name: Pal Control id: 2021080314 Phone: 9999888877 Email: abc@hh.com	Please enter a valid email address	Please enter a valid email address	Pass
3.	**Student registration** First name: Aaditya Middle name: Sheelkumar Last name: Pal Control id: 2021080314 Phone: 9999888877 Email: iconicronaldo0@gmail.com.	Student with this email is already registered.	Student with this email is already registered.	Pass

4.	**Student registration** First name: Aaditya Middle name: Sheelkumar Last name: Pal Control id: 2021080314 Phone: 9999888877 Email: chocolateassignment68@gmail.com	Proceed to Enter OTP page.	Proceed to Enter OTP page.	Pass
5.	**Student registration** First name: Aaditya Middle name: Sheelkumar Last name: Pal Control id: 2021080314 Phone: 9999888877 Email: chocolateassignment68@gmail.com	Student with this Control ID is already registered.	Student with this Control ID is already registered.	Pass
6.	**Faculty registration** First name: Rakhee Middle name: Last name: Rane Phone: 9999888877 Email: iconicronaldo0@gmail.com	Please enter the email provided by the college.	Please enter the email provided by the college.	Pass
7.	**Faculty registration** First name: Rakhee Middle name: Last name: Rane Phone: 9999888877 Email: rakheerane@vazecollege.net	Proceed to enter OTP page.	Proceed to enter OTP page.	Pass

8.	**Faculty registration** First name: Rakhee Middle name: Last name: Rane Phone: 9999888877 Email: rakheerane@vazecollege.net (already registered email)	Faculty with this email is already registered.	Faculty with this email is already registered .	Pass
9.	**Faculty registration** First name: Rakhee Middle name: Last name: Rane Phone: 9999888877 Email: rakheerane@vazecollege.com	Please enter a valid email address	Please enter a valid email address	Pass
10.	**Enter OTP Page** OTP: 1234	Please enter a valid OTP	Please enter a valid OTP	Pass
11.	**Enter OTP Page** OTP: 111111	Please enter a valid OTP	Please enter a valid OTP	Pass
12.	**Enter OTP Page** OTP: a correct OTP	Go to Set/Reset Password page	Go to Set/Reset Password page	Pass
13.	**Set/Reset Password Page** Password: 1234 Confirm Password:	Password should be minimum of 8 characters in length, should at least contain one uppercase, one lowercase and one digit.	Password should be minimum of 8 characters in length, should at least contain one uppercase , one lowercase	Pass

			and one digit.	
14.	**Set/Reset Password Page** Password: AAAAAAAA **OR** 111111111 **OR** aaaaaaaa Confirm Password:	Password should be minimum of 8 characters in length, should at least contain one uppercase, one lowercase and one digit.	Password should be minimum of 8 characters in length, should at least contain one uppercase, one lowercase and one digit.	Pass
15.	**Set/Reset Password Page** Password: Aa1234567 Confirm Password: 1234567Aa	The passwords don't match.	The passwords don't match.	Pass
16.	**Set/Reset Password Page** Password: Aa1234567 Confirm Password: Aa1234567	Continue to Registration/Resetting password.	Continue to Registration/Resetting password.	Pass
17.	**Student/Faculty Login** Email: example@example.com Password: Aa1234567	This user is not registered	This user is not registered	Pass
18.	**Student/Faculty Login** Email: rakheerane@vazecollege.net Password: Abc1234567	Wrong password. Please check your password.	Wrong password. Please check your password	Pass

19.	**Student/Faculty Login** Email: chocolateassignment68@gmail.com Password: Aa1234567	Login successful	Login Unsuccessful.	Fail. Please provide your registered email id
20.	**Forgot password** Email: example@gmail.com	Couldn't find anyone with the provided email.	Couldn't find anyone with the provided email.	Pass
21.	**Forgot password** Email: chocolateassignment68@gmail.com OTP: 111111	Please enter a valid OTP	Please enter a valid OTP	Pass
22.	**Forgot password** Email: chocolateassignment68@gmail.com OTP: received OTP in mail	To **Set/Reset Password page**	To Set/Reset Password page	Pass
23.	**View Items** If no items are reported	No items yet	No items	Pass
24.	**View Items** If items are reported	Display the items. The items should lazy load meaning that it should load only 20 items at a time	No items found	Fail. The MySQL query doesn't return any rows even though there is data present

25.	**Search Items** Search Query: Test1	Display the items that contain the search query's character. If no items are present then display "No results found"	Displaying the items if present. If no items are present then "No results found" message is displayed	Pass
26.	**Sort Items** The options to sort are. Selecting one at a time: - ● "Newest" ● "Oldest" ● "A-Z" ● "Z-A"	● If "Newest" is selected then the items should be arranged in descending order of their lost/found date. ● If "Oldest" is selected then items should be arranged in ascending order of their lost/found date. ● If "A-Z" is selected then the items should be displayed in the alphabetical order ● If "Z-A" is selected then the items	The items are sorted as required	Pass

		should be displayed in the reverse alphabetical order.		
27.	**Filter items according to Category** Options include: ● Electronics ● Clothing ● Jewelry ● Stationary ● Riding gear ● Other	The items are displayed based on the selected filter option	JWT token isn't valid	Failed. The JWT token expired after 30 days of user creating the account
28.	**Filter items according to Date Posted** From Date: 2/11/23	The items reported between the selected date range are displayed.	The filtered list of items is displayed	Pass
29.	**Report Item** (Current date is 1/12/23) Images: img1.png, img2.png.img3.jpeg Name: Blue bottle Description: Milton blue bottle Date lost: 20/10/2023 Time: 9:00 AM Location: Canteen Category: Other Radio button selected: I lost it Security QA1: What is the color? Answer: Blue Security QA2: Which brand? Answer: Milton	Please enter a valid date	Please enter a valid date	Pass

	Security QA3: What is the color of the cap? Answer: Red			
30.	**Report Item** (Current date is 1/12/23) Images: img1.png, img2.png.img3.jpeg Name: Blue bottle Description: Milton blue bottle Date lost: 30/11/23 Time: 1:00 AM Location: Canteen Category: Other Radio button selected: I lost it Security QA1: What is the color? Answer: Blue Security QA2: Which brand? Answer: Milton Security QA3: What is the color of the cap? Answer: Red	Please enter a valid time	Please enter a valid time	Pass
31.	**Report Item** Images: img1.png, img2.png.img3.jpeg Name: Blue bottle Description: Milton blue bottle	Item reported successfully	Item reported successfully	Pass

	Date lost: 30/11/23 Time: 10:00AM Location: Canteen Category: Other Radio button selected: I lost it Security QA1: What is the color? Answer: Blue Security QA2: Which brand? Answer: Milton Security QA3: What is the color of the cap? Answer: Red			
32.	**Report Item** Images: img1.png, img2.png.img3.jpeg Name: null Description: Milton blue bottle Date lost: null Time: 11:00:00 am Location:null Category: Other Radio button selected:null Security QA1: What is the color? Answer: Blue Security QA2: Which brand? Answer: Milton Security QA3: What is the color of the cap? Answer: Red	Please enter all the required details.	Please enter all the required details.	Pass

33.	**View Profile**	Fetch the profile details of the logged in user	Couldn't fetch the details of the user. Error at the output line (^1)	Failed. The details of the logged in user are not fetched due to some error when the network connectivity is poor. The page continuously keeps refreshing which is gives an annoying user experience.
34.	**Edit Profile** Email: iconicronaldo0@gmail.com	To **Enter OTP page**	To **Enter OTP page**	Pass
35.	**Edit Profile – update email** Email: abc12@hh.com	Please enter a valid email address.	Please enter a valid email address	Pass
36.	**Edit Profile – update email** OTP: OTP received in email	Email Updated successfully	Email Updated successfully	Pass
37.	**Edit Profile – update phone** Phone: 9999998888	Phone updated successfully	Phone updated successfully	Pass
38.	**Edit Profile – update password** Password: Aa1234567 Confirm Password: Aa12345679	The passwords don't match	The passwords don't match	Pass
39.	**Edit Profile – update password** Password: Aaa12345678 Confirm Password: Aaa12345678	Password updated successfully	Password updated successfully	Pass
40.	**Edit Item details** Same as report items	Same as report items	File couldn't be deleted from	Fail. The images that were uploaded earlier by the user were not

			Google Drive	successfully deleted.
41.	**Update progress of items** When the 'update progress' button is clicked then options to select the current progress is shown. Options include: ● Found ● Owner found	The status is updated accordingly	When the lost item's progress is updated to "found" and the found item's progress is updated to "owner found" successful	Pass
42.	**Delete multiple Items** The selected items need to be deleted	Items deleted successfully	The selected items are not actually deleted.	Failed
43.	**Delete a Single Item** The selected items need to be deleted	Item deleted successfully	Item deleted successfully	Pass
44.	**Fetch all chats**	All the chats of the user are fetched showing the latest message date and time	The chats are fetched but the latest date and time are not fetched	Failed. The latest message and date is not fetched even though the messages between the users are present.
45.	**Send messages** Sent message to Lost admin: hello from Aaditya	The message is sent and visible to "Lost Admin" user	The message is sent and visible to "Lost Admin" user	Pass

Table 5.1 Test cases

5.3.2 Modifications and Improvements

- **Test case 19**:

  The login for the user failed due to the spelling mismatch in the JSON “user_email” key.

  Solution:

  In authentication_service.dart file => loginUserServiceFunction

  - Map<String, dynamic> userLoginData = {
  - "**user_email**": email,
  - "user_password": password,
  - };

19.	**Student/Faculty Login** Email: chocolateassignment68@gmail.com Password: Aa1234567	Login successful	Login successful	Pass

Table 5.2 Updated Test Case 19

- **Test case 24**:

  The MySQL SELECT query doesn’t return the rows even though there is data present

  In the backend, in reported_items_controller.js file, fetchAllItems function, the new SQL queries should include GROUP_CONCAT() function and a GROUP BY clause as follows. This is also applicable to the fetchReportedItemsOfUser().

  Solution:

  New SQL Query code:

  SELECT reported_item.*, GROUP_CONCAT(DISTINCT reported_image.r_image) AS images, category.cat_name, GROUP_CONCAT(DISTINCT security_qa.sqa_question ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_questions, GROUP_CONCAT(DISTINCT security_qa.sqa_answer ORDER BY security_qa.sqa_question SEPARATOR '|~|') as sqa_answers

  FROM reported_item

  JOIN reported_image ON reported_item.r_id = reported_image.r_id

  JOIN category ON reported_item.category_id = category.cat_id

  JOIN security_qa on security_qa.sqa_r_id= reported_item.r_id

  WHERE reported_item.category_id IN (?) AND reported_item.r_date_of_loss BETWEEN ? AND ?

  AND reported_item.r_name REGEXP ?

```
    GROUP BY reported_item.r_id  ${sort_part_of_query}
  LIMIT ? OFFSET ?;`;
    //Main query parameters
    queryToExecute.params = [categoriesList, fromDate, toDate, searchQueryRegex, limitToFetchItems, offset];
```

24.	**View Items** If items are reported	Display the items. The items should lazy load meaning that it should load only 20 items at a time	The items are displayed correctly	Passed

Table 5.3 Updated Test Case 24

- **Test case 40 and 42**:

  The images of the reported item are not deleted from the Google drive hence causing an issue. Google drive returned an unauthorized error.

  In the backend code, in google_drive_api_controller.js=> delete

  Solution:

  The "auth" key was missing

```
//2. Delete images from google drive of that user and that item temporarily ie put in trash
const deleteImageOfReportedItemFromgDrive= async(imageFileIdToDelete)=>{
  try {
    const gdrive= google.drive({version: 'v3', auth:googleAuth });
    }
//Rest of code
}
```

40.	**Edit Item details** Same as report items	Same as report items	Item details updated successfully	Pass
42.	**Delete multiple Items** The selected items need to be deleted	Items deleted successfully	The items are deleted successfully	Pass

Table 5.4 Updated Test Case 40 and 42