

How to log in — by email or by Telegram

27 September 2026

There are no passwords in TOnline. A person proves who they are by receiving a **login link and code** on a contact they own — an email address or a Telegram chat — and opening the link or typing the code. This page explains both routes, how to set a person up for each, and what to check when it doesn't work.

Written for whoever sets people up and answers "I can't log in". Code references are for when you need to look deeper.

The idea in one paragraph

Every person who can log in is a **User**, and each User has one or more **Contacts** (Account → *Edit/Add/Delete Contacts*, `/contacts/<user_id>`). For logging in, a contact is an email address or a Telegram chat. (The contact form also offers **cell**: a phone number kept so an admin can contact the person directly. The system sends nothing to it, so it can't be used to log in.) Asking for a login sends a link and an 8-digit code to a contact; the link contains a signed token naming that contact. Opening the link, or typing the code on the login page, logs the browser in as the contact's User and marks the contact **confirmed**. After that the browser stays logged in (see [How long a login lasts](#)).

Logging in by email

What the person does

1. Go to `https://online.tagtough.co.za/login`.
2. Type their email address and press **Get token**.
3. Open the email *Login to Tagtough Online* and click **Click here to login** — or type the code from the email into **Or type your login code** on the login page and press **Log in with code**.

That's it — they land on the dashboard (a guard lands on GuardSense).

Link or code? The link logs in whichever browser it opens in. The code logs in the browser it is typed into — use it when the link would open somewhere else: on an **iPhone Home Screen app** (see below), or when the email is read on a computer but the login is wanted on a phone.

The page always answers the same way: *If that address belongs to an account, a login link and code are on their way...* — whether the address exists, belongs to a disabled user, or was held back by anti-spam. This is deliberate, so the page can't be used to find out who has an account. To see what really happened, look in the log (`Config.LOG_FILE`, and forwarded to Telegram): each attempt writes `(<address>) trying to login` followed by one of `email sent`, `email NOT sent: <reason>`, `account disabled` or `email does not exist`.

The login page accepts **email addresses only** — a Telegram-only person must use the Telegram route below.

Setting someone up for email login

- *New user:* **Users** page (`/users`) → add user with an email address. The login link is sent straight away.
- *Existing user:* open their contacts (`/contacts/<user_id>` , or Account → *Edit/Add/Delete Contacts*), add a contact of type **email**. A login link is sent to it straight away.
- An unconfirmed email contact has a **Send Verification** button on the Account page to send the link again.

Code: `users/routes.py login() , login_token() , login_code() ; users/utils.py send_login_token() , issue_login_code() , use_login_code() .`

Logging in by Telegram

A Telegram contact has to be **linked** once — the system needs to learn the person's Telegram chat id. After that they ask the bot for a login link whenever they need one.

Linking a Telegram contact (once)

1. Add a contact of type **telegram** to the user (Users page when creating them, or `/contacts/<user_id>`). The system gives it a short temporary code like `/sab3x` .
2. The contacts page shows **View/Print Telegram QR image link** and **or click here** next to it.
3. The person scans the QR code (or opens the link) on their phone. Telegram opens the Tagtough bot and sends it `/start sab3x` automatically. Typing `/sab3x` to the bot does the same thing.
4. The bot answers *Thank you for confirming your telegram contact* and immediately sends a login link.

The contact is now confirmed and holds their Telegram chat id instead of the code.

The production bot is **@tagtough_bot**. Each dev machine has its own bot (`Config.TELEGRAMQRURL`), so a code made on dev only works with the dev bot. Every link the pages show (the QR, *or click here*, the Account page's **Open the bot** button, the message after adding a user) is built from that setting, so it always opens the right bot with the code already filled in.

Getting a login link later

Open the Tagtough bot and send `/start` . The keyboard at the bottom shows a **Login** button — tap it and a login link and code arrive in the chat. Typing the word `login` (any capitals) does the same. The code goes into the login page's **Or type your login code** box, which is how a Telegram-only person logs in on a device other than the one Telegram opens links in.

The Login button appears for every login: each has at least the GuardSense page to open ([GuardSense — the guard's phone page](#)).

Guards: "Get Web Login"

A guard who registered through Telegram (a team's Telegram registration QR) has a guard identity but no login. The bot's ... menu offers **Get Web Login**. Tapping it creates a login User for them (role *Guard*), links it to their guard identity and sends a login link — this is how a guard reaches the GuardSense page in a browser. If they already have a login it links that instead.

Code: `telegram/routes.py` — `linkuser()`, the `/start` and `Login` handlers, and the `getweblogin` button.

On an iPhone: use the code in the Home Screen app

An app added to an iPhone's Home Screen (*Share* → *Add to Home Screen*, e.g. GuardSense) keeps its **own** login, separate from Safari. A login link always opens in Safari (or in Telegram's own browser), so it logs Safari in — never the Home Screen app. In the Home Screen app, ask for a login as usual, then **type the code** from the email or Telegram message into **Or type your login code**. Android doesn't have this problem: an installed app shares Chrome's login.

Afterwards use only the Home Screen app on that phone. Safari and the app count as two places, and the one-place-at-a-time rule (below) logs the other out.

How long a login lasts

- **The link:** valid for **one hour**, and it can be opened more than once in that hour — on purpose, because some mail systems' spam scanners open every link in an email before the person does, and a link that only worked once would already be used up. An expired link shows *That is an invalid or expired token* — just ask for a new one.
- **The code:** 8 digits, valid for the same hour, and it works **once**. Asking again sends a new code and the old one stops working. It is stored only as a hash (`UserSettings` , `forwhat = 40`). After 10 wrong codes from one address in an hour, codes from there are refused for the rest of that hour (the log says `login code: too many wrong codes from ...`); the link still works.
- **The logged-in browser:** stays logged in for up to a year (Flask-Login's "remember me", which is always on), until the person logs out.
- **One place at a time:** unless the person's role has `multiple_logins` (Super User, Distributor, Advance Distributor), logging in somewhere new logs out the previous browser. The old browser shows *Your account can only be logged in from one place at a time* on its next page.

"I can't log in" — checklist

1. **Is the User disabled?** Users page. A disabled user can't request or use a link.
2. **Is the email exactly right?** It must match the contact letter-for-letter (case doesn't matter).
3. **What does the log say?** Find `(<address> trying to login in Config.LOG_FILE` . The next words tell you: `email does not exist` (spelling, or add the address), `account disabled` , `email NOT sent:` ... (the reason — a mail-server failure or anti-spam), or `email sent` (check their spam folder).

4. **Too many requests?** A contact that is sent more than ~10 emails with less than 5 minutes between each is held back (anti-spam; 30 for Telegram) and the log says `email NOT sent: ... too many sent`. Wait 5 minutes and try once.
5. **Telegram: is the contact confirmed?** On the contacts page an unconfirmed Telegram contact still shows its QR link. Have them scan it again.
6. **Telegram: no Login button?** The chat isn't linked to a login yet (`/start` says so). Typing `login` also sends a link once it is.
7. **Keeps getting logged out?** Someone else is using the same account from another device, and the role doesn't allow multiple logins — or, on an iPhone, the person uses both Safari and a Home Screen app.
8. **iPhone Home Screen app still asks to log in after the link?** The link logged in Safari. Type the code in the app instead (above).
9. **"That code is wrong or has expired"** — only the newest code works, once, within the hour. Ask again.
10. **On a dev machine?** Messages to anyone who isn't a Super User are suppressed on purpose (`Contact.send()`); the log says `email NOT sent: [dev] suppressed send to ...`.

Seeing what they see: a Super User, Distributor or Advance Distributor can open the person's Account page and press **Login As User** (next to *Editing user, ...*). See [Logging in as someone else](#). To get the person themselves in, add or fix their contact and let the system send the link.

Logging in as someone else

For support: to see exactly what a customer sees. On the person's **Account** page (`/account/<user_id>`) press **Login As User**. The button only appears when all of these hold:

- your role has the `login_as_other_user` permission (Super User, Distributor, Advance Distributor);
- the person is in your own tree (anyone, for roles with `view_all_groups`);
- their role gives them nothing you don't have — a Distributor can't become a Super User;
- they are not disabled.

What happens: your browser becomes theirs until you **log out** (then log back in as yourself). They are *not* logged out — you share their session rather than replacing it. It is not remembered: closing the browser ends it. Every use writes an Audit log entry (*Logged in as User*) and a log line, which reaches Telegram.

There used to be a raw **Login token** link on the contacts page that let anyone who could open it log in as that person, with no permission check; it has been removed.