***API Documentation: Location-Based Todo Reminder Backend***

***Authentication***

***Endpoint: User Registration***

***URL: POST https://localist.onrender.com/user/register***

***Request Details***

***Request Body (JSON)***

***json***

***Copy***

***Download***

***{***

***"username": "string (required, trimmed)",***

***"email": "string (required, valid email format, unique)",***

***"password": "string (required, min length: 5)",***

***"profileUrl": "string (optional, URL to profile picture)"***

***}***

***Example Request***

***json***

***Copy***

***Download***

***{***

***"username": "Jane Smith",***

***"email": "jane.smith@example.com",***

***"password": "secure123",***

***"profileUrl": "https://example.com/profiles/jane.jpg"***

***}***

***Response***

***Successful Response (200 OK)***

***json***

***Copy***

***Download***

***{***

***"success": true,***

***"message": "User registered successfully, please verify your account first"***

***}***

***Validation Rules***

***Based on the provided schema:***

1. ***username:***
   * ***Required field***
   * ***Automatically trimmed***
   * ***Error message: "Please Enter user name"***
2. ***email:***
   * ***Required field***
   * ***Must be unique***
   * ***Valid email format (RFC 5322 standard)***
   * ***Automatically trimmed***
   * ***Error message: "Email is required" or "please enter valid email"***
3. ***password:***
   * ***Required field***
   * ***Minimum length: 5 characters***
   * ***Automatically hashed before saving***
   * ***Error message: "password is mandatory" or "your password should be greater than length of 5 characters"***
4. ***profileUrl (optional):***
   * ***Not required***
   * ***Should be a valid URL if provided***
   * ***No specific validation in schema beyond being a string***

***Error Responses***

* ***400 Bad Request:***
  + ***Missing required fields***
  + ***Invalid email format***
  + ***Password too short***
  + ***Other validation errors***
* ***409 Conflict:***
  + ***Email already exists***
* ***500 Internal Server Error:***
  + ***Server-side errors***

***Example Usage (JavaScript)***

***javascript***

***Copy***

***Download***

***const registerUser = async (userData) => {***

***try {***

***const response = await fetch('https://localist.onrender.com/user/register', {***

***method: 'POST',***

***headers: {***

***'Content-Type': 'application/json',***

***},***

***body: JSON.stringify({***

***username: userData.username,***

***email: userData.email,***

***password: userData.password,***

***profileUrl: userData.profileUrl || null***

***})***

***});***

***const data = await response.json();***

***if (!response.ok) {***

***throw new Error(data.message || 'Registration failed');***

***}***

***return data;***

***} catch (error) {***

***console.error('Registration error:', error);***

***throw error;***

***}***

***};***

***// Usage***

***registerUser({***

***username: "Alex Johnson",***

***email: "alex.j@example.com",***

***password: "alex12345",***

***profileUrl: "https://example.com/profiles/alex.jpg"***

***})***

***.then(response => console.log(response.message))***

***.catch(error => console.error(error.message));***

***Important Notes***

1. ***The profileUrl field is optional - you can omit it completely or pass null***
2. ***After registration:***
   * ***User receives a verification code (handled by backend)***
   * ***Account remains unverified (isVerified: false) until email verification***
   * ***Default role is set to "student"***
3. ***The password is automatically hashed before storage (never stored in plaintext)***
4. ***The response doesn't include the user object for security reasons***

***Frontend Implementation Tips***

1. ***Validate the email format client-side before sending***
2. ***Ensure password meets length requirement before submission***
3. ***For profile pictures:***
   * ***You can either:  
     a) Let users provide a URL, or  
     b) First upload the image to a storage service and then provide the URL***
4. ***Handle the verification flow after successful registration***

User Sign-In

Authenticates a user and returns user details with an access token.

**Endpoint:** POST /user/sign-in

**Base URL:** https://location-based-todo-reminder-backend.vercel.app

Request

{

"email": "yashpawar12122004@gmail.com",

"password": "yash@12122004"

}

Headers

* Content-Type: application/json

Response

**Success Response (200 OK)**

{

"success": true,

"user": {

"\_id": "66a9f980f77d23f50d9a9119",

"username": "yash choudhary",

"email": "yashpawar12122004@gmail.com",

"password": "$2b$10$6bMslGaFLNa/IwE35SP.Y.pfCZTtNCg6VthP7gpeBIQd0VeS675PG",

"verifyCode": "805479",

"isVerified": true,

"verifyCodeExpiry": "2024-07-31T09:44:48.960Z",

"Role": "admin",

"createdAt": "2024-07-31T08:44:48.969Z",

"updatedAt": "2024-09-30T14:29:21.298Z",

"\_\_v": 15,

"profileUrl": "https://th.bing.com/th/id/OIP.ed0RqUUqkqGszO7DikXhagAAAA?rs=1&pid=ImgDetMain"

},

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY2YTlmOTgwZjc3ZDIzZjUwZDlhOTExOSIsImVtYWlsIjoieWFzaHBhd2FyMTIxMjIwMDRAZ21haWwuY29tIiwicm9sZSI6ImFkbWluIiwiaWF0IjoxNzQ1MDc5NTEwLCJleHAiOjE3NDU5NDM1MTB9.HwQ5vKye1cLrnUYvkgo-wMu1N4ISi7WX1euh9m07oew"

}

**Error Responses:**

* 400 Bad Request: Invalid request body or missing fields
* 401 Unauthorized: Invalid credentials
* 500 Internal Server Error: Server error

Fields Description

**Request Fields:**

* email (string, required): User's email address
* password (string, required): User's password

**Response Fields:**

* success (boolean): Indicates if the request was successful
* user (object): User details object containing:
  + \_id (string): Unique user identifier
  + username (string): User's display name
  + email (string): User's email address
  + password (string): Hashed password (should not be used client-side)
  + isVerified (boolean): Whether email is verified
  + Role (string): User role (e.g., "admin")
  + profileUrl (string): URL to user's profile picture
  + Other fields are for internal use and can typically be ignored
* token (string): JWT access token for authenticated requests

Usage Example (JavaScript)

const signInUser = async () => {

try {

const response = await fetch('https://location-based-todo-reminder-backend.vercel.app/user/sign-in', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

email: 'yashpawar12122004@gmail.com',

password: 'yash@12122004'

})

});

const data = await response.json();

if (data.success) {

// Store token for future authenticated requests

localStorage.setItem('authToken', data.token);

// Use user data

console.log('Logged in user:', data.user);

} else {

console.error('Login failed');

}

} catch (error) {

console.error('Error:', error);

}

};

Notes

* The returned JWT token should be included in subsequent authenticated requests in the Authorization header as Bearer <token>
* The password field in the response contains the hashed password and should never be used or stored client-side
* The token has an expiration time (check the exp claim in the JWT)

Endpoint: Verify User Account

**URL:** POST https://localist.onrender.com/user/verify-code

Request Details

Request Body (JSON)

json

Copy

Download

{

"email": "string (required, must match registered email)",

"code": "string (required, 6-digit verification code)"

}

Example Request

json

Copy

Download

{

"email": "yashpawar12122004@gmail.com",

"code": "539367"

}

Response

Successful Verification (200 OK)

json

Copy

Download

{

"success": true,

"message": "your account has been successfully verified"

}

Error Responses

* **400 Bad Request**:
  + Missing email or code
  + Invalid code format (not 6 digits)
  + Code expired
* **404 Not Found**:
  + Email not found in system
  + No pending verification for this email
* **401 Unauthorized**:
  + Incorrect verification code
  + Account already verified
* **500 Internal Server Error**:
  + Server-side errors

Validation Rules

1. **email**:
   * Must match a registered email exactly (case-sensitive)
   * Must have a pending verification request
2. **code**:
   * Must be exactly 6 digits
   * Must match the verification code sent to the email
   * Must not be expired (codes expire after verifyCodeExpiry)

Example Usage (JavaScript)

javascript

Copy

Download

const verifyAccount = async (email, verificationCode) => {

try {

const response = await fetch('https://localist.onrender.com/user/verify-code', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

email: email,

code: verificationCode

})

});

const data = await response.json();

if (!response.ok) {

throw new Error(data.message || 'Verification failed');

}

return data;

} catch (error) {

console.error('Verification error:', error);

throw error;

}

};

// Usage example

verifyAccount("yashpawar12122004@gmail.com", "539367")

.then(response => {

console.log(response.message);

// Proceed with login or redirect to login page

})

.catch(error => {

console.error(error.message);

// Show error to user

});

Important Notes

1. **Verification Flow**:
   * User receives code via email after registration
   * Code is valid until verifyCodeExpiry (typically 1 hour)
   * After successful verification, isVerified flag is set to true
2. **Security**:
   * Codes are one-time use
   * Failed attempts may trigger rate limiting (not shown in schema)
   * Verified accounts can proceed to login
3. **Error Handling**:
   * For expired codes, users should request a new verification code
   * For already verified accounts, redirect to login page

Frontend Implementation Tips

1. Implement a countdown timer showing code expiry time
2. Provide a "Resend Code" option if code is expired
3. Validate code format (6 digits) client-side before submission
4. Clear the verification form after successful verification
5. Auto-redirect to login page after successful verification

Related Endpoints

* /user/register - Initial registration
* /user/resend-verification - Request new verification code
* /user/sign-in - Login after verification

Endpoint: User Authentication

**URL:** POST https://localist.onrender.com/user/sign-in

Request Details

Request Body (JSON)

json

Copy

Download

{

"email": "string (required, registered email)",

"password": "string (required, min length: 5)"

}

Example Request

json

Copy

Download

{

"email": "yash26it074@satiengg.in",

"password": "yash@12122004"

}

Response

Successful Authentication (200 OK)

json

Copy

Download

{

"success": true,

"user": {

"\_id": "6803e8df24360e3dad57d197",

"username": "yash choudhary",

"email": "yash26it074@satiengg.in",

"password": "$2b$10$2yI.6XSdGbbtoo232oX7B.y/Bf1iC0NNIW/7b4drRL/ke2ba3I2Z6",

"profileUrl": "https://res.cloudinary.com/duzmyzmpa/image/upload/v1745086686/r2ue2mmnwljudyzg308x.jpg",

"verifyCode": "134639",

"isVerified": true,

"verifyCodeExpiry": "2025-04-19T19:18:06.045Z",

"Role": "student",

"createdAt": "2025-04-19T18:18:07.026Z",

"updatedAt": "2025-04-19T18:18:37.939Z",

"\_\_v": 0

},

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY4MDNlOGRmMjQzNjBlM2RhZDU3ZDE5NyIsImVtYWlsIjoieWFzaDI2aXQwNzRAc2F0aWVuZ2cuaW4iLCJyb2xlIjoic3R1ZGVudCIsImlhdCI6MTc0NTA4Njc0NCwiZXhwIjoxNzQ1OTUwNzQ0fQ.9ejLeoyq-VZIM-\_gfjJ2jdxrs9zVKDIGBcZOFS2mpBM"

}

Response Fields Description

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| success | boolean | Indicates successful authentication |
| user.\_id | string | Unique user identifier |
| user.username | string | User's display name |
| user.email | string | Verified email address |
| user.password | string | Hashed password (for reference only) |
| user.profileUrl | string | URL to user's profile picture |
| user.isVerified | boolean | Account verification status |
| user.Role | string | User role ("student", "instructor", or "admin") |
| user.createdAt | string | Account creation timestamp |
| user.updatedAt | string | Last update timestamp |
| token | string | JWT for authenticated requests |

Error Responses

* **400 Bad Request**:
  + Missing email or password
  + Invalid email format
* **401 Unauthorized**:
  + Incorrect password
  + Account not verified (isVerified: false)
* **404 Not Found**:
  + Email not registered
* **500 Internal Server Error**:
  + Server-side errors

Example Usage (JavaScript)

javascript

Copy

Download

const signInUser = async (credentials) => {

try {

const response = await fetch('https://localist.onrender.com/user/sign-in', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

email: credentials.email,

password: credentials.password

})

});

const data = await response.json();

if (!response.ok) {

throw new Error(data.message || 'Authentication failed');

}

// Store token in localStorage/sessionStorage

localStorage.setItem('authToken', data.token);

localStorage.setItem('user', JSON.stringify(data.user));

return data;

} catch (error) {

console.error('Login error:', error);

throw error;

}

};

// Usage example

signInUser({

email: "yash26it074@satiengg.in",

password: "yash@12122004"

})

.then(response => {

console.log("Welcome", response.user.username);

// Redirect to dashboard

})

.catch(error => {

console.error(error.message);

// Show error to user

});

Important Notes

1. **Security**:
   * Always use HTTPS for requests
   * Never store plaintext passwords
   * The returned password field is hashed and should not be used client-side
2. **Token Usage**:
   * Include the JWT in subsequent requests as:

Copy

Download

Authorization: Bearer <token>

* + Token contains user ID, email, and role
  + Token expires after duration set in JWT\_EXPIRE

1. **Account Verification**:
   * Only verified accounts (isVerified: true) can log in
   * Unverified accounts should be redirected to verification flow
2. **Password Requirements**:
   * Minimum 5 characters
   * Case-sensitive
   * Validated against bcrypt hash

Frontend Implementation Tips

1. Implement form validation before submission
2. Show loading state during authentication
3. Securely store the JWT (preferably in httpOnly cookies)
4. Handle token expiration gracefully
5. Redirect unverified users to verification page
6. Implement password visibility toggle for better UX

Endpoint: Initiate Password Reset

**URL:** POST https://localist.onrender.com/user/forgot-password

Request Details

Request Body (JSON)

json

Copy

Download

{

"email": "string (required, registered email)"

}

Example Request

json

Copy

Download

{

"email": "yashpawar12122004@gmail.com"

}

Response

Successful Response (200 OK)

json

Copy

Download

{

"success": true,

"message": "sent forgot password email successfully"

}

Response Fields Description

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| success | boolean | Indicates successful request processing |
| message | string | Confirmation message |

Error Responses

* **400 Bad Request**:
  + Missing email field
  + Invalid email format
* **404 Not Found**:
  + Email not registered in system
* **500 Internal Server Error**:
  + Email sending failure
  + Server-side errors

Example Usage (JavaScript)

javascript

Copy

Download

const requestPasswordReset = async (email) => {

try {

const response = await fetch('https://localist.onrender.com/user/forgot-password', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

email: email

})

});

const data = await response.json();

if (!response.ok) {

throw new Error(data.message || 'Password reset request failed');

}

return data;

} catch (error) {

console.error('Error:', error);

throw error;

}

};

// Usage example

requestPasswordReset("yashpawar12122004@gmail.com")

.then(response => {

console.log(response.message);

// Show success message to user

})

.catch(error => {

console.error(error.message);

// Show error message to user

});

Important Notes

1. **Security**:
   * This endpoint will never confirm whether an email is registered to prevent email enumeration
   * The success message is the same whether the email exists or not
2. **Email Contents**:
   * The email will contain:
     + A password reset link with token
     + Instructions for resetting password
     + Expiration time (typically 1 hour)
3. **Rate Limiting**:
   * Multiple requests may be rate-limited to prevent abuse
   * Users should wait before requesting another reset email
4. **Token Generation**:
   * Generates ResetPasswordToken (crypto random string)
   * Sets ResetPasswordTokenExpire (typically 1 hour from generation)

Frontend Implementation Tips

1. Implement email validation before submission
2. Show loading state during request
3. Display success message even if email doesn't exist (for security)
4. Provide a countdown before allowing another reset request
5. Guide users to check spam folder if email doesn't arrive
6. Disable form submit button after successful request to prevent duplicates

Related Endpoints

* POST /user/reset-password - Actually reset the password using the token
* POST /user/verify-code - For account verification
* POST /user/sign-in - For logging in after password reset

Endpoint: Reset User Password

**URL:** PUT https://localist.onrender.com/user/reset-password/:resetToken

Request Details

URL Parameters

| **Parameter** | **Description** |
| --- | --- |
| resetToken | Required. The password reset token received via email (e.g., c608ed3d6b819c459820e7175b09060b07586946) |

Request Body (JSON)

json

Copy

Download

{

"password": "string (required, min length: 5)"

}

Example Request

json

Copy

Download

{

"password": "yash@12122004"

}

Response

Successful Response (200 OK)

json

Copy

Download

{

"success": true,

"message": "your password has been reset successfully"

}

Response Fields Description

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| success | boolean | Indicates successful password reset |
| message | string | Confirmation message |

Error Responses

* **400 Bad Request**:
  + Missing password field
  + Password too short (<5 characters)
  + Invalid reset token format
* **401 Unauthorized**:
  + Invalid or expired reset token
  + Token already used
* **404 Not Found**:
  + No user associated with reset token
* **500 Internal Server Error**:
  + Server-side errors

Example Usage (JavaScript)

javascript

Copy

Download

const resetPassword = async (resetToken, newPassword) => {

try {

const response = await fetch(`https://localist.onrender.com/user/reset-password/${resetToken}`, {

method: 'PUT',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

password: newPassword

})

});

const data = await response.json();

if (!response.ok) {

throw new Error(data.message || 'Password reset failed');

}

return data;

} catch (error) {

console.error('Error:', error);

throw error;

}

};

// Usage example

const resetToken = 'c608ed3d6b819c459820e7175b09060b07586946';

const newPassword = 'yash@12122004';

resetPassword(resetToken, newPassword)

.then(response => {

console.log(response.message);

// Redirect to login page or show success message

})

.catch(error => {

console.error(error.message);

// Show error message and allow retry

});

Important Notes

1. **Security**:
   * Reset tokens are single-use and expire after 1 hour (configurable)
   * Passwords are automatically hashed before storage
   * Previous tokens are invalidated after successful reset
2. **Password Requirements**:
   * Minimum 5 characters
   * No maximum length (but very long passwords may be truncated)
   * Strongly recommend enforcing additional complexity rules client-side
3. **Token Handling**:
   * Tokens are generated by the /user/forgot-password endpoint
   * Sent to user via email in a reset link
   * Typically looks like: c608ed3d6b819c459820e7175b09060b07586946
4. **User Experience**:
   * After successful reset:
     + All active sessions should be terminated
     + User should be prompted to login with new password
     + Success message should be displayed

Frontend Implementation Tips

1. Add password strength meter (client-side only)
2. Implement password confirmation field (client-side validation)
3. Show loading state during submission
4. Handle invalid tokens gracefully (redirect to forgot password page)
5. Auto-redirect to login page after successful reset
6. Consider adding CAPTCHA for public reset pages

Endpoint: User Logout

**URL:** POST https://localist.onrender.com/user/logout

Request Details

Headers

| **Header** | **Value** | **Required** |
| --- | --- | --- |
| Authorization | Bearer <your-auth-token> | Yes |

Request Body

None required

Response

Successful Response (200 OK)

{

"success": true,

"message": "Logged out successfully"

}

Response Fields Description

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| success | boolean | Indicates successful logout |
| message | string | Confirmation message |

Error Responses

* **401 Unauthorized**:
  + Missing or invalid authorization token
  + Already logged out
* **500 Internal Server Error**:
  + Server-side errors

Example Usage (JavaScript)

const logoutUser = async () => {

try {

const token = localStorage.getItem('authToken');

const response = await fetch('https://localist.onrender.com/user/logout', {

method: 'POST',

headers: {

'Authorization': `Bearer ${token}`,

'Content-Type': 'application/json'

}

});

const data = await response.json();

if (!response.ok) {

throw new Error(data.message || 'Logout failed');

}

// Clear client-side storage

localStorage.removeItem('authToken');

localStorage.removeItem('user');

return data;

} catch (error) {

console.error('Logout error:', error);

throw error;

}

};

// Usage example

logoutUser()

.then(response => {

console.log(response.message);

// Redirect to login page

})

.catch(error => {

console.error(error.message);

// Handle error (token may already be invalid)

});

Important Notes

1. **Authentication**:
   * Requires valid JWT in Authorization header
   * Token will be invalidated server-side after logout
2. **Session Management**:
   * Terminates the current user session
   * Invalidates the provided JWT token
   * All subsequent requests with this token will fail
3. **Client-Side Cleanup**:
   * Should remove any stored tokens/user data
   * Should redirect to login page
4. **Security**:
   * Implements proper token invalidation
   * Protects against token reuse

Frontend Implementation Tips

1. Call logout when:
   * User clicks logout button
   * Session expires
   * Getting 401 errors from API
2. UI Considerations:
   * Show loading state during logout
   * Display success message
   * Redirect to login page
3. Storage Cleanup:
   * Clear localStorage/sessionStorage
   * Reset application state
4. Error Handling:
   * Handle cases where token is already invalid
   * Provide feedback if logout fails