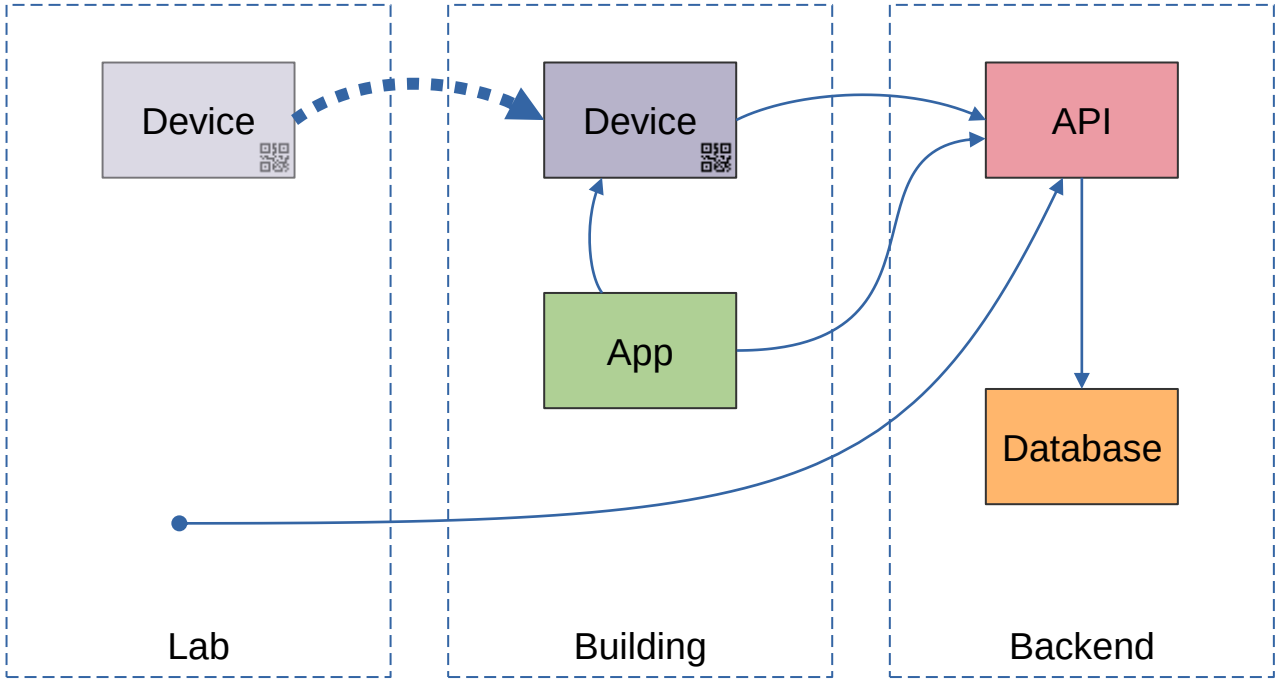
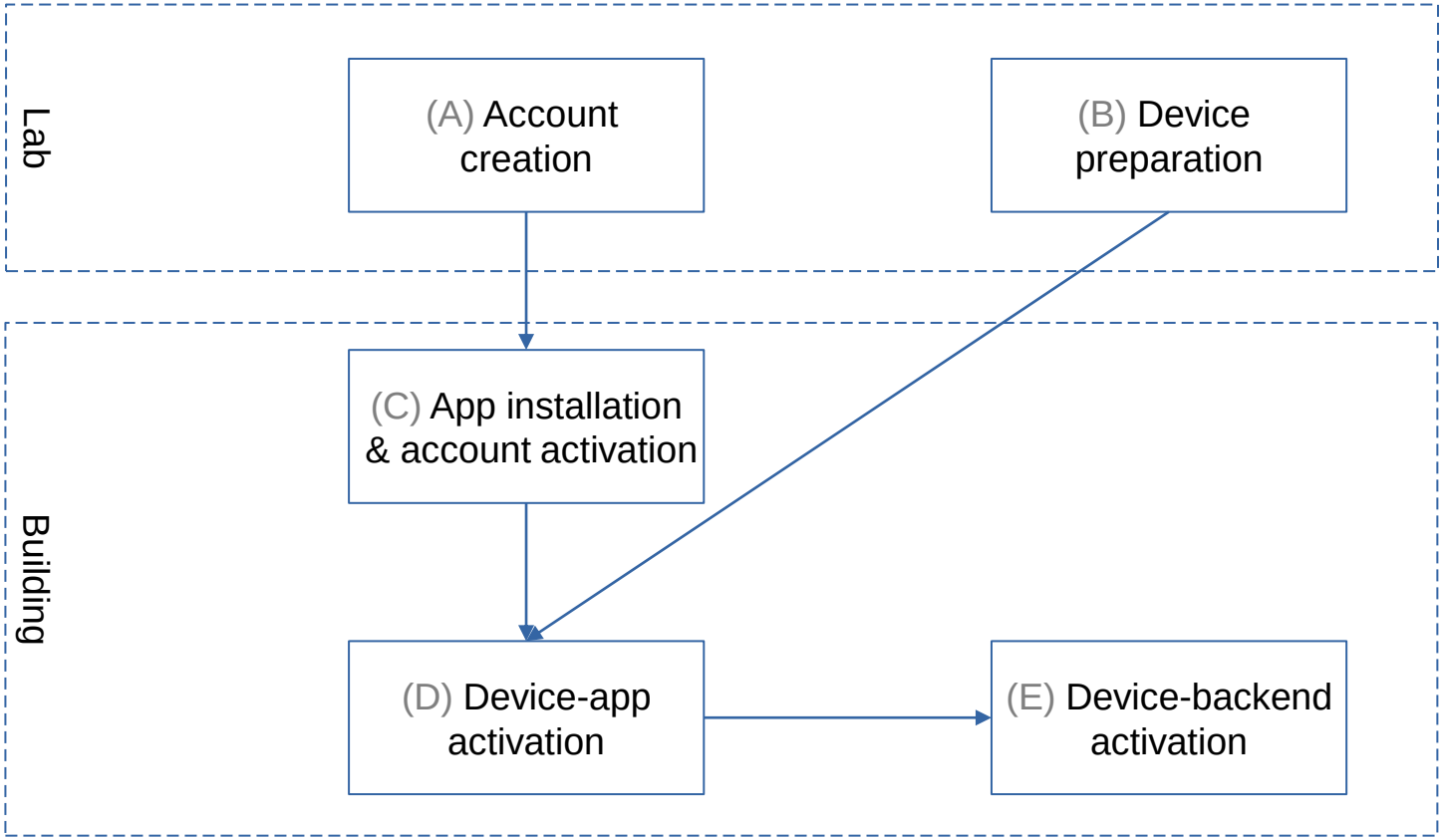
Twomes Provisioning



Date: 12-8-2021

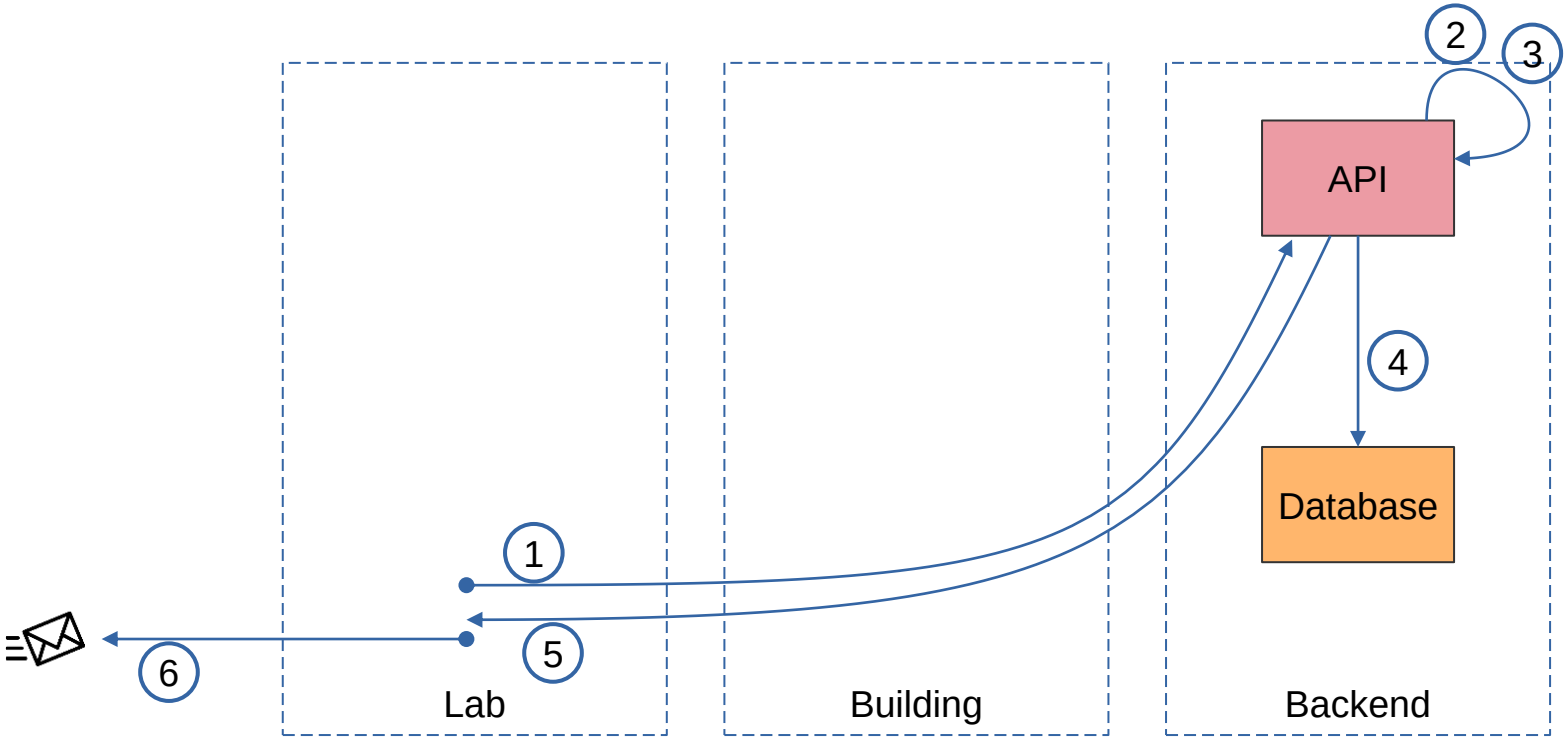
Dependencies between phases



Entities and roles

- **User**
  - a natural person, who participates in the research project as a subject, living in a building (in Assendorp, Zwolle);
  - represented by an `account` in the API and database;
  - represented by a unique, random, secret, short-lived `account_activation_token` that can be used once to link an app to an `account`;
  - tokenized as a unique, random, secret, short-lived `account.session_token` identifying the `account` in calls from the app;
  - represented by a `pseudonym` in communication between frontend helpdesk and backend researcher.
- **Device** (measurement device, a.k.a. Twomes 'planet' device, as opposed to Twomes 'satellite' device) :
  - represented by a `device` in the API and database;
  - represented by a unique, random, secret, long-lived `device.activation_token` that can be used once by the app and device to link the `device` to an `account`;
  - tokenized as a unique, random, secret, long-lived device `device.session_token` identifying the `device` in calls from the `device`;
- **App** (Twomes mobile app 'WarmteWachter')
  - A mobile app used by the user to facilitate installing/activating/monitoring and stopping devices in the user's home;
- **Backend**: **API** + **Database**
  - the server environment responsible for storing and managing accounts, devices and measurements;
- **Lab**
  - the place where devices preparation takes place;
- **Building**
  - the home of a user where device installation, activation and measurements take place;
- **Backend researcher**
  - A researcher (working for Windesheim) who is and remains unaware of personally identifiable information that would enable him/her to trace back measurement data to a natural person (or a specific building);
  - refers to user only by `pseudonym` (a unique number);
- **Frontend helpdesk**
  - A contact person (working for 50 Tinten Groen Assendorp), who can be contacted by users and who is aware of (but does not disclose to third parties) the relation between a `pseudonym`, and personally identifiable information about the user (e.g., name, e-mail address and/or street address).

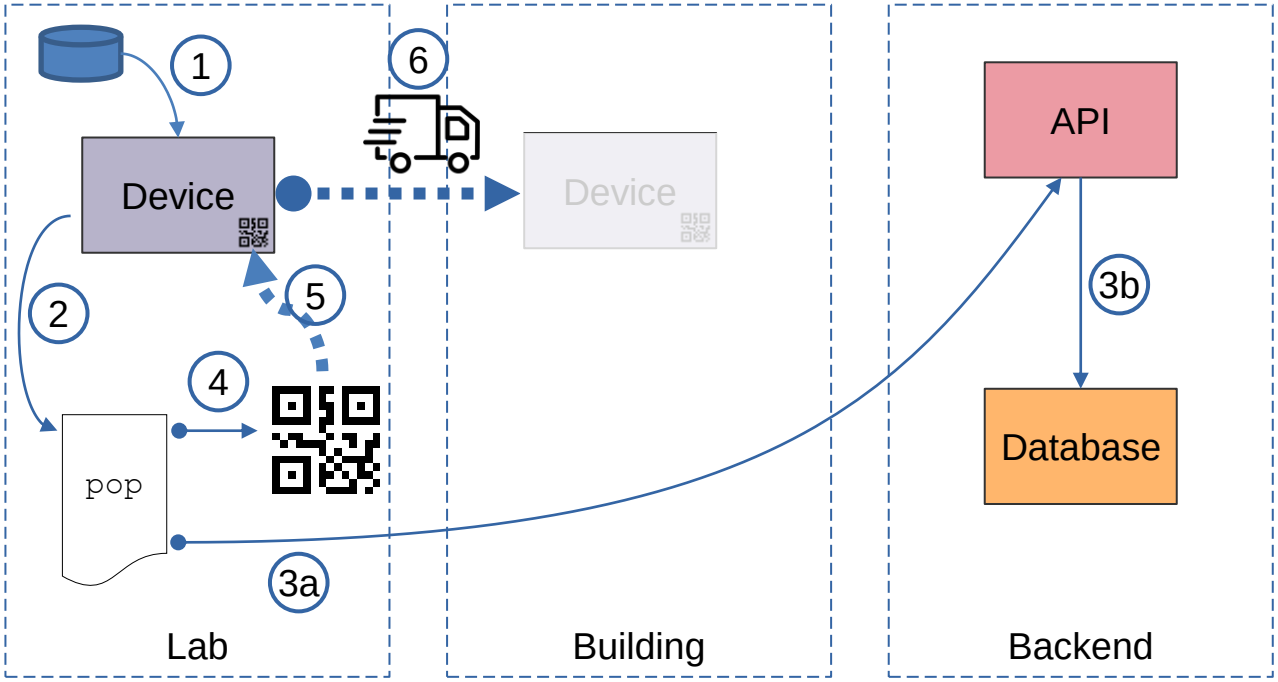
(A) Account creation



(A) Account creation

1. Backend researcher calls POST on `/account` API endpoint with input
   - `pseudonym`: a number between 800.000 and 899.999 (optional, will be automatically generated if not provided)
   - `location` for `building`: used to get weather info (optional, default~Assendorp, Zwolle); course-grained: avoid street address precision!
   - `timezone` for `building`: a valid TZ database name, used to interpret timestamps of measurements, which always use UTC timezone (optional, default="`Europe/Amsterdam`")
2. API creates an `account.activation_token`
   - A unique, random `account.activation_token`, to identify the account in a Firebase Dynamic Link
3. API creates Firebase Dynamic Link, including a unique value for `account.activation_token`
   - Format: `https://energietransitiewindesheim.page.link/`
     `?link=https%3A%2F%2Faccount%2F<account_activation_token>`
     `&apn=nl.windesheim.energietransitie.warmtewachter`
     `&ibi=nl.windesheim.energietransitie.warmtewachter`
     `&isi=1563201993`
     `&efr=1`
     while replacing `<account_activation_token>` with the URL-encoded value of `account.activation_token`
4. API creates entity in `account` and `building` table in database and stores
   - `account.activation_token`
   - `building.location, building.timezone`
5. API returns result of the POST on `/account` to backend researcher, with output
   - `Pseudonym`: a number between 800.00 and 899.999
   - `firebase_url`: the Firebase Dynamic Link for this specific account.
6. Backend researcher provides `pseudonym` and `firebase_url` to frontend helpdesk
   - Frontend helpdesk is responsible for sending the `firebase_url` to the proper user by e-mail
   - Frontend helpdesk is responsible for associating the `pseudonym` with the proper user (and his/her personally identifiable information)

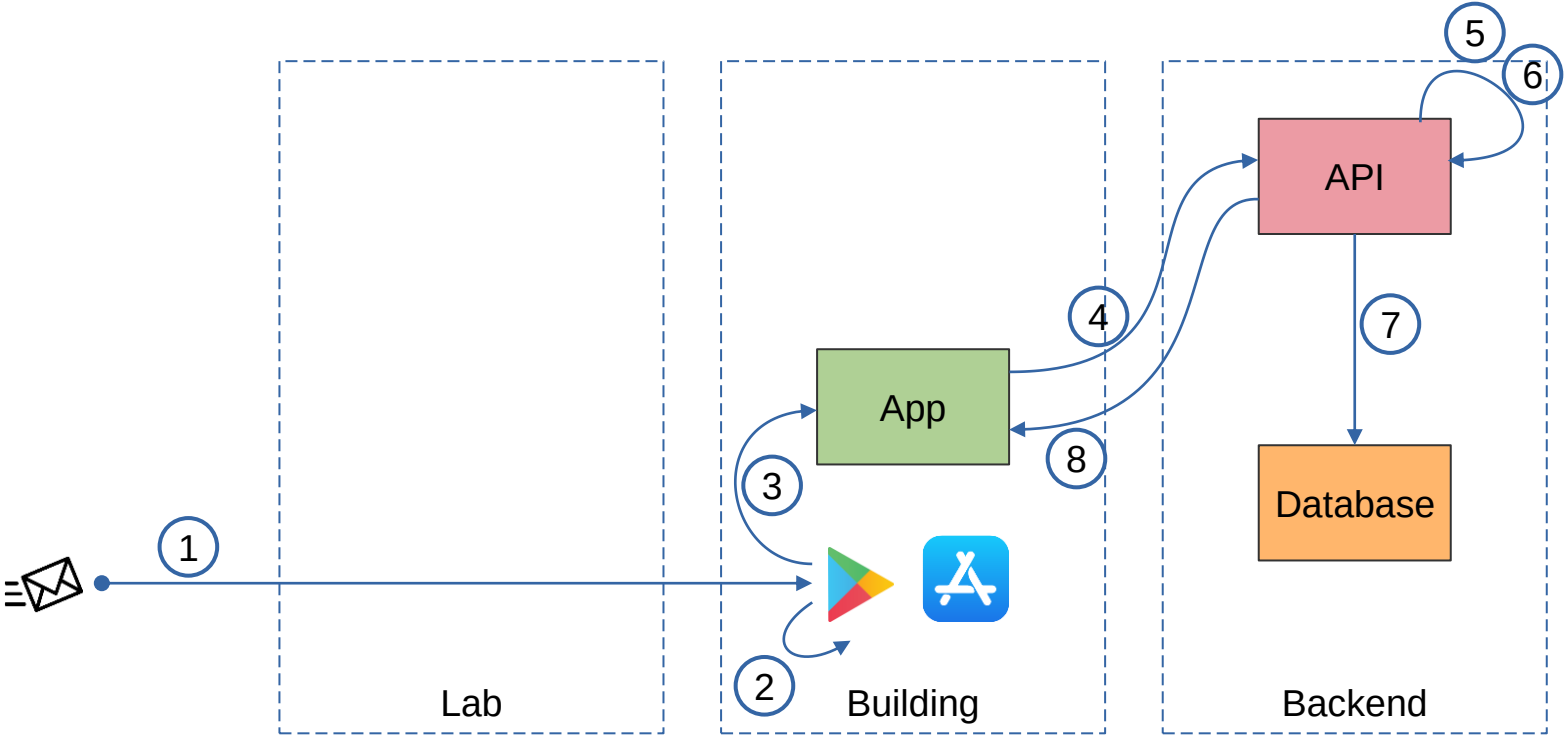
(B) Device preparation



(B) Device preparation

1. Backend researcher uploads proper firmware to device
2. Backend researcher establishes `device.name`, and `device.activation_token` on device
   - This is a randomly generated unique identifier for a device, stored in persistent memory on device and used as BLE service name during (`ble`) of SSID (`softap`) Unified Provisioning, on the backend server and used 'pop' value in QR-code payload. The `device.activation_token` serves a similar purpose for the device, as the `account.activation_token` for the account.
3. Backend researcher calls POST on `/device` API endpoint
   a. with input `name, device_type, activation_token`
   b. API stores new entry in `device` table
      - Device is in 'unattached' state, not connected to building
4. Backend researcher creates device-specific QR-code with offline QR-code generator; examples:
   `{"ver":"v1","name":"TWOMES-0D45DF","pop":"810667973","transport":"ble"}`
   `{"ver":"v1","name":"TWOMES-8E23A6","pop":"516319575","transport":"softap","security":"1","password":"516319575"}`
5. Backend researcher prints QR-code and attaches it to the (back of the enclosure of) the device
6. Backend researcher packages a set of devices per user
   - Frontend helpdesk defines sets of devices to be assembled per `pseudonym`, provides this to backend researcher
   - Backend researcher collates sets of devices per `pseudonym` and labels sets with `pseudonym`
   - Frontend helpdesk relabels `pseudonym` with the street address of the building of the user, initiates shipment

(C) App installation & account activation



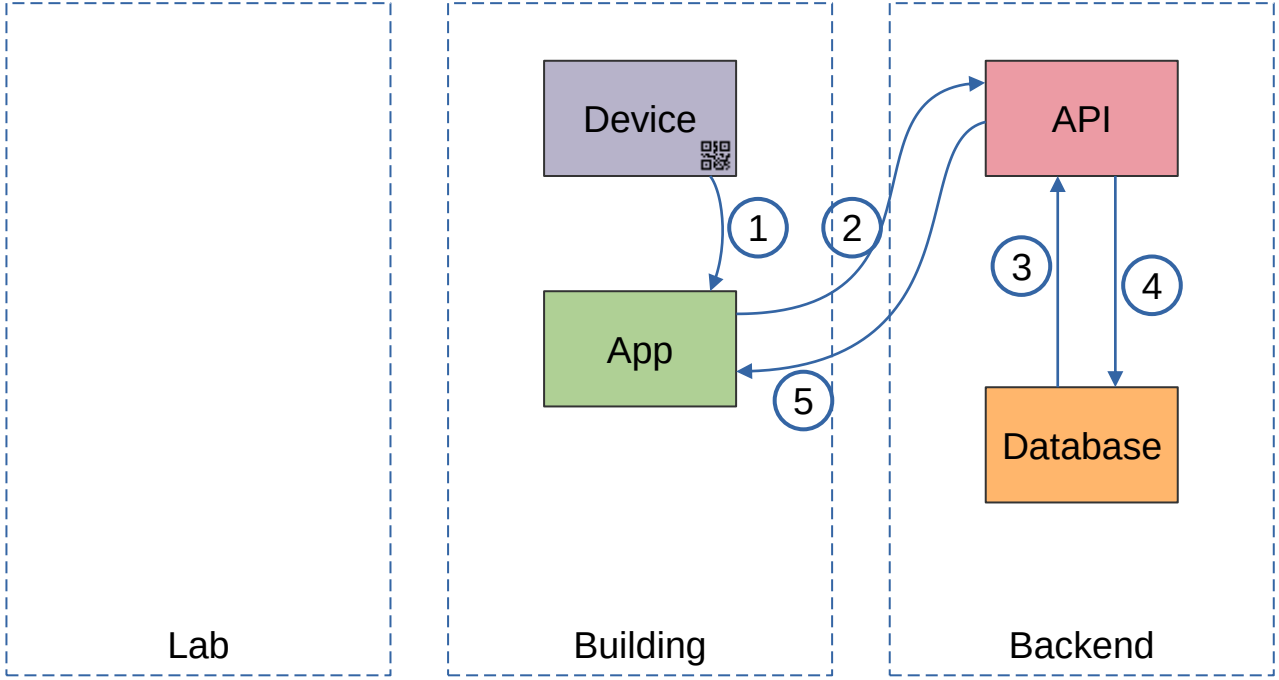
(C) App installation & account activation

1. User clicks on Firebase Dynamic Link in e-mail on smartphone
2. User clicks on ‘install’ and ‘continue’; app installs and opens on mobile device
3. App reads `account.activation_token` from Firebase Dynamic Link
   - App automatically retrieves the `account_activation_token` from the link

All part of regular Firebase workflow

4. App calls POST on `/account/activate`, sending `account.activation_token` as parameter
5. API verifies `account.activation_token`: `account` may not have been activated yet, token may not be expired
6. API generates a random, long-lived, secret `account.session_token`: to identify the `account` in calls from the app
7. API updates the associated entry in the `account` table in the database
   - `account.activated_on` timestamp; do not yet expire / invalidate the `account.activation_token`
   - `account.session_token`
   - Note: the `account.activation_token` expires the first time the `account.session_token` is used by the app (i.e, shortly after the first QR-code of a device is scanned)
8. API returns `account.session_token` in result of POST on `/account/activate`, to the app
   - Must be stored persistently by the app, to be used in all future API calls from the app (`AccountSessionTokenBearer`)

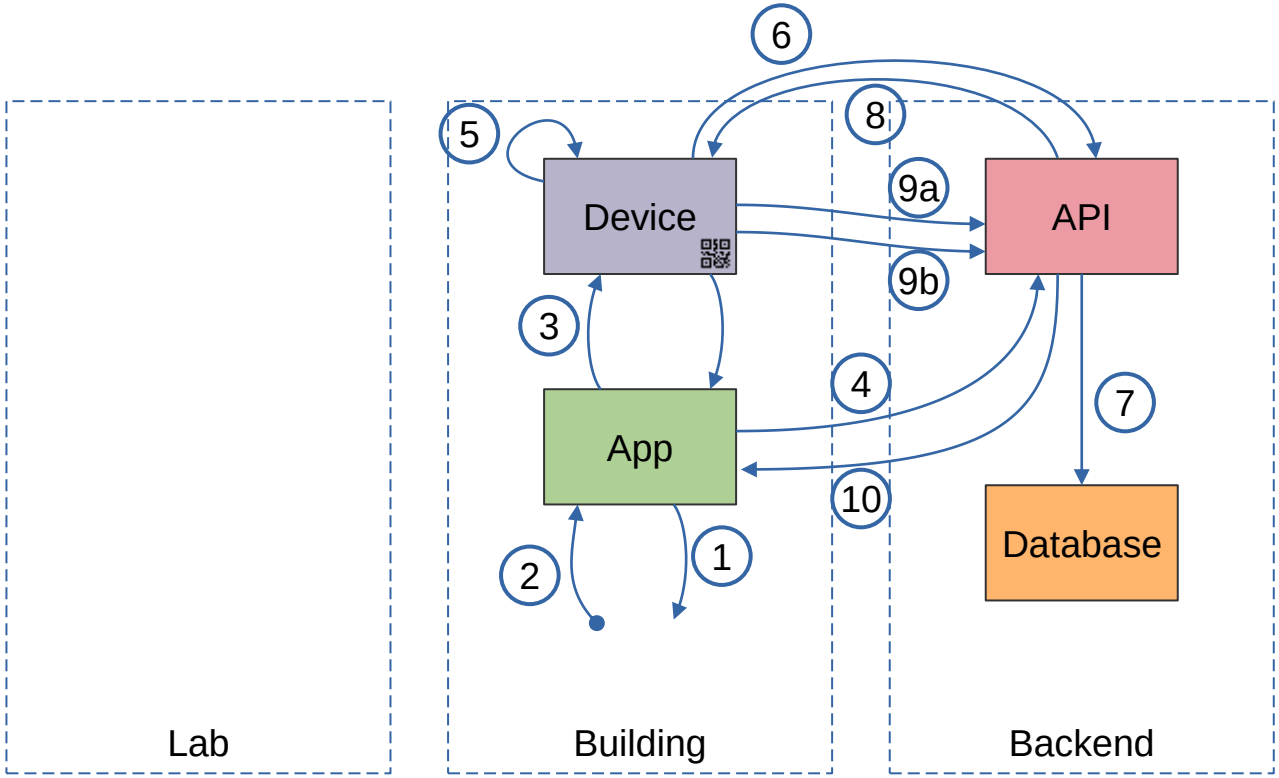
(D) Device-app activation



(D) Device-app activation

1. App scans the QR-code attached as a sticker to the back of the device enclosure, and reads values for the following keys in the QR-code payload:
   - `name`: the service name (BLE service or SSID) used during provisioning; in Twomes we use the value of `device.name`
   - `pop`: in Twomes we use the value of `device.activation_token`
   - `transport`: the type of transport (`softap`: Wi-Fi or `ble`: Bluetooth Low energy) used during Unified Provisioning
   - `security`: whether the transport is secured (`0` of `1`) during Unified Provisioning;
   - `password` : SSID password (only used for QR-codes containing `"transport":"softap","security":"1"`)
     in Twomes we use the value of `device.activation_token`
2. App calls `GET on /device_type/{device_name}` using the `name` value from the QR-code as `{device_name}`
3. API returns the results of `GET on /device_type/{device_name}`, to the app, including:
   - `device_type.display_name`, which is in the app used as header for the installation instruction manuals
   - `device_type.installation_manual_url`, where installation instructions for this device type can be found;
     URL format to expect: https://energietransitiewindesheim.nl/manuals/<DeviceType.name>/,
     where `<DeviceType.name>` is replaced with the specific `DeviceType.name, e.g.`
     https://energietransitiewindesheim.nl/manuals/Generic-Test/
4. User installs and powers up device according to the installation manual, confirms this in app.
5. App calls POST on `account/device/activate` API endpoint, with input
   - `device.activation_token`, as present in QR-code payload
6. API looks up `device.activation_token` in `device` table; re-activation is only allowed by the same account
7. API links the `device` to (the `building` of) the `account`
   - The link between `device` and `account` can be undone, manually (e.g. for device re-deploy later to building of another `account`)

(E) Device-backend activation



(E) Device-backend activation

1. App retrieves and shows installation instructions to user
   - Using installation instructions URL, using in-app webview, or similar
2. User confirms installation is done according to instructions, including powering up the device
3. App passes home Wi-Fi credentials to the device using the Espressif Unified Provisioning protocol
   - Using the preferred transport (`ble` or `softap`) indicated in the QR-code payload
   - Using `password` from the in QR-code payload, in case `softap` (temporary Wi-Fi access point) was indicated in QR-code payload
4. App starts calling GET on `/device/{device.name}`: check whether device is online (i.e. server receives its heartbeats)
   - Repeat every x seconds; at first, API returns results that indicate device is not online (see bullet 10.)
   - App indicates the device status to the user (partially provisioned; e.g., light green checkmark or green outline of checkmark)
5. Device detects it is online, and not yet activated
6. Device calls POST on `device/activate` API endpoint, with parameter `device.activation_token`
   - Note: unlike the `account.activation_token`, the `device.device.activation_token` does NOT expire
7. API generates a random, long-lived, secret `device.session_token`, and stores it in the device table
8. API returns the `device.session_token` as a result of the POST on `device/activate`
   - Must be stored persistently by the device, to be used for all future API calls from the device (`DeviceSessionTokenBearer`)
9. Device starts uploading measurements
   a) via POST on `/device/measurements/variable-interval`, or POST on `/device/measurements/fixed-interval`
   b) every Twomes device must send `heartbeat` measurements via POST on `/device/measurements/variable-interval`
      a) First upload with (a single) `heartbeat` measurement as soon as possible after step 8 (receiving `device/activate` result)
      b) Heartbeats are measured once per 600 s (~ heartbeats are measured once every 10 minutes)
      c) Heartbeats are uploaded once every 3600 s (~ heartbeats are uploaded once every hour)
10. App continues calling GET on `/device/{device.name}`; until first device measurement is seen, or until app times out
    - App indicates to user that device is activated successfully (e.g., dark green, solid checkmark), or not (e.g., a red cross)