2025

Software Architectuur

BIOMETRISCH TOEGANGSSYSTEEM

PIOTR TADRALA | INFORDB

O-PP-CMK

INHOUDSOPGAVE

1.0 INLEIDING

1.3 DOEL VAN DIT DOCUMENT

Dit document omschrijft de software architectuur van het biometrisch toegangssysteem ontwikkeld voor InforDB. De focus van het systeem ligt op een schaalbaar systeem dat gebruik maakt van anti-spoofing technieken en multi-factor authentication. Doel van dit document is om inzicht te geven over de opbouw, werking, dependencies en de gemaakte keuzes van het systeem.

1.2 CONTEXT VAN HET SYSTEEM

Het systeem bestaat uit vijf onafhankelijke applicaties die samen het volledige systeem vormen. Elke applicatie vervult een specifieke taak en communiceert met de rest van het systeem om via gebruikersinteractie het slot te openen.

1. **Verification Node**: verzamelt beelden en PIN-invoer, stuurt raw data naar de hub.
2. **Hub**: Verzameling van microservices, kern van het systeem.
3. **Lock**: Microcontroller dat het slot mechanisme.
4. **Interface**: Interface van het systeem, maakt users en nodes management mogelijk.
5. **Backend**: Cloud applicatie voor API integraties en beheer op afstand.

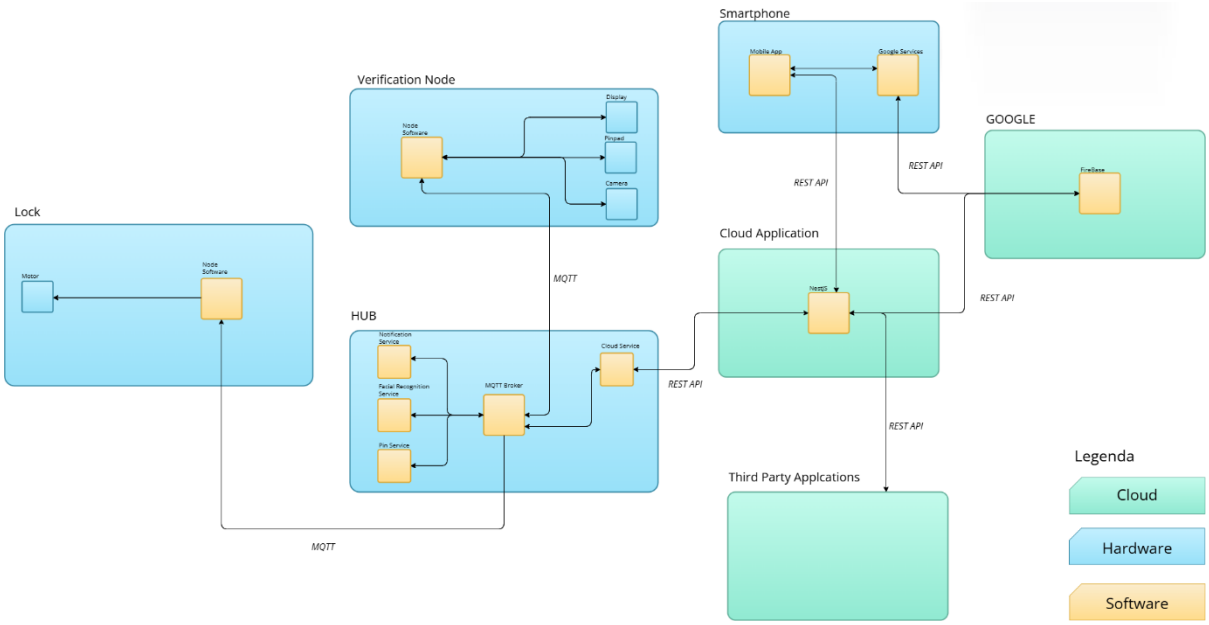
1.3 SCOPE

Dit document beschrijft de softwarearchitectuur van het biometrisch toegangssysteem en behandelt de vijf onafhankelijke applicaties, namelijk: Backend, Interface, Hub, Verification Node en Lock. Voor elke component wordt de rol binnen het systeem toegelicht, de gebruikte technologieën en de gemaakte keuzes. Daarnaast wordt de volledige datastroom, van verificatie tot slotactivering, helder geschetst.

2.0 OVERZICHT ARCHITECTUUR

2.1 SYSTEEMOVERZICHT

Het toegangssysteem bestaat uit vijf losse applicaties die samenwerken op basis van het MQTT-pub/sub-model en REST API. Iedere applicatie draait in een eigen container of embedded omgeving en is verantwoordelijk voor één deelstap in de verificatie- en toegangsflow. Samen vormen ze een modulair geheel dat eenvoudig scalable is.



2.2 COMPONENTEN OMSCHRIJVING

2.2.1 HUB

De hub vormt de kern van het systeem. Het is geen applicatie op zichzelf, maar bestaat uit een reeks gedockeriseerde microservices die via Docker Compose als één geheel worden gestart en beheerd.

De hub bestaat uit de volgende services:

- **MQTT Broker**
  Een Mosquitto MQTT-broker die functioneert als centrale communicatielaag tussen alle services en componenten. In de config zijn gebruikers gedefinieerd met de bijhorende permissions.

- **Facial Recognition Service**
  Een Python-applicatie die verantwoordelijk is voor het uitvoeren van gezichtsherkenning op binnenkomende webcam feed. Daarnaast beheert deze service ook gebruikersdata, zoals het toevoegen van nieuwe gezichten voor autorisatie.

- **Kiosk Service**
  Een HTTP-server die draait binnen het lokale netwerk. Bereikbar Via mDNS (bijvoorbeeld http://accesscontrol.home:3000) biedt deze service een preview van de webcamfeed. De applicatie is gebouwd in Node.js en draait een Express-server. De server stuurt de camerafeed door naar de frontend via WebSockets.

- **MQTT Service**
  Een Node.js-service die functioneert als de centrale logica van de hub. Aangezien de hub zelf geen enkele applicatie is, representeert deze service de HUB. De service is bijvoorbeeld gesubscribed op de presence-topic en detecteert welke apparaten beschikbaar zijn binnen het systeem.

- **Cloud Service**
  Een Node.js applicatie die als gateway werkt voor de communicatie tussen het systeem, en de backend via REST API.

- **Auth Service**
  De Authentication Service verwerkt de binnenkomende data om te verifiëren of de gebruiker geautoriseerd is het slot te openen.

2.2.2 VERIFICATION NODE

De Verification Node is een Python-applicatie die op een microcontroller draait en is gekoppeld aan een camera en een pinpad. Deze service verzamelt de camerafeed en de input van het pinpad en publiceert die via MQTT op de broker in de hub. Andere microservices, zoals de Kiosk Service, Recognition Service, Authentication Service, kunnen deze data vervolgens gebruiken om hun specifieke taken uit te voeren. De Verification Node beslist niet over het openen van het slot, het is enkel bedoeld voor het verzamelen van gebruikersinput.

2.2.3 LOCK

De slotsoftware is een C++ applicatie gebaseerd op het Arduino-framework en draait op een ESP32-microcontroller. De applicatie is minimalistisch opgezet om batterij te besparen en luistert enkel naa 3 topics, namelijk het openen van het slot, batterijniveau en identity presence.

2.2.4 INTERFACE

De interface, in de vorm vna een mobiele applicatie, biedt beheermogelijkheden. De app is ontwikkeld in Angular en Ionic en maakt via het lokale netwerk verbinding met het systeem. Met de app kun je gebruikers beheren, push-notificaties bevestigen voor 2FA en een live preview van de camerafeed van de Verification Node bekijken.

2.2.5 BACKEND

Het backend is een gedockeriseerde Nest.js-applicatie. Het biedt functionaliteiten zoals API-integraties, push-notificaties, webhooks en remote management. Hiermee kunnen bepaalde aspecten van het systeem op afstand worden beheerd, zonder dat er een verbinding met het lokale netwerk nodig is.

2.3 COMMUNICATIE

2.3.1 MQTT

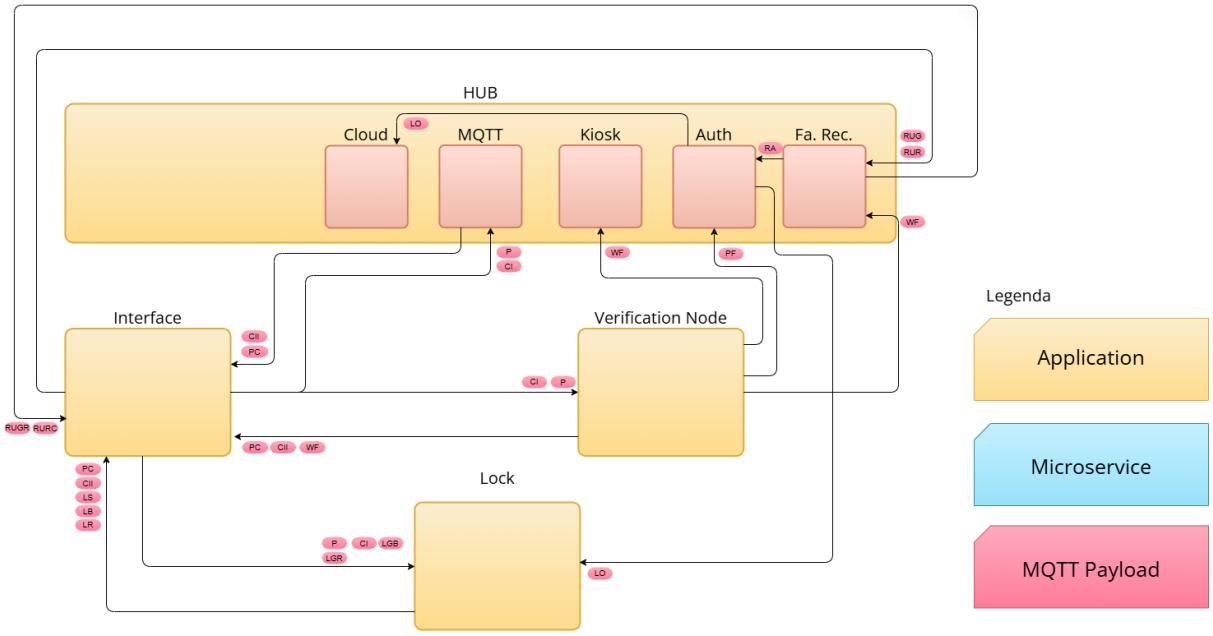
Communicatie binnen het lokale netwerk verloopt via MQTT. De MQTT-broker, die op de hub draait, beheert alle verbonden applicaties en services, inclusief hun permissions. De keuze voor MQTT is gebaseerd op uitgebreid onderzoek. Voor meer informatie zie *"Biometrisch Toegangssysteem Onderzoek"* en *"LoRa vs MQTT"*.

Topic	Index	Publishers	Subscribers	Omschrijving
Presence	P	Interface	MQTT Service (HUB), Verification Node, Lock	Gebruikt voor nodes discovery
Presence/confirm	PC	MQTT Service (HUB), Verification Node, Lock	Interface	Reactie op presence checks met unieke device identity
Client/identity	CI	Interface	MQTT Service (HUB), Verification Node, Lock	Gebruikt voor het opvragen van node identity
Client/identity/{id}	CII	MQTT Service (HUB), Verification Node, Lock	Interface	Specifieke client identity response
Webcam/feed	WF	Verification Node	Interface, Kiosk Service, Auth Service, Facial Recognition Service	Video feed van de camera
Pinpad/feed	PF	Verification Node	Auth Service	Input feed van de pinpad
Recognition/authorized	RA	Facial Recognition Service	Auth Service	Authorization trigger
recognition/user/register	RUR	Interface	Facial Recognition Service	Registratie van nieuwe gebruikers
recognition/user/register/confirm	RURC	Facial Recognition Service	Interface	Bevestiging van gebruikersregistratie
Recognition/users/get	RUG	Interface	Facial Recognition Service	Opvragen van gebruikers
Recognition/users/get/response	RUGR	Facial Recognition Service	Interface	Response met de gebruikers
Lock/open	LO	Auth Service	Lock, Cloud Service	Request om het slot te openen
Lock/get/battery	LGB	Interface	Lock	Opvragen van batterijstatus
Lock/get/rssi	LGR	Interface	Lock	Opvragen van wifi sterkte
Lock/status	LS	Lock	Interface	Statusupdate van het slot
Lock/battery	LB	Lock	Interface	Batterijstatus updates
Lock/rssi	LR	Lock	Interface	WIFI sterkte updates

2.4 DATA FLOW

In het onderstaand diagram is de route van elk MQTT topic weergegeven. De pijlen visualiseren de dependencies tussen publishers en subscribers. Zo loop er bijvoorbeeld een pijl van de interface naar de verification Node met de index P. In de bovenstaande tabel zie je dat P voor het presence topic staat, dat gebruikt wordt voor devices discovery op het lokale netwerk.

De authenticatieflow van de gebruiker verloopt als volgt: de Verification Node verzamelt camerabeelden en stuurt deze continu door naar de Facial Recognition Service. Zodra een gebruiker wordt herkend, stuurt de Facial Recognition Service een trigger naar de Auth Service. Vanaf dat moment wacht de Auth Service op invoer van de pinpad voor multi-factor authenticatie (indien nodig). Wanneer alle vereiste gegevens zijn ontvangen en de gebruiker succesvol is geverifieerd, stuurt de Auth Service een message naar de Lock om het slot te openen. Tegelijkertijd gebruikt de Cloud Service dit message ook om een webhook naar de cloud te sturen voor eventuele API integraties.



CI/CD

Het systeem maakt gebruik van een CI/CD-pipeline om testen en deployment te automatiseren via GitHub Actions. Bij het mergen van een feature branch met de production branch moet de nieuwe branch voldoen aan alle tests. Zodra een feature branch met de production branch wordt gemerged, start er automatisch een deployment flow.

Voor de backend wordt de Docker-image gebuild en gepusht naar de container registry in Azure. Deze image wordt vervolgens gebruikt als de nieuwste versie.

Voor de microservices vergelijkt de hub om de vijf minuten de lokale commit hash met de laatste commit hash op GitHub. Zodra deze niet overeenkomen, haalt de hub automatisch de nieuwste code binnen en voert een build uit van de Docker Compose.


build-and-deploy
succeeded on Apr 15 in 47s
Search logs
Set up job
Checkout code
Log in to Azure Container Registry
Build Docker image
Push Docker image
▶ Run docker push ***/nestjs-backend:latest
The push refers to repository [***/nestjs-backend]
b39bf987d721: Preparing
c8aab354a944: Preparing
7806be558335: Preparing
f90e2be28fa8: Preparing
83142697d429: Preparing
71966995f85c: Preparing
31d5ff869df5: Preparing
4f35fe5263c7: Preparing
08000c18d16d: Preparing
31d5ff869df5: Waiting
4f35fe5263c7: Waiting
08000c18d16d: Waiting
71966995f85c: Waiting
83142697d429: Pushed
b39bf987d721: Pushed
f90e2be28fa8: Pushed
71966995f85c: Layer already exists
c8aab354a944: Pushed
31d5ff869df5: Layer already exists
4f35fe5263c7: Layer already exists
08000c18d16d: Layer already exists
7806be558335: Pushed
latest: digest: sha256:027b97bfb2d7c354f630b40defb1b754f0b8b3e5e30cc1016306e03c86d08705 size: 2205
Post Log in to Azure Container Registry
Post job cleanup.
/usr/bin/docker logout ***
Removing login credentials for ***

3.0 HUB

3.1 STACK

De hub is opgebouwd volgens een microservicearchitectuur waarbij de services onderling communiceren via MQTT. Alle microservices worden afzonderlijk gedockeriseerd en vervolgens als één geheel opgebouwd en gestart met Docker Compose.

Gebruikte technologieën:

- **Docker + Docker Compose**
  Alle microservices worden gedockeriseerd als Docker-containers en als een stack beheerd via docker-compose.

- **Node.js**
  Gebruikt voor de MQTT Service, Kiosk Service, Cloud Service en Auth Service, vanwege snelle I/O-afhandeling.

- **Python**
  Gebruikt in de Facial Recognition Service vanwege de optimale ondersteuning voor AI- en beeldverwerkings libraries.

- **MQTT (Mosquitto Broker)**
  De centrale communicatielaag tussen alle services. Draait in een eigen container en is beveiligd met gebruikersauthenticatie en permissions.

- **WebSockets**
  Wordt gebruikt door de Kiosk Service om de webcamfeed in real-time door te sturen naar de frontend via een browserinterface.

- **REST API (Cloud Service)**
  Verzorgt de communicatie tussen de hub en de backend, bijvoorbeeld voor logboekverzending, webhook-acties en beheer op afstand.

3.2 COMMUNICATIE

3.3 KEUZES EN OVERVERGINGEN

//TODO
//Tech Stacks
//Internal Auth
//Remote Auth
//Overdracht
//Docker
//Overige overwegingen
//DTO's
//Network access