Req-001: createUser API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will create a new user and store it in db.

- ➔ Method type: **POST.**

- ➔ **API:**
  - name: **createUser**
  - **api/users/createUser**

- ➔ DB Bucket to be used: **userInfo** and **userImage**.

- ➔ Path parameters should be empty.
- ➔ Mandatory parameters to be sent as part of the request body: firstName, lastName, userName, emailId, and password.
- ➔ Payload must be verified if all the mandatory parameters are available or not before proceeding with the create user functionality.
- ➔ "password" needs to be encoded using bcrypt crypto with saltRounds before storing it into the db.
- ➔ A random "verificationCode" needs to be generated and sent to the provided emailId via mail for verification of the user.
- ➔ Generated "verificationCode" needs to be stored as a value of the parameter "verificationCode" in the user object.
- ➔ On successful creation of a new customer the verification email should be sent to the provided emailId for the verification of the customer.
- ➔ Create a default user profile image, user basic finance details, and user dashboard setting records in the database.

- ➔ **Response messages:**
  - Response message should be "201 Created" in case the operation is successful.
  - Response message should be "400 Bad Request" in case any required parameter is missing.
  - Response message should be "400 Bad Request" in case of any error occurring while saving the user in db.
  - Response message should be "409 Conflict" in case emailId or userName is duplicated.
  - Response message should be "500 Internal Server Error" in case any other error occurs in the entire operation.

- ➔ **Email Functionality:**
  - The "verificationLink" which is sent to the user as part of the verification email should contain three parameters: userId, time of creation, and verificationCode.
  - The "verificationLink" should be of below form:
    api/users/verify/{userId}/{time}/{verificationCode}
  - The Email template should contain the following text in the same order as provided.

Subject: “Welcome to dailyFinance - Verify Your Email and Activate Your Account”

“Dear [User name],”

“We are thrilled to welcome you to dailyFinance! Thank you for choosing us as your trusted platform. Your journey toward financial empowerment and success begins here.”

“To ensure the security and integrity of your account, we kindly ask you to verify your email address and activate your account. Please follow the simple steps below:”

“**Step 1: Verify Your Email**”

- “Click on the following verification link to confirm your email address:”
  [Verification Link]

“**Step 2: Activate Your Account**”

- “After verifying your email, you can log in to your account and start exploring all the powerful features that dailyFinance offers.”

“With dailyFinance, you can:”

- “Track your daily expenses and income.”
- “Interactive Reports and Charts.”
- “User-Friendly Interface”
- “Data Export and Backup”
- “And so much more!”

“If you didn't register for an account with us, please ignore this email.”

“If you encounter any issues during the verification process, feel free to contact us at shadow.works.1998@gmail.com.”

“We're excited to have you join our community of financially savvy users. Together, we can help you achieve your financial goals and make smart money decisions.”

“Thank you for choosing dailyFinance, and welcome aboard!”

“Best regards,”

“Team dailyFinance”

Req-002: verifyUser API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will activate the user's account by verifying the verificationCode sent in the request.

➔ Method type: **GET**.

➔ **API:**
- name: **verify**
- **api/users/verify/{verificationLink}**

➔ DB Bucket to be used: **userInfo**.

➔ Parameters to be sent as part of the path: userId, time of creation, and verificationCode.

➔ Request body should be empty.

➔ Payload must be verified if user id is present or not before proceeding with verifying user functionality.

➔ Verify if the time at which the verification request is received and the "time" parameter, which is received as part of the path, is within six hours or not.

➔ If the above validation is successful then match the verification code provided as part of the payload with the verification code that is stored in db against the user for which the id is provided as part of the path of an api.

➔ If the above validation is also successful then update the value of "isValidated" parameter as true for the requested user, and send the successful validation mail to the user. And if the above validation fails then send the new verification mail to the user.

➔ **Response messages:**
- Response message should be "200 Ok" in case user verification is successful.
- Response message should be "400 Bad Request" in case the verification request time is beyond six hours or verificationCode did not match.
- Response message should be "404 Not Found" in case the data for requested userId does not exist.
- Response message should be "500 Internal Server Error" in case of any error occurred in storing the data in db.

➔ **Verification successful/failed message html page:**
- The following html template to be displayed on the web in case the verification is successful. The text needs to be in the same order as provided.
  "Congrats! You're Officially a Member of financeTracker."
  "Thanks for joining us."
  "Regards, Team financeTracker"

- The following html template to be displayed on the web in case the verification failed. The text needs to be in the same order as provided.
  "Verification Code Expired!"
  "Please relogin and get a new verification code to activate your account."

“Note: Your account may have already been verified. Please try to login to the portal. If you're not authorized, you'll get a new verification code to activate your account.”
“Regards, Team financeTracker”

➔ **Email Functionality:**

- The Email to be sent if the verification is successful, and the template should contain the following text in the same order as provided:
  Subject: “Account Verified - Welcome to dailyFinance”
  “Dear [User name],”
  “Congratulations! Your account has been successfully verified and is now active. Welcome to dailyFinance - your gateway to financial empowerment.”
  “We're thrilled to have you as a member of our community. As a verified user, you can now access all the amazing features and benefits that dailyFinance offers. Here are some of your basic details for your reference:”
  “**First Name:** [User's First Name]”
  “**Last Name:** [User's Last Name]”
  “**Username:** [User's Username]”
  “**Contact Number:** [User's Contact Number]”
  “**Email ID:** [User's Email Address]”
  “With your account now verified, you can start making the most of our platform right away. Whether you want to track your daily expenses, track investment, register income, or work toward your financial goals, we've got you covered.”
  “Need assistance or have questions? Don't hesitate to reach out to us at shadow.works.1998@gmail.com.”
  “Thank you for trusting dailyFinance as your financial partner. We look forward to supporting you on your financial journey.”
  “Best regards,”
  “Team dailyFinance”

Req-003: userLogin API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will validate the credentials entered by the user and return the token if credentials are correct.

- ➔ Method type: **POST**.

- ➔ **API:**
  - name: **userLogin**
  - **api/users/userLogin**

- ➔ DB Bucket to be used: **userInfo**.

- ➔ Path parameters should be empty.
- ➔ Mandatory parameters to be sent as part of the request body: userName, and password.
- ➔ Payload must be verified if all the mandatory parameters are available or not before proceeding with the login user functionality.
- ➔ Find the correct user record in db based on entered userName, then match the user entered password with the stored password to check if the credentials are correct or not. If the credentials are correct then check if the user is verified or not.
  - If the user is a verified and active user then generate the token and return the token as a response along with userId and userName.
  - If the user is not verified then generate the verification code and send it to the provided emailId via mail for verification of the user.
  - If the user is a deactivated user then reactivate and login the user and send a mail to the user.
- ➔ Use the secret key to generate the token with an expiration time of 1 hour.
- ➔ Update the login count and last login time of the user.

- ➔ **Response messages:**
  - Response message should be “200 Ok” in case the verification is successful and return the token as a response.
  - Response message should be “201 Created” in case the verification is required and email is sent.
  - Response message should be “400 Bad Request” in case the required parameters are missing.
  - Response message should be “401 Unauthorized” in case the users entered credentials are incorrect.
  - Response message should be “404 Not Found” in case the required user is not found.
  - Response message should be “500 Internal Server Error” in case any other error occurred.

- ➔ **Email Functionality:**
  - The following email to be sent when user account has been reactivated, and the template should contain the following text in the same order as provided:
    Subject: “Welcome Back! Your Account is Reactivated.”

“Dear [User name],”
“We are thrilled to welcome you back to dailyFinance! Your account has been successfully reactivated, and we're excited to have you as part of our community once again.”
“We understand that sometimes circumstances change, and we appreciate your decision to return. We're here to provide you with the best experience possible.”
“Your account is now fully active, and you can enjoy all the benefits of dailyFinancejust like before. Here's a quick reminder of what you can do:”

1. “Expense Tracking: ”
   - “Record and categorize daily expenses to understand where your money is going.”
2. “Income Tracking:”
   - “Log your sources of income to get a clear picture of your financial inflow.”
   - Track salary, freelance earnings, investments, and other income streams.
3. “Interactive Reports and Charts:”
   - “View your financial data through interactive graphs and charts.”
   - “Gain insights into your spending habits and trends over time.”
4. “Savings and Investment Tracking:”
   - “Monitor your savings accounts, investments, and retirement funds.”
   - “Calculate your net worth to see your overall financial health.”
5. “Mobile Accessibility:”
   - “You can access the dailyFinance website on your mobile without any problem.”
6. “User-Friendly Interface:”
   - “Provide an intuitive and easy-to-navigate design for a smooth user experience.”
   - “Customizable dashboards and settings to suit individual preferences.”
7. “Data Export and Backup:”
   - “Allow users to export their financial data or set up automatic backups.”
   - “Ensure data is secure and retrievable in case of system failures.”

“If you have any questions, need assistance, or want to provide feedback, feel free to reach out at shadow.works.1998@gmail.com anytime.”
“Thank you for choosing dailyFinance. We look forward to serving you and making your experience exceptional.”
“Welcome back, and happy exploring!”
“Best regards,”
“Team dailyFinance”

Req-004: validateToken API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will validate the provided token and user id, and return the appropriate response based on verification results if it is successful or not.

The validateToken is an internal API used by other APIs to verify the validity of the token before delivering the API response to the user.

- ➔ Method type: **POST**.

- ➔ **API:**
  - name: **validateToken**
  - **api/users/validateToken**

- ➔ DB Bucket to be used: **userInfo**.

- ➔ Path parameters should be empty.
- ➔ Mandatory parameters to be sent as part of the request body: userId and token.
- ➔ Payload must be verified if all the mandatory parameters are available or not before proceeding with the verify token functionality.
- ➔ Verify the token if the token is correct and not expired.
- ➔ Once the token has been verified then check if the user with the provided user id exists or not.

- ➔ **Response messages:**
  - Response message should be “200 Ok” in case the token verification is successful.
  - Response message should be “400 Bad Request” in case the required parameters are missing.
  - Response message should be “401 Unauthorized” in case the user id or token is not valid.
  - Response message should be “500 Internal Server Error” in case any other error occurred.

Req-005: getUserInfo API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will return the details of the required user using the unique user id present in db.

- ➔ Method type: **GET**.

- ➔ **API:**
  - name: **getUserDetails**
  - **api/users/getUserDetails/{userId}**

- ➔ DB Bucket to be used: **userInfo**.

- ➔ Parameter to be sent as part of the path: userId.
- ➔ Request body should be empty.
- ➔ Token must be sent as part of authorization bearer token.
- ➔ Call a validateToken API internally with the userId and token.
- ➔ If the validateToken API returns 200 response, then proceed to get the records from db for the required user.
- ➔ Fields to be returned as part of the response: userId, firstName, lastName, userName, bio, gender, dob, occupation, emailId, contactNumber, createdOn, lastLogin, and loginCount.

- ➔ **Response messages:**
  - Response message should be “200 Ok” in case the operation is successful.
  - Response message should be “400 Bad Request” in case the userId is missing.
  - Response message should be “404 Not Found” in case the required user does not exist.
  - Response message should be “500 Internal Server Error” in case of any other error occurred.
  - Responses from validateToken API will be returned.

Req-006: updateUserDetails API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will update the information of users using their unique id stored in db.

- ➔ Method type: **PUT**.

- ➔ **API:**
  - name: **updateUserDetails**
  - **api/users/updateUserDetails/{userId}**

- ➔ DB Bucket to be used: **userInfo**.

- ➔ Parameter to be sent as part of the path: userId.
- ➔ Token must be sent as part of authorization bearer token.
- ➔ Parameters that can be sent as part of the request body: firstName, lastName, contactNumber, emailId, bio, userName, gender, and occupation.
- ➔ Call a validateToken API internally with the userId and token.
- ➔ If the validateToken API returns 200 response, then proceed with updating the details of the user in db.
- ➔ If the operation is successful then send the mail to the user with the updated details of the user.

- ➔ **Response messages:**
  - Response message should be “201 Created” in case the operation is successful.
  - Response message should be “400 Bad Request” in case the required parameters are missing.
  - Response message should be “404 Not Found” in case the data for requested userId does not exist.
  - Response message should be “500 Internal Server Error” in case any other error occurred.
  - Responses from validateToken API will be returned.

- ➔ **Email Functionality:**
  - The Email to be sent if the user update details operation is successful, and the template should contain the following text in the same order as provided:
    Subject: “Account Details Successfully Updated”
    “Dear [User name],”
    “We are writing to confirm that your account details have been successfully updated. Thank you for keeping your information current. Here are your updated basic details:”
    “**First Name:** [User's Updated First Name]”
    “**Last Name:** [User's Updated Last Name]”
    “**Username:** [User's Updated Username]”
    “**Email ID:** [User's Updated Email Address]”
    “**Contact Number:** [User's Updated Contact Number]”
    “**Date of Birth (DOB):** [User's Updated Date of Birth]”

“**Bio:** [User’s Updated Bio]”
“**Occupation:** [User’s Updated Occupation]”
“**Account creation date:** [User’s Account creation date]”
“**Last login time:** [User’s Last login time]”
“With your account information now up to date, you can continue to enjoy all the features and benefits of our platform, while ensuring that you receive important notifications and personalized services tailored to your needs.”
“If you have any further updates to make or any questions, please feel free to reach out at shadow.works.1998@gmail.com. We're here to assist you.”
“Thank you for choosing dailyFinance as your trusted financial partner.”
“Best regards,”
“Team dailyFinance”

Req-007: updateUserPassword API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will update the password of users using their unique id stored in db.

- ➔ Method type: **PUT**.

- ➔ **API:**
  - name: **updateUserPassword**
  - **api/users/updateUserPassword/{userId}**

- ➔ DB Bucket to be used: **userInfo**.

- ➔ Parameter to be sent as part of the path: userId.
- ➔ Token must be sent as part of authorization bearer token.
- ➔ Parameters that can be sent as part of the request body: oldPassword, and newPassword.
- ➔ Call a validateToken API internally with the userId, and token.
- ➔ If the validateToken API returns 200 response, then proceed with updating the password of the user in db.
- ➔ Validate if the user's oldPassword is the same as the stored password of the required user or not. If the password is the same then check if the newPassword and oldPassword are not the same. If both the conditions satisfy then update the password.
- ➔ Before updating the password, encode it using bcrypt crypto with saltRounds.
- ➔ Send the user a mail informing them that their password has been successfully updated if the process is successful.

- ➔ **Response messages:**
  - Response message should be "201 Created" in case the operation is successful.
  - Response message should be "400 Bad Request" in case the required parameters are missing.
  - Response message should be "401 Unauthorized" in case the oldPassword mismatched with the stored password for the requested user.
  - Response message should be "404 Not Found" in case the data for requested userId does not exist.
  - Response message should be "500 Internal Server Error" in case any other error occurred.
  - Responses from validateToken API will be returned.

- ➔ **Email Functionality:**
  - The Email to be sent if the user password update operation is successful, and the template should contain the following text in the same order as provided:
    Subject: "Password Updated Successfully"
    "Dear [User's Name],"
    "We're writing to inform you that your password has been successfully updated. If you initiated this change, you can disregard this email. However, if you did not request a

password change, please take immediate action by using our "Forgot Password" functionality to secure your account."

"**If you initiated the password change:**"

"You can rest assured that your account is now protected with your new password. Your account security is of utmost importance to us, and we're here to support you in keeping it safe."

"**If you did not initiate the password change:**"

"We take unauthorized access to your account very seriously. We recommend the following steps:"

1. "**Reset Your Password:** If you didn't request this change, reset your password as soon as possible using the "Forgot Password" link on our login page."
2. "**Review Your Account:** After resetting your password, please review your account details and ensure that no other unauthorized changes have occurred."
3. "**Contact Support:** If you suspect any unusual activity or have concerns about your account's security, please contact at shadow.works.1998@gmail.com. We will assist you in securing your account and investigating any potential issues."

"Your account's security is our top priority, and we are committed to providing you with a safe and secure experience on our platform. Thank you for your vigilance and cooperation in maintaining the integrity of your account."

"Best regards,"

"Team dailyFinance"

Req-008: deactivateUser API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will update the parameter “isDeleted” of the users to deactivate their account, using their unique user id stored in db.

- Method type: **PUT**.

- **API:**
  - name: **deactivateUser**
  - **api/users/deactivateUser/{userId}**

- DB Bucket to be used: **userInfo**.

- Parameter to be sent as part of the path: userId.
- Token to be sent as part of authorization bearer token.
- Parameter to be sent as part of the request body: userName, and password.
- Call a validateToken API internally with the userId and token.
- If the validateToken API returns 200 response, then proceed with deactivating the user.
- Verify the payload to check if the userName, and password is present or not.
- Verify if the entered userName, and password are correct or not for the user which is stored in db with the entered userId.
- If both the verifications succeed then update the parameter “isDeleted” of the requested user to deactivate it’s account.
- Send the user a mail informing them that their account has been deactivated, and after 1 month their account will automatically be deleted, and to reactivate they’ve to login back within 1 month.

- **Response messages:**
  - Response message should be “200 Ok” in case the operation is successful.
  - Response message should be “400 Bad Request” in case the required parameters are missing.
  - Response message should be “401 Unauthorized” in case the userName or password mismatched with the stored details of the requested user.
  - Response message should be “404 Not Found” in case the data for requested userId does not exist.
  - Response message should be “500 Internal Server Error” in case any other error occurred.
  - Responses from validateToken API will be returned.

- **Email Functionality:**
  - The email to be sent if the user deactivation operation is successful, and the template should contain the following text in the same order as provided:
    Subject: “Account Deactivation Confirmation”
    “Dear [User name],”

“We are writing to confirm that your account has been deactivated. We value your time with us and understand that account deactivation is a significant decision. We want to ensure you are aware of the following:”

“**Account Deactivation Date:** [Date of Deactivation]”

“**Account Deletion Deadline:** [Date of Deletion Deadline - 30 days from deactivation]”

“Your account is now inactive and will remain in this state until [Date of Deletion Deadline]. If you change your mind or wish to reactivate your account, you can do so by simply logging in to your account within this timeframe.”

“**What Happens Next?**”

- “If you choose not to reactivate your account by [Date of Deletion Deadline], it will be permanently deleted from our system, and all data associated with your account will be irretrievable.”
- “To reactivate your account, simply log in as you normally would. Your account will be restored with all your previous data, settings, and information.”

“**We're Here for You**”

“If you have any questions, concerns, or need assistance, please don't hesitate to contact us at shadow.works.1998@gmail.com. We're here to provide support during this process.”

“Thank you for being a part of our community, and we hope to have the opportunity to welcome you back in the future. If you decide to reactivate your account, we'll be here to assist you.”

“Best regards,”

“Team dailyFinance”

Req-009: requestPasswordReset API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will check for the userName or emailId, and if the record found in db with the provided userName or emailId then send the mail to the user to reset their password.

- ➔ Method type: **GET**.

- ➔ **API:**
  - name: **requestPasswordReset**
  - **api/users/requestPasswordReset**

- ➔ DB Bucket to be used: **userInfo**.

- ➔ Path parameter should be empty.
- ➔ Mandatory parameter to be sent as part of the request body: userName or emailId.
- ➔ Payload must be verified if the mandatory parameter is present or not.
- ➔ Validate if there's any user present in db with the entered emailId or userName or not. If the user is found then a random "verificationCode" needs to be generated and sent to the provided emailId via mail for resetting the password of the user.
- ➔ Generated "verificationCode" needs to be stored as a value of the parameter "verificationCode" in the user object.

- ➔ **Response messages:**
  - Response message should be "200 Ok" in case the operation is successful.
  - Response message should be "400 Bad Request" in case the required parameter is missing.
  - Response message should be "404 Not Found" in case the requested user does not exist.
  - Response message should be "500 Internal Server Error" in case any other error occurred.

- ➔ **Email Functionality:**
  - The email with the reset password link to be sent to the designated user in case the requested user is found, and the template should contain the following text in the same order as provided.
  - The "Password reset link" should be of below form:
    api/users/requestPasswordReset/{userId}/{verificationCode}
  - The Email template should contain the following text in the same order as provided:
    Subject: "Password Reset Request"
    "Dear [User name],"
    "We received a request to reset your password for your account at dailyFinance. To complete this process, please follow the instructions below."
    passwordResetLink
    "**Password Reset Link:** [Insert Password Reset Link Here]"
    "**Please Note:**"

- “This password reset link is valid for a limited time, so make sure to use it promptly.”
- “If you did not request this password reset, please contact us immediately at shadow.works.1998@gmail.com. It may indicate a potential security concern.”

“**Steps to Reset Your Password:**”

1. “Click on the provided "Password Reset Link" above.”
2. “You will be directed to a secure page where you can create a new password.**”**
3. “Choose a strong and unique password that you can remember, and enter it in the designated fields.”
4. “Confirm your new password by re-entering it.”
5. “Save your changes.”

“After completing these steps, your password will be successfully reset, and you can use it to log in to your account.”

“If you encounter any issues or need further assistance, please do not hesitate to reach out at shadow.works.1998@gmail.com. We are here to help.”

“Thank you for choosing dailyFinance. Your security and privacy are of utmost importance to us, and we're committed to ensuring a safe and smooth experience.”

“Best regards,”
“Team dailyFinance”

Req-010: resetPassword API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will update the password of users using their unique id stored in db.

- Method type: **PUT**.

- **API:**
  - name: **resetPassword**
  - **api/users/resetPassword/{userId}/{verificationCode}**

- DB Bucket to be used: **userInfo**.

- Parameter to be sent as part of the path: userId, and verificationCode.
- Mandatory parameter to be sent as part of the request body: password.
- Payload must be verified if the mandatory parameter is present or not.
- Validate if the user with the entered userId exists in DB or not. If the user exists then update the password with the new password.
- Before updating the password, encode it using bcrypt crypto with saltRounds.
- Send the user a mail informing them that their password has been successfully updated if the process is successful.

- **Response messages:**
  - Response message should be “201 Created” in case the operation is successful.
  - Response message should be “400 Bad Request” in case the required parameters are missing.
  - Response message should be “404 Not Found” in case the data for requested userId does not exist.
  - Response message should be “500 Internal Server Error” in case any other error occurred.

- **Email Functionality:**
  - The Email to be sent if the user password update operation is successful, and use the same email template used under update user password API successful response.

Req-011: getUserProfileImage API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will return the profile image details of the required user using the unique user id present in db.

➔ Method type: **GET**.

➔ **API:**
- name: **getUserProfileImage**
- **api/users/getUserProfileImage/{userId}**

➔ DB Bucket to be used: **userImage**.

➔ Parameter to be sent as part of the path: userId.
➔ Request body should be empty.
➔ Token must be sent as part of authorization bearer token.
➔ Call a validateToken API internally with the userId and token.
➔ If the validateToken API returns 200 response, then proceed to get the record from db for the required user.
➔ Image file to be returned as part of the response.

➔ **Response messages:**
- Response message should be “200 Ok” in case the operation is successful.
- Response message should be “400 Bad Request” in case the userId is missing.
- Response message should be “404 Not Found” in case the data for the required user does not exist.
- Response message should be “500 Internal Server Error” in case of any other error occurred.
- Responses from validateToken API will be returned.

Req-012: updateUserProfileImage API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will update the user profile image using the user's unique id stored in db.

- ➔ Method type: **PUT**.

- ➔ **API:**
  - name: **updateUserProfileImage**
  - **api/users/updateUserProfileImage/{userId}**

- ➔ DB Bucket to be used: **userImage**.

- ➔ Parameter to be sent as part of the path: userId.
- ➔ Token must be sent as part of authorization bearer token.
- ➔ An image file must be sent as part of the request body.
- ➔ Call a validateToken API internally with the userId and token.
- ➔ If the validateToken API returns 200 response, then proceed with updating the image detail of the user in db.

- ➔ **Response messages:**
  - Response message should be “201 Created” in case the operation is successful.
  - Response message should be “400 Bad Request” in case the required parameters are missing.
  - Response message should be “404 Not Found” in case the data for the requested userId does not exist in the userImage module.
  - Response message should be “500 Internal Server Error” in case any other error occurred.
  - Responses from validateToken API will be returned.

Req-013: deleteUserProfileImage API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will delete the user stored profile image details and store the default image details in db using a unique user id.

➔ Method type: **PUT**.

➔ **API:**
- name: **deleteUserProfileImage**
- **api/users/deleteUserProfileImage/{userId}**

➔ DB Bucket to be used: **userInfo**.

➔ Parameter to be sent as part of the path: userId.
➔ Token must be sent as part of authorization bearer token.
➔ Request body should be empty.
➔ Call a validateToken API internally with the userId and token.
➔ If the validateToken API returns 200 response, then proceed with updating the image details with the default image in db.

➔ **Response messages:**
- Response message should be “200 Ok” in case the operation is successful.
- Response message should be “400 Bad Request” in case the required parameters are missing.
- Response message should be “404 Not Found” in case the data for requested userId does not exist.
- Response message should be “500 Internal Server Error” in case any other error occurred.
- Responses from validateToken API will be returned.

Req-014: getUserFinanceInfo API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will return the finance details of the required user using their unique id present in db.

- Method type: **GET**.

- **API:**
  - name: **getUserFinanceDetails**
  - **api/users/getUserFinanceDetails/{userId}**

- DB Bucket to be used: **userInfo**.

- Parameter to be sent as part of the path: userId.
- Request body should be empty.
- Token must be sent as part of authorization bearer token.
- Call a validateToken API internally with the userId and token.
- If the validateToken API returns 200 response, then proceed to get the records from db for the required user.
- Fields to be returned as part of the response: userId, availableFunds, lifetimeIncome, lifetimeInvestment, and lifetimeExpenditure.

- **Response messages:**
  - Response message should be “200 Ok” in case the operation is successful.
  - Response message should be “400 Bad Request” in case the userId is missing.
  - Response message should be “404 Not Found” in case the data for the required user does not exist.
  - Response message should be “500 Internal Server Error” in case any other error occurred.
  - Responses from validateToken API will be returned.

Req-015: updateUserFinanceInfo API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will update the basic finance details of users using their unique id stored in db.

- Method type: **PUT**.

- **API:**
  - name: **updateUserFinanceInfo**
  - **api/users/updateUserFinanceInfo/{userId}**

- DB Bucket to be used: **userInfo**.

- Parameter to be sent as part of the path: userId.
- Token must be sent as part of the authorization bearer token.
- Parameters that can be sent as part of the request body: availableFunds, lifetimeIncome, lifetimeInvestment, and lifetimeExpenditure.
- Call a validateToken API internally with the userId and token.
- If the validateToken API returns 200 response, then proceed with updating the details of the user finance in db.

- **Response message:**
  - Response message should be “201 Created” in case the operation is successful.
  - Response message should be “400 Bad Request” in case the required parameters are missing.
  - Response message should be “404 Not Found” in case the data for requested userId does not exist.
  - Response message should be “500 Internal Server Error” in case any other error occurred.
  - Responses from validateToken API will be returned.

Req-016: getUserBasicSettings API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will return the expense, credit card, funds, and investment basic report settings of the user using their unique user id present in db.

- ➔ Method type: **GET**.

- ➔ **API:**
  - name: **getUserBasicSettings**
  - **api/users/getUserBasicSettings/{userId}**

- ➔ DB Bucket to be used: **userDashboardSettings**.

- ➔ Parameter to be sent as part of the path: userId.
- ➔ Request body should be empty.
- ➔ Token must be sent as part of authorization bearer token.
- ➔ Call a validateToken API internally with the userId and token.
- ➔ If the validateToken API returns 200 response, then proceed to get the records from db for the required user.
- ➔ Fields to be returned as part of the response: expense (bar & pie chart) settings, credit card (bar & pie chart) settings, income (bar & pie chart) settings, and investment (bar & pie chart) settings.

- ➔ **Response messages:**
  - Response message should be “200 Ok” in case the operation is successful.
  - Response message should be “400 Bad Request” in case the userId is missing.
  - Response message should be “404 Not Found” in case the data for the required user does not exist.
  - Response message should be “500 Internal Server Error” in case any other error occurred.
  - Responses from validateToken API will be returned.

Req-017: getDashboardReportSettings API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will return the dashboard report settings of the user using their unique user id stored in db.

- ➔ Method type: **GET**.

- ➔ **API:**
  - name: **getDashboardReportSettings**
  - **api/users/getDashboardReportSettings/{userId}**

- ➔ DB Bucket to be used: **userDashboardSettings**.

- ➔ Parameter to be sent as part of the path: userId.
- ➔ Request body should be empty.
- ➔ Token must be sent as part of authorization bearer token.
- ➔ Call a validateToken API internally with the userId and token.
- ➔ If the validateToken API returns 200 response, then proceed to get the records from db for the required user.
- ➔ Fields to be returned as part of the response: Expense bar chart, pie chart & line chart settings, Investment bar chart, pie chart & line chart settings, Income bar chart, pie chart & line chart settings, Credit card bar chart, pie chart & line chart settings, Spendings report, Income report, Investment report, Credit card report, and general report settings for each day, last month, last 3 months, last 6 months, last year, and custom date settings.

- ➔ **Response messages:**
  - Response message should be “200 Ok” in case the operation is successful.
  - Response message should be “400 Bad Request” in case the userId is missing.
  - Response message should be “404 Not Found” in case the data for the required user does not exist.
  - Response message should be “500 Internal Server Error” in case any other error occurred.
  - Responses from validateToken API will be returned.

Req-018: getUserDashboardSettings API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will return the entire report settings of the user using their unique user id present in db.

- ➔ Method type: **GET**.

- ➔ **API:**
  - name: **getUserDashboardSettings**
  - **api/users/getUserDashboardSettings/{userId}**

- ➔ DB Bucket to be used: **userDashboardSettings**.

- ➔ Parameter to be sent as part of the path: userId.
- ➔ Request body should be empty.
- ➔ Token must be sent as part of authorization bearer token.
- ➔ Call a validateToken API internally with the userId and token.
- ➔ If the validateToken API returns 200 response, then proceed to get the records from db for the required user.
- ➔ All fields to be returned as part of the response.

- ➔ **Response messages:**
  - Response message should be “200 Ok” in case the operation is successful.
  - Response message should be “400 Bad Request” in case the userId is missing.
  - Response message should be “404 Not Found” in case the data for the required user does not exist.
  - Response message should be “500 Internal Server Error” in case any other error occurred.
  - Responses from validateToken API will be returned.

Req-019: updateDashboardSettings API - Account Management Services

As part of the **accountManagementServices**, we need to create an API that will update the dashboard settings of the user using their unique id stored in db.

- Method type: **PUT**.

- **API:**
  - name: **updateDashboardSettings**
  - **api/users/updateDashboardSettings/{userId}**

- DB Bucket to be used: **userInfo**.

- Parameter to be sent as part of the path: userId.
- Token must be sent as part of the authorization bearer token.
- Parameters that can be sent as part of the request body: Expense bar chart, pie chart & line chart settings, Investment bar chart, pie chart & line chart settings, Income bar chart, pie chart & line chart settings, Credit card bar chart, pie chart & line chart settings, Spendings report, Income report, Investment report, Credit card report, and general report settings for general view pages and for each day, last month, last 3 month, last 6 month, last year, and custom date settings.
- Call a validateToken API internally with the userId and token.
- If the validateToken API returns 200 response, then proceed with updating the details of the user dashboard settings in db.

- **Response messages:**
  - Response message should be "201 Created" in case the operation is successful.
  - Response message should be "400 Bad Request" in case the required parameters are missing.
  - Response message should be "404 Not Found" in case the data for requested userId does not exist.
  - Response message should be "500 Internal Server Error" in case any other error occurred.
  - Responses from validateToken API will be returned.

Req-020: User Verification Check Scheduled Job - Account Management Services

As part of the **accountManagementServices**, we need to create a job scheduler that runs everyday at 12 in the afternoon and sends the mail to the registered users, who have not yet verified their account and their account creation time beyond 6 hours.

- ➔ DB Bucket to be used: **userInfo**, and **userLogs**.

- ➔ Run a query in the database to check for all the users whose email is not yet verified and their account is not yet activated.
- ➔ Then further check for all those users for whom their account creation time is beyond 6 hours.
- ➔ Then send out a mail to all these users that they've to verify their account within 10 days, or after 10 days their account will be self deactivated.
- ➔ Additionally, remove the users' existing verificationCode because doing so will help to ensure that the mail is sent just once.
- ➔ Generate the logs and store the records of such users in DB, also in case of an error generate the appropriate log and store in DB.

- ➔ **Email Functionality:**
  - The Email to be sent to the required users, and the template should contain the following text in the same order as provided:
    "Account Activation Required!"
    "We can see that you have not yet activated your account by verifying your email. Please use the new verification link to verify your account after logging in to our website."
    "Notification: Our system will terminate your account if you don't authenticate and activate it within the next 10 days."
    "Regards, Team financeTracker"

Req-021: Auto Deactivate User Scheduled Job - Account Management Services

As part of the **accountManagementServices**, we need to create a job scheduler that runs at every midnight and mark those registered users as soft deleted, who have not yet verified their account and their account creation time is beyond 10 days.

➔ DB Bucket to be used: **userInfo**, and **userLogs**.

➔ Run a query in the database to check for all the users whose email is not yet verified and their account is not yet activated.

➔ Then further check for all those users for whom their account creation time is beyond 10 days.

➔ Then send out a mail to all these users saying that, to reactivate the account, the user has to login to the website and get the new verification code and verify their account to activate within the next 30 days, or after 30 days, their account will be permanently deleted.

➔ Mark the user as soft deleted, and last login time as current time. This will help in further proceedings.

➔ Generate the logs and store the records of such users in DB, also in case of an error generate the appropriate log and store in DB.

➔ **Email Functionality:**

- The Email to be sent to the required users, and the template should contain the following text in the same order as provided:
  “Account Deactivated!”
  “We can see that you have not activated your account by verifying your email in the past 10 days, which caused your account to be self-deactivated. To reactivate your account, use the new verification link to verify your account after logging in to our website.”
  “Notification: Our system will delete your account if you don’t reactivate it in the next 30 days.”
  “Regards, Team financeTracker”

Req-022: Auto Delete User Scheduled Job - Account Management Services

As part of the **accountManagementServices**, we need to create a job scheduler that runs at midnight and delete those users and their related information who were marked as soft deleted and whose last login time is before 30 days.

- ➔ DB Bucket to be used: **userInfo**, **userImage**, **userBasicFinance**, **userDashboardSettings** and **userLogs**.

- ➔ Run a query in the database to check for all the users who are marked as soft deleted and their last login time is before 30 days.
- ➔ Then send out a mail to all these users saying that their account has been deleted permanently.
- ➔ Delete all such users from the database that were marked as soft deleted and whose last login time is beyond 30 days.
- ➔ Also delete all the related informations of all such users from other DBs.
- ➔ Generate the logs and store the records of such users in DB, also in case of an error generate the appropriate log and store in DB.

- ➔ **Email Functionality:**
  - The Email to be sent to the required users, and the template should contain the following text in the same order as provided:
    "Account Deleted!"
    "With a heavy heart, we would like to inform you that your account has been deleted. We'll miss you and hope to have you back soon, but unfortunately, you've got to start from scratch."
    "Regards, Team financeTracker"

Req-023: Auto Delete Logs Scheduled Job - Account Management Services

As part of the **accountManagementServices**, we need to create a job scheduler that runs at every midnight and delete the logs which are generated before 30 days.

- ➔ DB Bucket to be used: **userLogs**.

- ➔ Run a query in the database to check for all those logs which are generated before 30 days and delete all those logs.
- ➔ Before deleting the logs, generate an excel file with all the logs getting deleted and send it via mail to the official mail.
- ➔ In case of an error generate the appropriate log and store it in DB.

- ➔ **Email Functionality:**
  - The Email to be sent to our official mail, and the template should contain the following text in the same order as provided:
    “Deleted Logs Details”
    “Dear Admin, please find the details of the deleted logs in the attached file.”
    “Number of records:”
    “Log date interval:”
    “Regards, Team financeTracker”