PLUTO PAYMENTS

Push your limit

Johann Hager, Maksym Morozov
och Carl Broman
2025

Index

1. Inledning

Detta projekt syftar till att utveckla en enkel, säker och användarvänlig betalningslösning som kombinerar hårdvara och mjukvara. Systemet består av en kortläsare (ESP32), en backend byggd i Java Spring Boot, en MySQL-databas samt en frontend skapad i Next.js. Målet är att möjliggöra säkra transaktioner med hjälp av RFID-teknik och PIN-kod, samtidigt som användaren kan logga in via webbgränssnittet för att se sina uppgifter, saldo, transaktioner och fakturor.

Projektet lägger särskild vikt vid säker kommunikation mellan enheterna genom HTTPS, HMAC och kryptering, för att säkerställa att användardata och betalningsinformation hanteras på ett tillförlitligt sätt.

2. Projektmål

Pluto Payments är en helhetslösning för smidiga och säkra betalningar. Projektet syftar till att utveckla ett komplett system som hanterar både fysiska och digitala betalningar genom en egenutvecklad kortläsare och, i framtiden, en betalningsgateway för e-handel.

Vad?
Vår lösning gör det möjligt för företag att ta emot betalningar via kortläsare och online, med säker kommunikation mellan hårdvara, backend och databas. Privatpersoner kan använda sitt Pluto Card för att genomföra köp, samla poäng och hantera sina betalningar via en användarvänlig webbapplikation.

Vem?
Tjänsten riktar sig främst till små och medelstora företag som vill ha en enkel och pålitlig betalningslösning, samt till privatpersoner som önskar ett modernt kreditsystem kopplat till sitt betalkort.

Varför?
Behovet av flexibla och säkra betalningsalternativ ökar ständigt, både i butik och online. Pluto Payments erbjuder en lösning som förenklar betalningsflödet, minskar kostnader för företag och ger kunder bättre kontroll över sina betalningar genom samlade fakturor och tydlig översikt över kredit och poäng.

3. Kravspecifikation

3.1. Funktionella krav

Krav som syftar på hur systemet ska fungera gentemot den input det får. Detta kan till exempel vara affärslogik, felhantering eller sensor/aktuator hantering.

ID	Lager	Funktion	
1	HW	Visa feedback på en LCD skärm	Finished
2	FE	Logga in som användare och se egna uppgifter	Finished
3	FE	Logga ut som användare och bara få inloggningskärm	Finished
4	FE	Visa kreditgräns och kredit kvar	Finished
5	FE	Visa poäng användaren har samlat	Finished
6	FE	Visa fakturor och deras status	Finished
7	FE	Visa transaktioner	Finished
8	HW	Waiting State - väntar på instruktioner	Finished
9	HW	Create payment - skapa en betalning för kunden	Finished
10	HW	Make payment - genomför betalning, skicka JSON-sträng till servern med kundens information	Finished
11	HW	Check payment - kontrollera att betalningen gått igenom mha HTTP responsen	Finished
12	HW	Knappsats för att navigera.	Finished
13	HW	RFID scanner för att läsa av kort	Finished
14	BE	Check pin - om fel öka wrong entries. Vid 3 wrong_entries spärra kortet. Vid rätt entry reset till 0.	Finished
15	BE	Check saldo - kontrollera saldot när vi tar emot en betalning	Finished
16	FE	Visa kort	Finished

3.2. Icke-funktionella krav

Detta beskriver hur väl systemet ska utföra sina uppgifter och innefattar ämnen som säkerhet, prestanda eller tillgänglighet. Till exempel svarstid på förfrågningar, säker förvaring av data eller strömförsörjning.

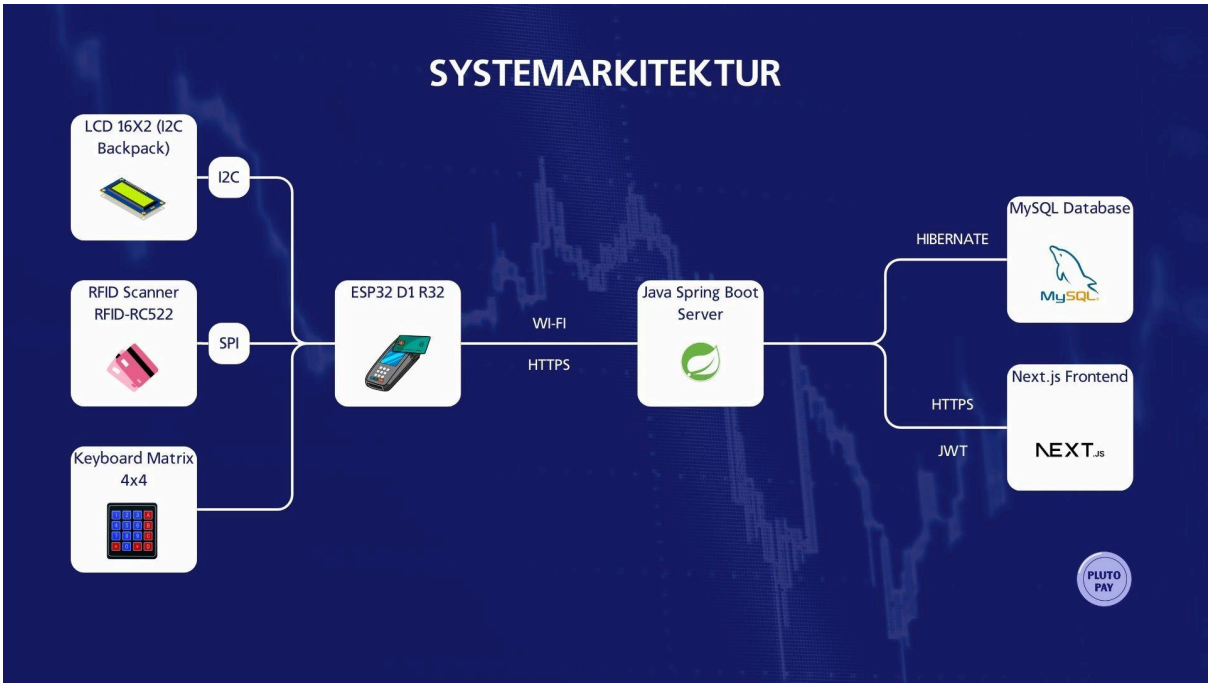
ID	Lager	Funktion	
1	HW	Skicka information med HTTPS till backend	Finished
2	FE	JWT för auktorisering i sessionsbaserad inloggning	Finished
3	HW	Hasha pinkoden med SHA256 innan den skickas	Finished
4	HW	Kontrollera att wifi är uppkopplat	Finished
6	FE	HTTPs mellan FE och BE	Abandoned
7	HW	HMAC. 1. Hasha body 2. Hasha *metod, api, content och hashad body* tillsammans med *device_secret*	Finished
8	BE	Kontrollera att HMAC är korrekt	Finished
9	BE	Meddelanden från HW får inte vara äldre än 5 minuter	Finished
10	HW	Kan ej ange en summa lägre än 0 när man skapar betalningar	Finished

3.3. Gränssnittskrav

Förklarar vilka gränssnitt som kommer användas i systemet. Här behandlas alla typer av protokoll och API:er som interfacear mellan olika system.

ID	Lager	Funktion	Status
HTTPS	ALL	Säker kommunikation	Abandoned
JWT	FE	Token säkerhet för sessionsbaserad inloggning	Finished
HMAC	HW	HMAC för att säkerställa relay	Finished
WIFI	HW	För uppkoppling till internet	Finished
HTTPS	HW	Säker kommunikation	Finished

4. Systemarkitektur



5. Moduler

Vårt val av moduler har varit begränsade till det vi tagit till oss i skolan, samt det vi stött på i projekt utanför.

5.1. Hårdvara

Hårdvarukomponenter vi har använt i projektet är:

ESP32 D1 R32

Utvecklingskort med alla nödvändiga periferier för vårt behov:

- Wi-Fi
- SPI
- I^2C
- embedTLS
- Kryptografiska acceleratorer

LCD 16x2 med I^2C

Täcker vårt behov för projektet:

- 16 karaktärer bred, två rader.
- I^2C ryggsäck som sparar in pinnar. Det krävs endast två GPIOs för att kommunicera.
- Kan drivas med 5V utan att skada ESP32an.

En LCD-skärm med 16 karaktärers bredd och två rader. Kommunicerar genom I^2C. Fungerar bra för vår lösning då den endast kräver 5V för drift, men inte för signaler och då inte orsakar någon risk för MCU:n.

RFID-RC522 modul

RFID läsare som följer standard. Eftersom ingen av oss har tidigare erfarenhet av radioprotokoll inom RFID så valde vi denna modul som hade ett färdigt bibliotek för ESP32.

- Standardiserad 13,56 MHz-läsning (MIFARE) med beprövat ESP32-bibliotek.
- SPI ger stabil och snabb kommunikation med låg CPU-belastning.

Knappsats 4x4

En enkel knappsats med 16 knappar. Fyller de behov som krävs för projektet.

- Skannar matrixen kontinuerligt vilket är dåligt för energiförbrukning. Bör bytas ut till en mer effektiv lösning.

Firmware och protokoll

- Protokoll: HTTPs till backend
- Drivrutiner för Wi-Fi, mTLS och HTTP finns i ESP-IDF toolkit med implementation.
- Drivrutiner för LCD, RFID och knappsats.
- Lättviktig logik i ESP32an, all affärslogik ligger i backend.

5.2. Backend - Java Spring Boot

Snabb uppstart och utveckling.

- Säkerhet out-of-the-box (Spring Security) och enkel TLS-konfiguration för inkommande trafik.
- Skalbart
- Ett stort ekosystem med bibliotek och dependencies via Maven.

Ansvar i modulen:

- API för enheter & frontend (REST).
- Auktorisering (HMAC kontroll och mTLS med godkända certifikat)
- Sessionbaserad inloggning med tokens.
- Validering av affärslogik.

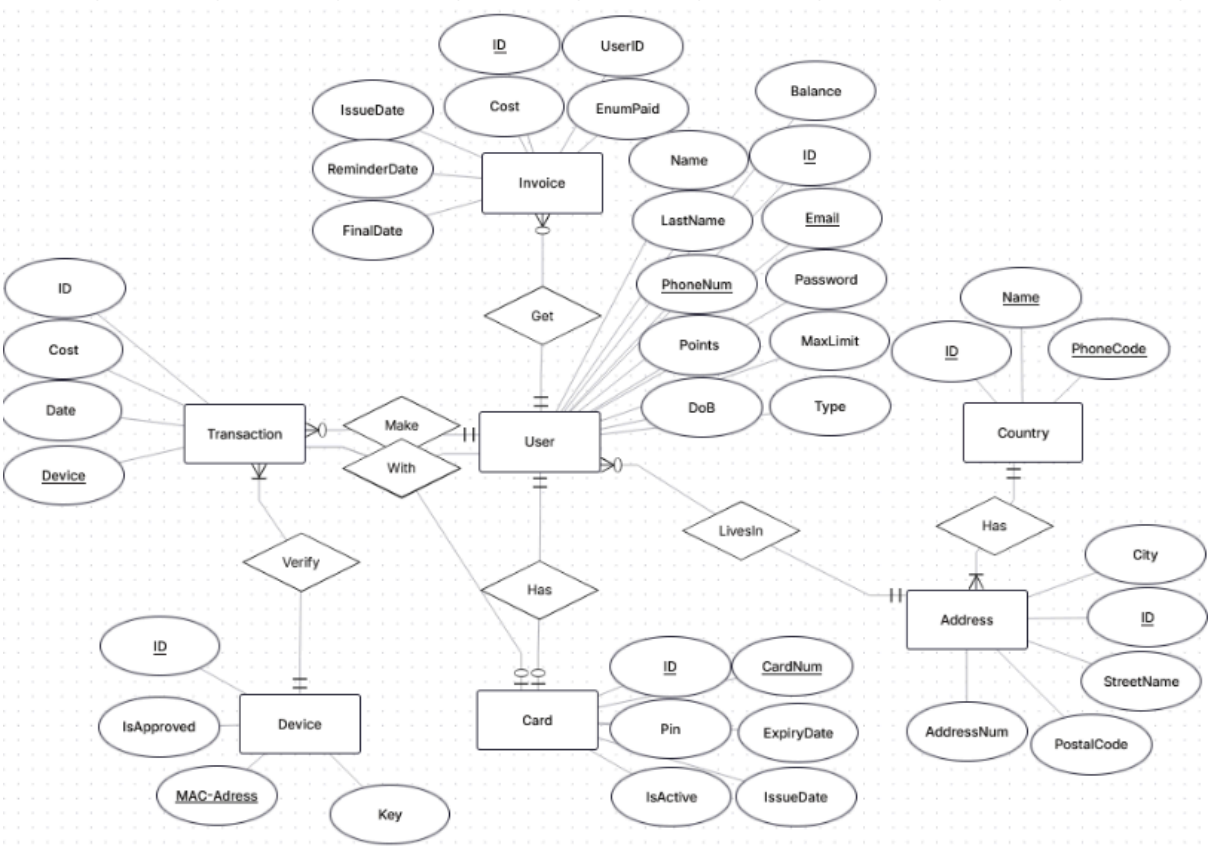
5.3. Databas - MySQL

Vi valde MySQL eftersom att det kommer göras många sorterade sökningar i databasen, både vid betalningar och när kunder använder webbapplikationen.

Det kan även vara en idé att komplettera med en tidsseriedatabas i framtiden för loggning och trendanalys

ER-Diagram

Vi tog beslutet att användarens konto står för balansen och kan koppla flera olika kort till kontot. Utöver det



5.4. Frontend - Next.js

Next.js

- Ger ett modernt React-ramverk med inbyggd routing och server-side rendering (SSR).
- Förbättrar laddningstid, SEO och möjliggör skyddade sidor baserat på sessioner.
- API Routes användes för enklare backend-logik utan separat server.

TanStack Query (React Query)

- Effektiv hantering av serverdata med cache, refetch och felkontroll.
- Håller gränssnittet synkroniserat med backend och minskar behovet av egen state-hantering.

Axios

- Flexibel HTTP-klient med interceptors för automatisk token- och felhantering.
- Förhindrar dubbla anrop och hanterar autentiseringsfel centralt.

Tailwind CSS

- Snabbt och konsekvent sätt att bygga moderna gränssnitt.
- Möjliggör responsiv design direkt i komponenterna utan externa CSS-filer.
- Ger enhetlig stil och enkel temahantering i hela applikationen.

Spring Security (sessionsbaserad inloggning)

- Starkt säkerhetsramverk med serverlagrade sessioner och `HTTP-only` cookies.
- Lätt att återkalla sessioner och tillämpa roll- eller behörighetsstyrd åtkomst.
- Passar bättre än JWT i betalningssystem tack vare omedelbar revokering och inbyggt CSRF-skydd.

6. Förbättringsmöjligheter

Om vi hade haft mer tid till att genomföra projektet, skulle vi vilja implementera följande lösningar i de olika delarna av stacken.

6.1. Hårdvara

Förbättrad kortkontroll
Det nuvarande systemet har begränsad verifiering av RFID-kortens giltighet. En förbättring vore att implementera en starkare kontroll av kortets kompatibilitet mot terminalen. Genom att använda ett mer avancerat bibliotek för MIFARE- och RFID-protokoll, alternativt stöd för NFC (t.ex. PN532), kan terminalen identifiera och validera kort med högre tillförlitlighet samt stödja fler typer av kort.

Förbättrad skärm
Den nuvarande LCD-skärmen med I2C-gränssnitt fungerar, men är känslig för timingfel och kan visa skräpvärden vid störningar i kommunikationen. En uppgradering till en OLED-skärm med högre upplösning och inbyggd buffert skulle öka både stabilitet och läsbarhet. En OLED-display kan även möjliggöra bättre användargränssnitt med symboler, animationer och statusindikatorer.

Integrerad PCB-design
I dagsläget består systemet av flera separata komponenter, vilket gör konstruktionen otymplig och svår att reproducera. Att utveckla ett anpassat kretskort (PCB) skulle samla alla komponenter i en kompakt och robust design. Detta skulle minska risken för lösa kopplingar, förbättra tillförlitligheten och göra produkten mer lämpad för testning och serieproduktion. En färdig PCBA (Printed Circuit Board Assembly) kan även förberedas för montering i ett skyddande chassi.

Utökad knappsats
Den befintliga 4x4-knappsatsen möjliggör endast inmatning av siffror, vilket begränsar funktionaliteten. Ett förbättrat system med stöd för bokstavsinmatning, exempelvis genom T9 eller multitap, skulle göra det enklare att konfigurera nätverk, ange användarnamn eller felsöka direkt på enheten. Alternativt kan en liten pekskärm övervägas för framtida versioner.

Transaktionscache
För att kunna hantera nyligen genomförda betalningar mer flexibelt, kan en lokal transaktionscache implementeras i hårdvaran. Detta skulle möjliggöra att användaren enkelt kan avbryta, uppdatera eller kreditera de senaste transaktionerna direkt från terminalen utan att behöva kommunicera med backend i realtid.

Utökade användarfunktioner
Terminalen kan förbättras med fler användarfunktioner, såsom möjligheten att välja Wi-Fi-nätverk, ändra inställningar, se diagnostik och hantera anslutningsproblem. Istället för att låsa hela skärmen vid förlorad uppkoppling kan systemet visa en enkel statusikon som indikerar nätverksstatus.

Fler uppkopplingsmöjligheter
För att öka stabilitet och flexibilitet kan hårdvaran utökas med stöd för Ethernet och eventuellt mobil uppkoppling (4G/LTE) via ett modulärt gränssnitt. Det skulle minska beroendet av Wi-Fi och ge bättre tillgänglighet i miljöer med svag trådlös signal. Till exempel på marknader eller i mobila enheter som foodtrucks.

Enhetskontroll
Implementera kontroll av enhetens olika värden som temperatur, minnesanvändning och uppkopplingsstatus. Detta för att den själv ska kunna till exempel stängas av automatiskt vid risk för överhettning, rensa minnet eller uppmana användaren till att koppla upp sig till internet.

Strömförsörjning
Utöka dagens micro-USB-matning med:

- **Sömlägen**
  använd ESP32 light/deep sleep. Den väcks via en knapp och sätts igång efter en användar-angiven period av inaktivitet.
- **Batteribackup**
  Integrera ett LiPo-batteri som möjliggör drift vid flytt och vid korta strömavbrott. Implementera en IC för laddning av batteri som kontrollerar under/överladdning och överström när enheten är kopplat till en strömkälla.
- **Batteriövervakning**
  Kontrollera och uppskatta batteriförbrukning, samt rapportera statusen till användare genom grafisk feedback på skärmen.

Säkrare enhet
Aktivera Secure Boot på ESP32an så att man inte kan köra otillåtna program, samt kryptera flashminnet. Vid tillverkning skapas även en permanent privat nyckel i enheten som används vid bootstrapping.

Fleet Management
Bygga en pipeline för att hantera enheter från produktion -> bootstrapping -> enrollment -> rotation -> EOL. För att säkerställa att endast behöriga enheter har tillgång till systemet.

6.2. Backend

Backend-delen av systemet har fungerat stabilt under utvecklingen, men det finns flera förbättringsmöjligheter som skulle kunna öka både säkerhet, prestanda och skalbarhet. En viktig förbättring vore att implementera mer omfattande loggning och övervakning med hjälp av verktyg som **Prometheus** och **Grafana**. Detta skulle möjliggöra realtidsinsikter i systemets hälsa, upptäckt av flaskhalsar och snabbare felsökning vid problem.

En annan potentiell förbättring är att införa **caching** med exempelvis **Redis** för att minska belastningen på databasen vid återkommande förfrågningar, såsom hämtning av användar- eller transaktionsdata. Det skulle ge kortare svarstider och en mer responsiv användarupplevelse.

API:et kan vidareutvecklas med **bättre felhantering** och mer konsekventa svarskoder enligt REST-standard, vilket underlättar integration med framtida system och gör felsökning enklare. Det kan även vara aktuellt att införa **versionering av API:et**, så att äldre klienter kan fortsätta fungera även när nya funktioner tillkommer.

Säkerhetsmässigt kan backend stärkas genom **rate limiting**, förbättrad **certifikathantering** och regelbunden **rotation av HMAC- och JWT-nycklar**. Stöd för **mTLS** mellan interna tjänster skulle ytterligare skydda kommunikationen från obehörig åtkomst.

Slutligen kan utvecklingsprocessen effektiviseras genom att bygga ut **CI/CD-pipelinen** för automatiska tester, kodgranskning och kontinuerlig leverans. Med fler **enhetstester och integrationstester** säkerställs att nya ändringar inte påverkar befintlig funktionalitet. Detta skulle ge en mer robust backend som är lättare att underhålla, skala upp och vidareutveckla i takt med att systemet växer.

6.3. Frontend

Frontend-delen har fungerat bra under utvecklingen, men det finns flera saker som skulle kunna förbättras för att göra systemet snabbare och mer stabilt. En tydlig förbättring vore att optimera prestandan genom att använda fler av Next.js inbyggda funktioner, som image-optimering, dynamic imports och ISR/SSG. Det skulle kunna minska laddningstider och göra upplevelsen smidigare för användaren.

Ett annat område är felhantering. Genom att lägga till ett verktyg som Sentry för att logga fel i realtid, och hantera nätverksfel centralt via Axios-interceptors, skulle det bli enklare att upptäcka och åtgärda problem.

Datahanteringen med TanStack Query kan också förbättras genom tydligare struktur på “query keys” och cache-strategier, så att gränssnittet hålls uppdaterat utan onödiga anrop.

På designsidan kan vi utveckla ett litet designsystem ovanpå Tailwind CSS för att få mer konsekvent stil och återanvändbara komponenter. Dessutom vore det bra att lägga mer fokus på tillgänglighet (a11y), till exempel bättre kontraster och tangentbordsnavigering.

Slutligen kan vi förbättra kodkvaliteten genom fler tester och strängare TypeScript-inställningar. Med en enkel CI-pipeline som kör lintning och tester innan deploy blir det lättare att upptäcka fel tidigt och hålla frontenden stabil även när nya funktioner läggs till.

7. Sammanfattning

Vi uppnådde de centrala målen för projektet inom den utsatta tidsramen och lyckades skapa ett proof-of-concept för betalningslösningen som integrerar hårdvara och mjukvara. Systemet uppfyller grundkraven för kommunikation, säkerhet och användarupplevelse. Inför en framtida produktionssättning finns det dock flera förbättringar som bör genomföras, bland annat optimering av hårdvaran, utökad loggning och övervakning i backend samt vidareutveckling av gränssnittet för bättre prestanda och användbarhet.