accountManagementServices - [includes] - userServices, and privilegeServices

Req-001: createUser API - userServices

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

➔ Method type: **POST**.

➔ **API:**
- name: **createUser**
- **api/v1/users/createUser**

➔ DB Bucket to be used: **userInfo**.

➔ Path parameter should be empty.
➔ Parameters to be sent as part of the request body: firstName, lastName, phoneNo, emailId, bio, userName, password, and profileImg.
➔ Other parameters: role, blogsCount, createdAt, lastLogin, and isVerified should store default values.
➔ “password” needs to be encoded using bcrypt crypto with saltRounds as 10 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.
➔ The “verificationLink” which is sent to the user should contain three parameters: userId, time of creation, and verificationCode.
➔ The “verificationLink” should be of below form:
api/v1/users/verify/{userId}/{time}/{verificationCode}

➔ **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 “500 Internal Server Error” in case of any error occurred in storing the data in db.

➔ **Email Template:**
- The Email template should contain the following text in the same order as provided with the font size and color as specified.
  “We got a request for a new account creation with this emailId. Please verify your emailId to activate the account.” (font-size: 20px, type: text)
  “Please click on the following link to proceed.” (font-size: 18px, type: text)
  “Verify” (font-size: 16px, color: blue, type: link)
  “Stamp” (type: text) should be present at the end of the email.

Req-002: verifyAccount API - userServices

As part of the **userServices**, 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/v1/users/verify**

➔ DB Bucket to be used: **userInfo**.

➔ Parameters to be sent as part of the path: userId, time, and verificationCode.
➔ Request body should be empty.
➔ 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 "yes" then update the value of "isValidated" parameter as true for the requested user. And if "no" then send the appropriate message to 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 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.

➔ **HTML Templates:**
- 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 with the font size and color as specified.
  "Congrats! Your account has been verified successfully." (font-size: 28px, type: text, color: wheat)
  "Thanks for joining us, you're officially an Author and part of our Team." (font-size: 22px, type: text, color: wheat)
  "Stamp" (type: text) should be present at the end.

- 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 with the font size and color as specified.
  "Verification Code Expired!" (font-size: 28px, type: text, color: wheat)
  "Please relogin and get a new verification code to activate your account." (font-size: 22px, type: text, color: wheat)

“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.” (font-size: 22px, type: text, color: wheat)
“Stamp” (type: text) should be present at the end.

➔ **Email Template:**

- The Email template to be sent if the verification is successful, and the template should contain the following text in the same order as provided, with the font size and color as specified.
  “Welcome to our Family” (type: heading)
  “Congrats! Your account has been created successfully. You're now an Author and part of our Family.” (font-size: 20px, type: text)
  “Following are your registered details:” (font-size: 18px, type: text)
  “First Name:” (type: text)
  “Last Name:” (type: text)
  “Username:” (type: text)
  “Phone Number:” (type: text)
  “Email Id:” (type: text)
  “Role:” (type: text)
  “Bio:” (type: text)
  “If any of your details are wrong, please visit to our portal and update your details.” (font-size: 18px, type: text)
  “Note: You cannot update your Role.” (font-size: 18px, type: text)
  “Stamp” (type: text) should be present at the end.

Req-003: getAllUsersInfo API - userServices

As part of the **userServices**, we need to create an API that will return the list of all the users stored in db.

➔ Method type: **GET**.

➔ **API:**
- name: **getAllUsersInfo**
- **api/v1/users/getAllUsersInfo**

➔ DB Bucket to be used: **userInfo**.

➔ Path parameter should be empty.
➔ Request body should be empty.
➔ Fields to be returned as part of the response: firstName, lastName, phoneNo, emailId, bio, userName, role, profileImg, blogsCount, blogsList, createdAt, lastLogin, and isVerified.

➔ **Response messages:**
- Response message should be “200 Ok” in case the operation is successful.
- Response message should be “500 Internal Server Error” in case of any error occurred in connection with db.

Req-004: getUserDetails API - userServices

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

➔ Method type: **GET**.

➔ **API:**
- name: **getUserDetails**
- **api/v1/users/getUserDetails/{userId}**

➔ DB Bucket to be used: **userInfo**.

➔ Parameter to be sent as part of the path: “userId”.
➔ Request body should be empty.
➔ Fields to be returned as part of the response: firstName, lastName, phoneNo, emailId, bio, userName, role, profileImg, blogsCount, blogsList, createdAt, lastLogin, and isVerified.

➔ **Response messages:**
- Response message should be “200 Ok” in case the operation is successful.
- 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 connection with db.

Req-005: updateUserDetails API - userServices

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

➔ Method type: **PUT**.

➔ **API:**
- name: **updateUserDetails**
- **api/v1/users/updateUserDetails/{userId}**

➔ DB Bucket to be used: **userInfo**.

➔ Parameter to be sent as part of the path: “userId”.

➔ Parameters that can be sent as part of the request body: firstName, lastName, phoneNo, emailId, bio, userName, and profileImg.

➔ **Response messages:**
- Response message should be “201 Created” in case the operation is successful.
- Response message should be “400 Bad Request” in case the parameter body is null.
- 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 connection with db.

➔ **Email Template:**
- The Email template to be sent in case of operation is successful, and the template should contain the following text in the same order as provided, with the font size and color as specified.
  “Account updated successfully” (type: heading)
  “Congrats! Your account details have been updated successfully.” (font-size: 20px, type: text)
  “Following are your updated details:” (font-size: 18px, type: text)
  “First Name:” (type: text)
  “Last Name:” (type: text)
  “Username:” (type: text)
  “Phone Number:” (type: text)
  “Email Id:” (type: text)
  “Role:” (type: text)
  “Bio:” (type: text)
  “Total number of Blogs you've written:” (type: text)
  “Account created at:” (type: text)
  “Last login at:” (type: text)
  “Is your account verified?” (type: text)
  “If any of your details are wrong, please visit to our portal and update your details.” (font-size: 18px, type: text)
  “Note: You cannot update your Role.” (font-size: 18px, type: text)

“Stamp” (type: text) should be present at the end.

Req-006: updateUserPassword API - userServices

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

➔ Method type: **PUT**.

➔ **API:**
- name: **updateUserPassword**
- **api/v1/users/updateUserPassword/{userId}**

➔ DB Bucket to be used: **userInfo**.

➔ Parameter to be sent as part of the path: “userId”.
➔ Parameters to be sent as part of the request body: oldPassword, and newPassword.
➔ Validate if the user's oldPassword is the same as the stored password of the required user or not. If the same then check if the newPassword and oldPassword are not the same. If both the conditions satisfy then update the password.

➔ **Response messages:**
- Response message should be “201 Created” in case the operation is successful.
- Response message should be “400 Bad Request” in case the parameter body is null.
- Response message should be “403 Forbidden” in case any of the two required conditions is not satisfied.
- 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 connection with db.

➔ **Email Template:**
- The Email template to be sent in case of operation is successful, and the template should contain the following text in the same order as provided, with the font size and color as specified.
  “Password updated successfully” (type: heading)
  “Congrats! Your password has been updated successfully.” (font-size: 20px, type: text)
  “Stamp” (type: text) should be present at the end.

Req-007: updateUserRole API - userServices

As part of the **userServices**, we need to create an API that will update the role of the users using their unique id present in db.

➔ Method type: **PUT**.

➔ **API:**
- name: **updateUserRole**
- **api/v1/users/updateUserRole/{userId}**

➔ DB Bucket to be used: **userInfo**.

➔ Parameter to be sent as part of the path: “userId”.

➔ Parameter to be sent as part of the requested body: “role”.

➔ **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 parameter is 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 of any error occurred in connection with db.

➔ **Email Template:**
- The Email template to be sent in case of operation is successful, and the template should contain the following text in the same order as provided, with the font size and color as specified.
  “Role updated successfully” (type: heading)
  “We would like to inform you that your role has been updated to role.” (font-size: 20px, type: text)
  “Stamp” (type: text) should be present at the end.

Req-008: deactivateUser API - userServices

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

➔ Method type: **PUT**.

➔ **API:**
- name: **deactivateUser**
- **api/v1/users/deactivateUser/{userId}**

➔ DB Bucket to be used: **userInfo**.

➔ Parameter to be sent as part of the path: “userId”.
➔ Request body should be empty.
➔ Deactivate the user’s account by changing the value of “isVerified” parameter to false.

➔ **Response messages:**
- Response message should be “201 Created” in case the operation is successful.
- 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 connection with db.

➔ **Email Template:**
- The Email template to be sent in case of operation is successful, and the template should contain the following text in the same order as provided, with the font size and color as specified.
  “Account Deactivated!!!” (type: heading)
  “We would like to inform you that your account has been deactivated successfully.” (font-size: 20px, type: text)
  “To re-activate your account, kindly login to the portal, you'll get the mail with the verification code to reactivate your account.” (font-size: 18px, type: text)
  “Stamp” (type: text) should be present at the end.

Req-009: deleteUser API - userServices

➔ As part of the **userServices**, we need to create an API that will delete the user using its unique id present in db.
➔ Method type: **DELETE**.

➔ **API:**
- name: **deleteUser**
- **api/v1/users/deleteUser/{userId}**

➔ DB Bucket to be used: **userInfo**.

➔ Parameter to be sent as part of the path: “userId”.
➔ Parameter to be sent as part of the request body: “password”.
➔ Before deleting the account, validate if the provided password is correct or not. If validation is successful then delete the account.

➔ **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 “403 Forbidden” in case the required condition is not satisfied.
- 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 connection with db.

➔ **Email Template:**
- The Email template to be sent in case of operation is successful, and the template should contain the following text in the same order as provided, with the font size and color as specified.
  “Account Deleted!!!” (type: heading)
  “We feel sorry to inform you that your account has been deleted successfully as per your request.” (font-size: 20px, type: text)
  “We hope to see you back again someday.” (font-size: 18px, type: text)
  “Stamp” (type: text) should be present at the end.

Req-010: loginUser API - userServices

As part of the **userServices**, 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: **loginUser**
- **api/v1/users/loginUser**

➔ DB Bucket to be used: **userInfo**.

➔ Path parameter should be empty.
➔ Parameter to be sent as part of the request body: userName, and password.
➔ Parameters to be passed in token as a response: firstName, lastName, phoneNo, emailId, bio, role, blogsCount, blogsList, userName, isVerified, createdAt, lastLogin.
➔ Match the credentials with all the users stored in the db. If the credentials are correct then check if the user is verified or not. If the user is verified then generate the token and return the token as a response. 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.
➔ Generate the secret key, randomBytes of 64 bytes. Use this secret key to generate the token.

➔ **Response messages:**
- Response message should be “201 Created” in case the verification is required and email is sent.
- Response message should be “202 Accepted” in case the operation is successful and the token is returned as a response.
- Response message should be “400 Bad Requests” in case the required parameter is missing.
- Response message should be “403 Forbidden” in case the users entered credentials are incorrect.
- Response message should be “500 Internal Server Error” in case of any error occurred in connection with db.

Req-011: addBlogsToUser API - userServices

As part of the **userServices**, we need to create an API that will add a blog to the user's blogsList using the user's unique id present in db.

➔ Method type: **PUT**.

➔ **API:**
- name: **addBlogsToUser**
- **api/v1/users/addBlogsToUser/{userId}**

➔ DB Bucket to be used: **userInfo**.

➔ Parameter to be sent as part of the path: "userId".
➔ Parameter to be sent as part of the request body: "blogsId".
➔ "blogsCount" parameter should get incremented by one, and the "blogId" should get added to the "blogsList" array.

➔ **Response messages:**
- Response message should be "201 Created" in case the operation is successful.
- Response message should be "400 Bad Requests" in case the required parameter is 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 of any error occurred in connection with db.

Req-012: removeBlogsFromUser API - userServices

➔ As part of the **userServices**, we need to create an API that will remove a particular blog from the user's blogsList using the user's unique id present in db.
➔ Method type: **PUT**.

➔ **API:**
- name: **removeBlogsFromUser**
- **api/v1/users/removeBlogsFromUsers/{userId}**

➔ DB Bucket to be used: **userInfo**.

➔ Parameter to be sent as part of the path: "userId".
➔ Parameter to be sent as part of the request body: "blogsId".
➔ "blogsCount" parameter should get decremented by one, and the provided "blogId" should get removed from the "blogsList" array.

➔ **Response messages:**
- Response message should be "201 Created" in case the operation is successful.
- Response message should be "400 Bad Requests" in case the required parameter is 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 of any error occurred in connection with db.

Req-013: createPrivilege API - privilegeServices

➔ As part of the **privilegeServices**, we need to create an API that will create a new role and store it in db.
➔ Method type: **POST**.

➔ **API:**
- name: **createPrivilege**
- **api/v1/privileges/createPrivilege**

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

➔ Path parameter should be empty.
➔ Parameter to be sent as part of the request body: role, and privilegesAvailable.
➔ Other parameter: isValidated should store default value when created.

➔ **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 “500 Internal Server Error” in case of any error occurred in storing the data in db.

Req-014: getAllPrivileges API - privilegeServices

As part of the **privilegeServices**, we need to create an API that will return the list of all the roles stored in db.

➔ Method type: **GET**.

➔ **API:**
- name: **getAllPrivileges**
- **api/v1/privileges/getAllPrivileges**

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

➔ Path parameter should be empty.
➔ Request body should be empty.
➔ Parameters to be sent as part of the response: role, and privilegesAvailable.

➔ **Response messages:**
- Response message should be “200 Ok” in case the operation is successful.
- Response message should be “500 Internal Server Error” in case of any error occurred in connection with db.

Req-015: getActivePrivileges API - privilegeServices

As part of the **privilegeServices**, we need to create an API that will return the list of all the active roles present in the db.

➔ Method type: **GET**.

➔ **API:**
- name: **getActivePrivileges**
- **api/v1/privileges/getActivePrivileges**

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

➔ Path parameter should be empty.
➔ Request body should be empty.
➔ Parameters should be returned as part of the response: role, and privilegesAvailable.
➔ Only privileges which are having “isValidated” parameter value as “true” need to be returned.

➔ **Response messages:**
- Response message should be “200 Ok” in case the operation is successful.
- Response message should be “500 Internal Server Error” in case of any error occurred in connection with db.