Recap of the v3 auth flow in plain English

==
=======================================

☑ Server side (shared hosting / cPanel)

Everything in xcmxfa_auth_php_v3.zip is server-side only.

Lives on your shared server

Typical location:

public_html/auth/

Written in PHP

Talks directly to:

your MySQL database

PHP mail()

Exposes HTTPS endpoints like:

```
POST /auth/login
POST /auth/register/start
GET  /auth/verify-email
POST /auth/register/complete
POST /auth/refresh
POST /auth/logout
```

The mobile / React Native app never sees these PHP files.
It only talks to them over HTTPS.

==
===================================

☑ Client side (React Native app)

Your client-side /auth/ folder contains only:

```
/auth
  ├─ authClient.js
  └─ deviceId.js
```

Their roles

authClient.js

wraps fetch()

calls /auth/login, /auth/register/start, /auth/refresh, etc

stores tokens (e.g. AsyncStorage / memory)

is the only place the app knows about auth endpoints

deviceId.js

generates & persists a stable per-device ID

used only when rememberDevice=true

never touches passwords or credentials

No PHP, no DB logic, no email logic on the client.

===

☑ Architectural separation (important)

You now have a clean boundary:

Layer	Responsibility
React Native app	UI, form validation, navigation
authClient.js	HTTP calls + token handling
PHP /auth	Identity, verification, security
Legacy PHP	Flights, schedules, business data

This means:

You can redesign UI freely

You can refactor auth server-side later

You can swap hosting or introduce Node/Go later

Legacy app can continue to coexist safely

==

REGISTRATION (NEW USER)

1. User fills the RegisterScreen (company + job + staff number, and HV email local-part if HV).

2. App calls POST /auth/register/start with:

```
{
  "company": "KLM" | "HV",
  "job": "cockpit" | "cabin" | "ground",
  "staffNumber": "12345" | "123456",
  "hvEmailLocalPart": "firstname.lastname" // HV only
```

}

3. Server derives:

   username = KLM12345 or HV12345

   email:

   KLM → klm12345@klm.com or k123456@klm.com (6-digit rule)

   HV → <localpart>@transavia.com

4. Server creates or updates a row in users (if needed) with:

   active = 0

   password = NULL

   email = derived/stored

   company = KLM|HV

5. Server emails the verification link.

EMAIL VERIFICATION

6. User clicks the link: GET /auth/verify-email?token=...

7. Server sets:

   active = 1

   email_verified_at = NOW()

   …and redirects to your app (e.g. https://app.xcmxfa.com/verified?...).

SET PASSWORD + FIRST LOGIN

8. After verification, the user sets a password via:

   POST /auth/register/complete with { username, password, rememberDevice, device? }

9. Server checks:

   user exists

   active = 1

   email_verified_at is set
   …then stores password_hash() and returns tokens.

NORMAL LOGIN (EXISTING USERS TOO)

10. App calls POST /auth/login with { username, password, rememberDevice, device? }

11. Server requires:

active = 1

password matches

REMEMBER ME

12. If rememberDevice=true, server returns a refresh token and your app can use:

POST /auth/refresh to mint new access tokens silently.