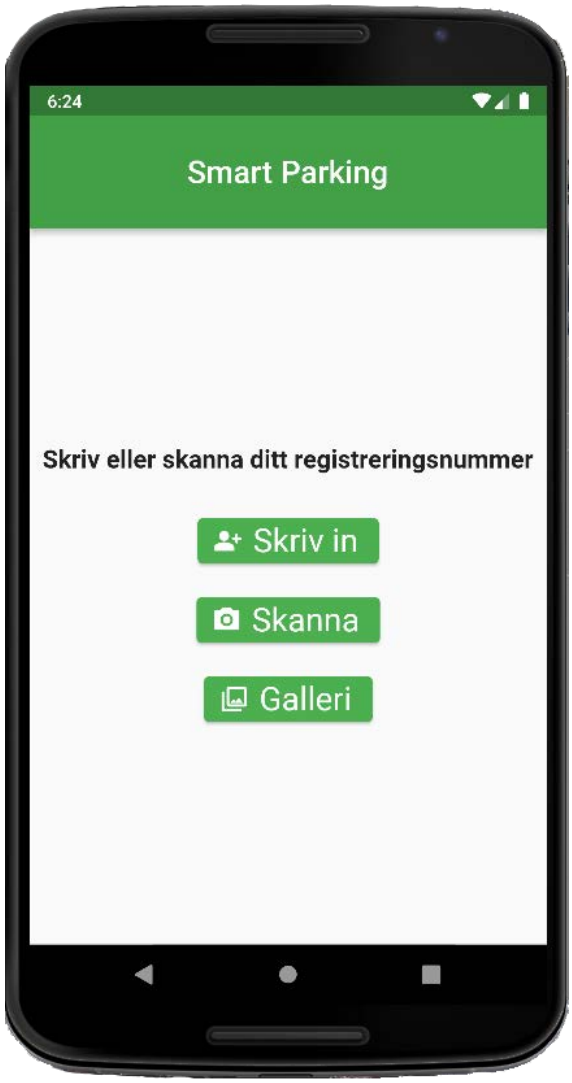
CHALMERS



SmartParking

Ett automatiserat inpasseringssystem för parkering

Examensarbete inom Data- och Informationsteknik

Jimmie Berger
Hamza Kadric

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2022

EXAMENSARBETE

SmartParking

Ett automatiserat inpasseringssystem för parkering

Jimmie Berger
Hamza Kadric

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg, Sverige 2022

SmartParking
Ett automatiserat inpasseringssystem för parkering
Jimmie Berger
Hamza Kadric

Handledare: Sakib Sistek, Institutionen för Data- och Informationsteknik
Examinator: Jonas Duregård, Institutionen för Data- och Informationsteknik

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
412 96 Göteborg
Sverige
Telefon: +46 31 772 1000

Omslag: Visualisering av mobilapplikationen SmartParking

Institutionen för Data- och Informationsteknik
Göteborg, Sverige 2022

Sammanfattning

Nu för tiden finns det små, billiga och energisnåla minidatorer som är tillräckligt kraftfulla för att ha allsidig funktionalitet såsom trådlös kommunikation, sensorer för avståndsläsning eller kameror som kan ta högupplösta bilder. Dessa minidatorer är även relativt enkla att programmera och kan på så vis lösa alla tänkbara och otänkbara problem, vilket är något som vi ofta strävar efter i dagens moderna samhälle där allting ska gå snabbt och smidigt. Ett sådant problem är parkering, ett tröttsamt måste i dagens tätbefolkade storstäder. Syftet med det här projektet var att skapa ett system för att antingen manuellt via en applikation i mobilen starta, avsluta och betala för parkering alternativt med hjälp av sensorer, kameror och bildigenkänning automatisera detsamma. Resultatet som vi kom fram till var att det är möjligt med relativt små medel och tidsåtgång att lösa båda problemen genom att skapa en fungerande mobilapplikation och även ett grundläggande system som lyckas läsa av registreringsskyltar via bildigenkänning och därmed automatisera hantering av parkering.

Nyckelord: Arduino, automatisering, bildigenkänning, maskininlärning, minidatorer, servomotor

Abstract

Nowadays, there are small, inexpensive and energy-efficient minicomputers that are powerful enough to have versatile functionality such as wireless communication, distance reading sensors or cameras that can take high-resolution images. These minicomputers are also relatively easy to program and are able to solve all kinds of problems, which is something we strive for in todays modern society where everything has to happen fast and with minimal effort. One such problem is parking, an annoying problem in todays densely populated megacities. The purpose of this project was to create a system to either manually via an application in the phone start, finish and pay for parking or alternatively with the help of sensors, cameras and image recognition automate the same. The result we reached was that it is possible with relatively small funds and time required to solve both problems by created a working mobile app and also an elemental system that is able to interpret licenseplates and via image recognition and thus automate the parking process.

Keywords: Arduino, automation, image recognition, machine learning, minicomputers, servomotor

Förord

Detta examensarbete är utfört av Jimmie Berger och Hamza Kadric på Datateknikprogrammet vid Chalmers tekniska högskola under höstterminen 2021.

Vi vill tacka alla de kreatörer som med sina tips, råd och guider har hjälpt oss hitta lösningar, utveckla och göra framsteg med arbetet. Ett stort tack till vår handledare Sakib Sistek för möjligheten att arbeta med just detta projekt samt för all inspiration och stöd han gav oss under arbetets gång.

Jimmie Berger & Hamza Kadric
Januari 2022
Göteborg

Terminologi och förkortningar

Förklaring av relevanta ord, termer och förkortningar.

API - Application Programming Interface, ett gränssnitt mot vilket man kan kommunicera med ett program eller system.

Frontend - ”Framsidan” av ett program/system, de grafiska element en användare ser och kan interagera med.

Backend - ”Baksidan” av ett program/system, datahantering, kommunikation med andra system och liknande som sker bakom kulisserna.

CRUD - Create, Read, Update, Delete. De fyra vanligaste funktionerna som man använder för datahantering.

GIT - Git är ett distribuerat versionshanteringssystem, vilket innebär att flera personer kan arbeta i det samtidigt (distribuerad) och att det hanterar olika versioner av filer (versionshantering).

iOS - iOS är ett mobilt operativsystem skapat och utvecklat av Apple Inc.

I/O-enheter - I/O betyder ”Input/Output” och möjliggör kommunikation mellan exempelvis en krets och ett annat informationssystem. Kan vara både analog och digital kommunikation.

JSON - JavaScript Object Notation, en öppen filstandard och datakommunikationsformat som är utformat att vara lättläsligt för människor.

REST - Representational state transfer, den arkitektur som internet (WWW) är uppbyggd enligt. Används även för kommunikation mellan webbläsare och webbsidor, samt mellan olika program.

SDK - Software Development Kit, en samling av verktyg paketerade för utveckling av en specifik mjukvara.

SQL - Structured Query Language, är ett standardiserat programmeringsspråk som används för att hantera och utföra olika operationer i relationsdatabaser.

Server - En dator vars syfte är att tillhandhålla tjänster eller funktionalitet.

SSL - Secure Sockets Layer, kryptografiskt protokoll för säker kommunikation över nätverk.

Tech stack/Lösningsstack - Den kombination av teknologier som används för att bygga och driva en applikation eller ett projekt.

Innehållsförteckning

Lista över figurer

1 Inledning

I detta kapitel beskrivs projektets bakgrund, syfte, mål och avgränsningar.

1.1 Bakgrund

Vi har velat skapa en parkeringstjänst som underlättar in- och utpassage till en parkering och valde att göra detta till vårt examensarbete. De bilägare som är registrerade i systemet ska automatiskt släppas in till parkeringen genom att en grind öppnas. En sådan tjänst ska även kunna hantera betalning på ett smidigt och användarvänligt sätt.

1.2 Syfte

Examensarbetet går ut på att göra ett system som genom att skanna av registreringsnummer på bilar kan avgöra ifall tillträde till exempelvis en parkeringsplats, skall tillåtas eller inte, baserat på om användaren betalat. Tanken är att visa upp en ny variant av en parkeringslösning där användaren via en applikation i förväg registrerar sin plats på parkeringen.

1.3 Mål

Målet med projektet är att skapa en demoversion av en automatiserad parkeringstjänst som innefattar skanning av registreringsskyltar, styrning av passeringsgrind samt presentation i en applikation. De moment som vi ämnar att utföra i arbetet är följande: Undersöka möjlighet att med automatisk skanning av registreringsnummer med hjälp av kamera och betalningstjänst styra öppning och stängning av en motorstyrd grind. Det skannade registreringsnumret kommer sen att matchas mot databasen över godkända och ej godkända registreringsnummer. Undersöka möjligheten att presentera lösningen i form av en mobilapplikation som kopplas ihop med databasen.

1.4 Avgränsningar

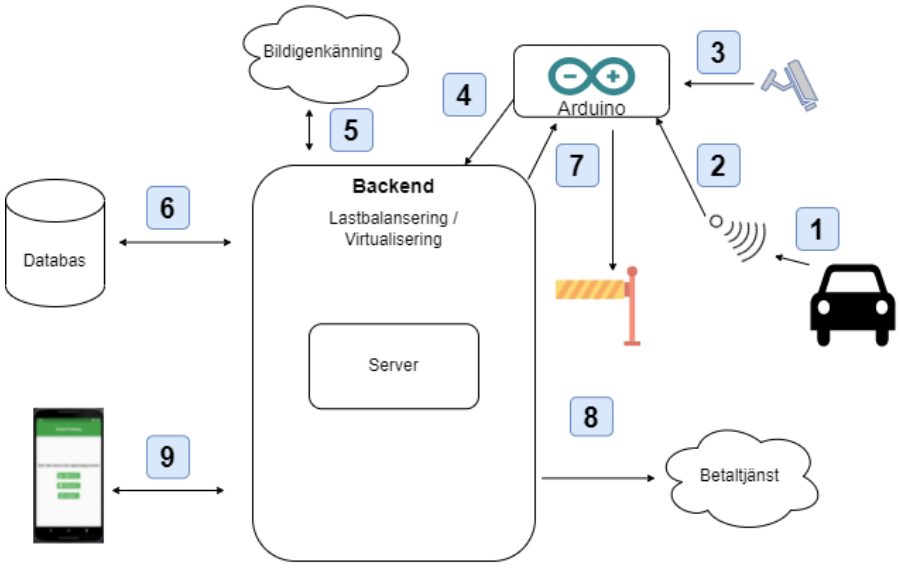
Projektet omfattar endast mjukvaruutveckling. Projektet inkluderar inte utveckling av hårdvara. Viss hårdvara som är tänkt att användas i arbetet, såsom hårdvara för styrning av passeringsgrind, kommer att lånas från Chalmers. Då projektet är en demoversion, kommer endast ett begränsat antal parametrar att beaktas vid passering till parkeringen. Dessa parametrar är registreringsskyltar, tid för in- och utpassering och eventuella variabler som krävs för betalning.

2 Teknisk bakgrund

I detta kapitel förklaras de verktyg och tekniska termer som har använts i projektet.

2.1 Överblick

I figur 1 har vi en grafisk visualisering av hur vi har tänkt att slutprodukten ska vara konstruerad och vilka fundamentala funktionaliteter samt vilken kommunikation som bör finnas med.



Figur 1: Flödesschema för plattformen SmartParking

1. Ett fordon närmar sig en parkeringsgrind och upptäcks av en sensor.
2. Sensorn skickar en signal till Arduinon att den upptäckt något.
3. Arduinon tar en bild med kamera.
4. Arduinon skickar bilden till backend.
5. Backend skickar bild till bildigenkänningsmjukvaran/molntjänsten, som returnerar registreringsnummer i textformat.
6. Backend kollar i databasen om registreringsnumret redan existerar, och i så fall om det står som parkerat eller inte.

   (a) Om registreringsnumret inte var registrerat eller inte stod som parkerat, så uppdaterar backend databasen att det registreringsnumret nu har påbörjat parkering.
   (b) Om registreringsnumret stod som parkerat, så uppdaterar backend databasen att det registreringsnumret nu inte längre är parkerat.

7. Backend meddelar Arduinon att öppna grind för in-/utfart som då öppnar grinden.
8. Backend meddelar betaltjänst att användare X ska betala för parkering.
9. Man kan också använda mobilapplikationen för att manuellt skicka antingen registreringsnumret som text alternativt bild på registreringsskylten och processen fortsätter likt ovan.

2.2 Mjukvara

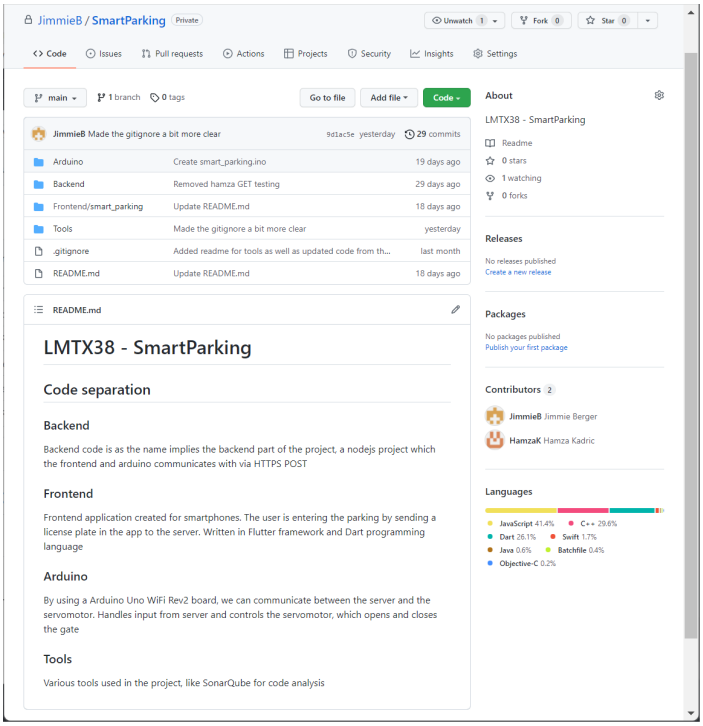
I denna del nämns de program, tekniker och tjänster som användes för att skapa plattformen.

2.2.1 Git

Git är ett open-source system för distribuerad versionskontroll [13], vars syfte är att två eller fler personer ska kunna samarbeta i programmeringsprocessen och samtidigt hålla koll på alla ändringar samtidigt som man undviker att olika personers ändringar krockar med varandra.

2.2.2 GitHub

GitHub är en molnplattform som bygger på Git och lite förenklat skulle man kunna säga att Git är backend medan GitHub är ett frontend för det distribuerade versionskontroll systemet. Utöver Git-funktionaliteten erbjuder GitHub ett flertal funktioner såsom lagring av ens projekt i molnet, automatiseringsfunktioner, CI/CD (Continuous Integration / Continuous Deployment), notifikationer med flera [14]. Sammanfattat är det en plattform som förenklar samarbete och diskussioner i utvecklingsprojekt.

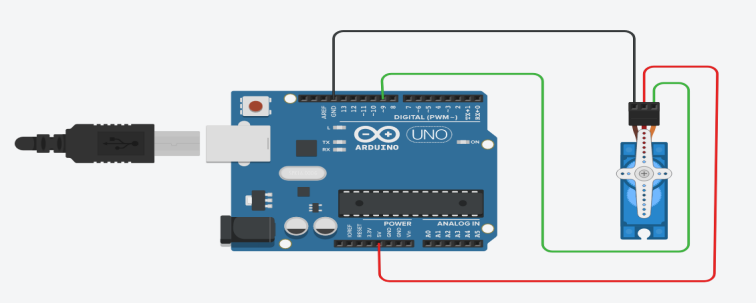


Figur 2: Github repository för projektet SmartParking

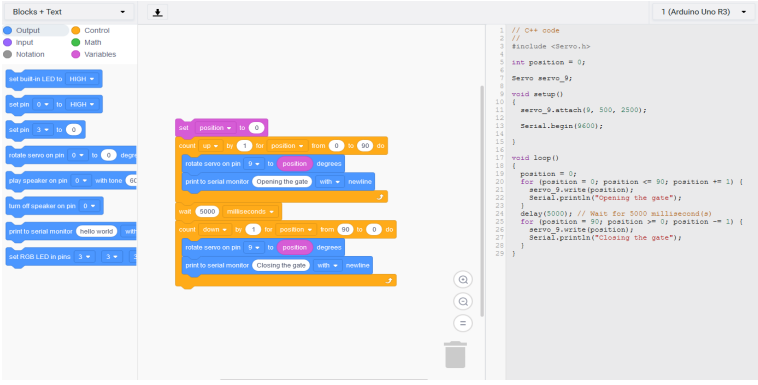
2.2.3 Tinkercad

Innan vi kopplade ihop Arduinon med servomotorn så valde vi först att simulera upplägget i det webbaserade programmet Tinkercad. I Tinkercad kan man 3D-modellera allt från datorstödd konstruktion (CAD), blockbaserad programmering till att simulera elektronisk utrustning. Genom att använda sig utav Tinkercad undvek vi att riskera förstöra någon av den tekniska hårdvaran och kunde istället smärtfritt implementera vårt system. Första steget var att välja rätt komponenter och koppla ihop till respektive in- och utgång.

Därefter ska man programmera systemet så att det utför de kommandon som efterfrågas. Detta kan göras med hjälp av blockprogrammering som är grafisk programmering med block som exekverar de kommandon man applicerar varje block med, som i sin tur skapar ett fungerade system. Slutligen kan man simulera programmet för att hitta eventuella fel och justera koden så den utför efterfrågat arbete. Tinkercad översätter automatiskt kodblocken till C++ kod som man senare kan implementera på den riktiga hårdvaran, i vårt fall på Arduino UNO och servomotorn.



Figur 3: Simulering i Tinkercad med ihopkopplad krets



Figur 4: Blockprogrammering och automatisk översättning till C++ kod

2.2.4 Flutter

En av de viktigaste delarna i vårt projekt är mobilapplikationen och för att få maximal utväxling på vår arbetsinsats så vill vi använda en plattform och SDK (software development kit) som kan generera applikationer både för Android och iOS utifrån samma kod. Det finns en plattform vid namnet React Native, skapat av Meta Platforms, som erbjuder denna möjlighet och som är ett av de mest populära verktygen för skapande av mobilapplikationer. Vi har dock valt ett annat alternativ, nämligen nya plattformen Flutter som också genererar appar för Android och iOS utifrån en enda kodbas. Det är Google och en community av utvecklare som har skapat detta ramverk.

Det finns flera anledningar till att vi valde Flutter för att skapa vår applikation. I Flutter används Dart, ett nyutvecklat och lättlärt programmeringsspråk som är snarlikt Java med typer, klasser, funktioner och arv men som även inkluderar dynamisk variabeltyp i stil med JavaScript. Kortfattat så kan man säga att allt är uppbyggt som *widgets*, som i sin tur kan ha en eller flera *subwidgets*. En *widget* kan man förenklat säga är en behållare som innehåller data, och med detta programmeringstänk att allt består av behållare gör apputvecklingen okomplicerad och mer rakt på sak. Plattformen Flutter är tämligen ny och ihop med Dart har systemet flera unika banbrytande egenskaper, till exempel realtidsexekvering och så kallad *hot reload* som innebär att ändringar i koden syns direkt i applikationen som körs. Plattformen har en bra layout metodologi som lånats från responsiv webbdesign som är en utvecklingsstrategi som gör att en och samma webbsida fungerar bra på många olika enheter. Applikationerna som skapas med Flutter ger en bra användarupplevelse som är snabb och smidig.

2.2.5 MongoDB

En databas är en strukturerad samling av lagrad data som är organiserad för att man så snabbt och effektivt som möjligt ska kunna filtrera och komma åt rätt data. MongoDB är en dokumentorienterat databasplattform [1] som lagrar dokument i BSON format (Binary JSON eller Binary JavaScript Object Notation) [2], snarare än traditionella relationsdatabaser som lagrar data i tabellform. JSON är ett öppet fil-/dataformat som använder vanlig text lagrad i attribut-värde par, som figur 5 visar ett exempel på.

```
db.users.insertOne(           <- samling
  {
    "firstName":"John",       <- attribut:värde  }
    "lastName":"Doe",         <- attribut:värde  } dokument
    "email":"john@doe.com"    <- attribut:värde  }
  }
)
```

Figur 5: Exempel på en JSON insättning i MongoDB

Det går även att överföra filer och annan data på liknande sätt, vilket vi nyttjar när vi skickar foton på registreringsskyltar från mobilapplikationen i frontend för att tolkas av bildigenkänningsmjukvaran i backend. Vi valde att använda MongoDB då det tillsammans med ExpressJS, AngularJS och NodeJS bildar lösningsstacken **MEAN**[3], där de fyra bokstäverna står för just **M**ongoDB, **E**xpressJS, **A**ngularJS, **N**odeJS. Vi har dock valt att inte använda AngularJS utan skapar vårt frontend via Flutter istället och kör då på den löst kallade **MEN**-stacken. Det faktum att MongoDB har fått en egen förkortning i MEAN-stacken, och även i andra lösningsstackar, visar på plattformens popularitet samt att det finns mycket dokumentation och resursinformation för dess användare.

2.2.6 ExpressJS

ExpressJS är ett modulärt applikationsramverk för NodeJS som är till för att underlätta skapandet av webbapplikationer och APIer i NodeJS. Det simplifierar mycket kod som annars är onödigt komplext att skriva för hand i NodeJS, alternativt att vi själva hade varit tvungna att skapa olika komponenter för att hantera HTTP-/HTTPS förfrågningar, SSL, rutthantering, sessioner, felhantering och liknande.

2.2.7 NodeJS

NodeJS är en modern open-source plattform byggd på Google Chromes V8 Javascriptmotor, som är till för att exekvera kod skriven i JavaScript. NodeJS är designat för att vara skalbart och modulärt [4], som snabbt blivit allt mer populärt [5]. Istället för att använda mer traditionella webbplattformar som Apache, Nginx eller IIS, som hanterar både frontend och backend, så passar NodeJS vår applikation bättre då det primärt är backend som plattformen behöver hantera då vi använder mobilapplikation som frontend. Det går dock att använda även NodeJS till att skapa och visa frontend för användare, framförallt när man har dynamiska webbsidor som ändrar utseende baserat på olika faktorer snarare än statiska sidor som är oföränderliga. Därtill är NodeJS skapat för att vara skalbart och används idag av flera stora företag såsom Netflix, LinkedIn, Uber, Paypal, etc.. Då är NodeJs ett lämpligt val om vår applikation skulle bli väldigt populär och snabbt skulle få behov av mer datorkraft [6].

2.2.8 REST

Representational state transfer, REST, är ett begrepp för hur kommunikation mellan datorer kan ske över webben och är en grundpelare i hur internet och webbplatser fungerar. Det består utav följande principer [7]:

- Klient-Server arkitektur: I klient-server designmönstret har man separerat användargränssnitt i en klient del, och datahantering en serverdel vilket klientdelen förlitar sig på.

- Tillståndslöshet: En mottagare sparar ingen information om något tillstånd utan varje enskilt meddelande hanteras helt fristående utan att vara kopplat till något tidigare eller senare meddelande.
- Cachemöjlighet: Att kunna lagra svar på förfrågningar, vilket är vanligt att använda sig utav för att snabba upp responstid om man exempelvis besöker en hemsida flera gånger och då inte behöver ladda ner stora bildfiler som är oförändrade vid varje besök.
- Lager: Att kommunikation kan ske genom flera lager, en klient behöver inte nödvändigtvis skicka sin förfrågan direkt till slutdestination och få svar direkt därifrån utan det är vanligt att man har flera lager av tjänster, till exempel lastbalanserare för att på så vis kunna förbättra skalbarheten likväl som tillförlitligheten eller brandväggar för ökad säkerhet.
- Enhetligt gränssnitt: Varje förfrågan ska vara utformad på samma vis, där det går att utläsa ur förfrågan vilken resurs det är som efterfrågas, att datan i förfrågan är tillräcklig för att med enbart den datan kunna modifiera förfrågan, att beskrivningen av datan är självförklarande och tydlig samt att klienten ska kunna hantera de svar som ges.

När ett system delvis eller helt uppfyller principerna ovan så anses det vara RESTful, vilket vår applikation är.

2.2.9 RESTFul

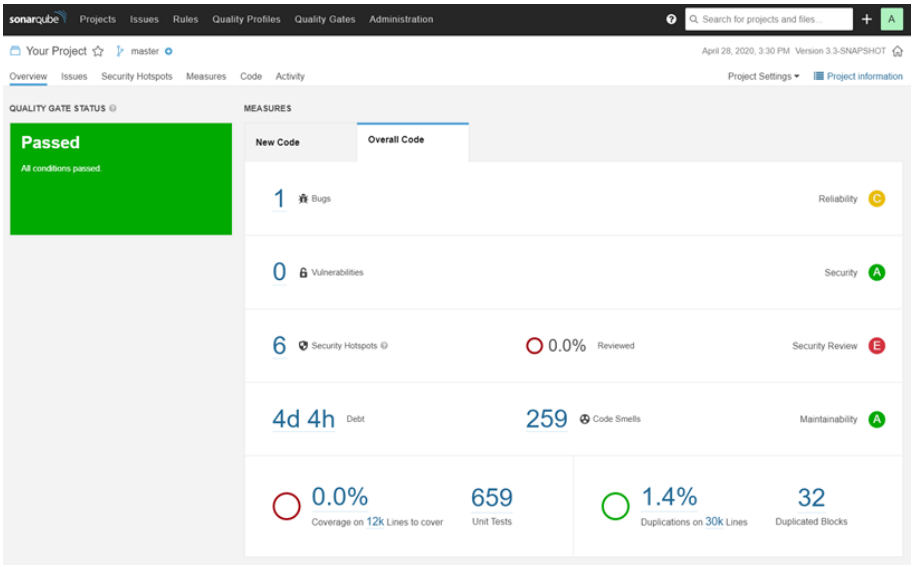
RESTFul eller RESTful API som det också kallas är generellt vad som syftas på när man pratar om REST i webbapplikationssammanhang, vilket är hur en klient kommunicerar med en server rent praktiskt. Kommunikationen sker över HTTP eller HTTPS och använder sig utav ett HTTP-kommando, följt av ett meddelande [8]. Exempel på kommandon är följande

- POST
- GET
- PUT
- DELETE

Dessa kommandon motsvarar de fyra standardfunktionerna som förväntas utav ett API, Create, Read, Update och Delete, i vad som vanligtvis brukar kallas CRUD [9]. Utöver ett kommandot så består ett meddelande av en Header samt en Body. I headern specificeras olika tekniska attribut, exempelvis längden på meddelandet eller vad för format meddelandet är i. Det finns flera olika format, JSON, XML, CSV eller exempelvis ren text, men vanligtvis används JSON, som trots sitt namn inte är bundet till något specifikt språk och det är även det som vi har använt oss utav i vår applikation. Slutligen har vi Body, vilket inte är något som nödvändigtvis behövs utan kan lämnas tomt, men i vårt fall är vad som innehåller vad vi faktiskt vill kommunicera.

2.2.10 SonarQube

SonarQube är en open-source plattform för att underlätta skapandet av bra och säkra applikationer och samtidigt bibehålla hög kodkvalité [10]. Det har stöd för de flesta moderna och populära språk och det går även att integrera med utvecklingsmiljöer såsom Visual Studio, Eclipse och GIT. Programmet utför en statisk kodanalys på en kodbas och därigenom hittas automatiskt eventuella buggar och säkerhetsrisker. SonarQube går även igenom kodstandarden för det specifika språk som analyseras och ger en bra överblick på hur väl koden är skriven för just det språket. När analysen är färdig får man en rapport som listar alla upptäckta fel, brister eller noteringar, samt ofta utförliga förklaringar på varför något är fel och förslag på hur man kan åtgärda det.



Figur 6: Exempel på hur en SonarQube rapport ser ut

2.2.11 OpenALPR

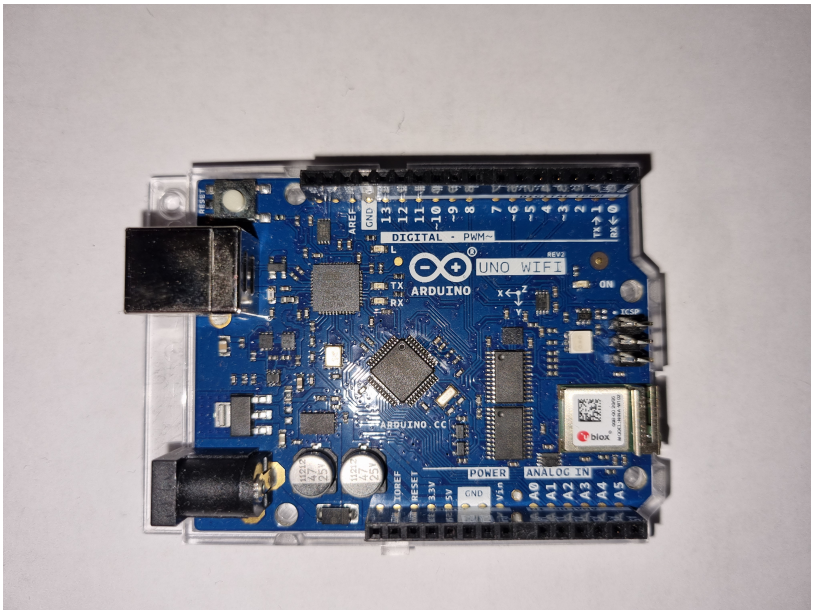
OpenALPR (Open Automatic License Plate Recognition) är ett open-source kodbibliotek för att analysera registreringsskyltar. Det bygger på OpenCV (Open Computer Vision) samt Tesseract OCR (Optical Character Recognition), det förstnämnda är ett bibliotek ursprungligen utvecklat av Intel i början av 2000-talet [11] för, som namnet antyder, ge en dator *"mänsklig syn"* och det senare har rötter som är ännu äldre, utvecklat under 1980-talet av Hewlett-Packard (HP) [12], är till för att med en dators hjälp tolka tecken utifrån en bild.

2.3 Hårdvara

I denna del nämns den hårdvara som användes för att skapa en prototyp av plattformen.

2.3.1 Arduino UNO

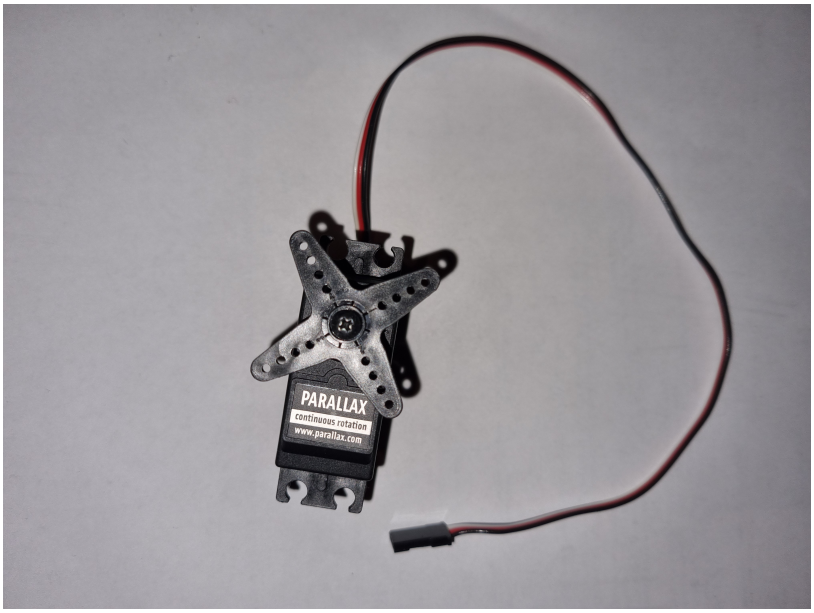
För att kunna simulera en grind till parkeringsplatsen som ska kunna öppnas och stängas vid order så har vi valt att använda en Arduino UNO WiFi Rev 2. Denna mikrokontroller är en kraftfull enkretsdator som kan styra och kommunicera mellan diverse elektronisk utrustning. Arduino är så kallad öppen hårdvara vars mål är att göra elektronikanvändning mer enkel och tillgänglig för nybörjare likväl som för professionella. Hårdvaran består bland annat av en Atmel AVR mikroprocessor, flera digitala och analoga I/O-enheter samt inbyggd WiFi-funktionalitet. Programmeringen av hårdvaran skrivs med vissa förenklingar i C++ i Arduinos egna integrerade utvecklingsmiljö där kompilering och uppladdning till styrenheten sker. Denna modell passade oss bra då den är enkel att använda och WiFi-egenskapen gör det lättarbetat att sätta upp kommunikation mellan exempelvis en server och en servomotor.



Figur 7: Arduino UNO WiFi Rev 2

2.3.2 Servomotor

Då det hade varit osmidigt att ha en grind med automatisk rörelseförmåga i verklig storlek så har vi istället valt att ha en liten servomotor kopplad till Arduinon. Servomotorer är små i storleken och eftersom de har inbyggda kretsar för att styra deras rörelser kan de kopplas direkt till en Arduino som redan har väletablerade bibliotek för servostyrning. Det finns två typer av servomotorer, konventionella servomotorer som kan röra sig mellan ett min-max intervall, och kontinuerliga rotationsservomotorer som kan rotera fritt och vars hastighet kan kontrolleras. En servomotor har vanligtvis tre sladdar: strömkabel, jordledning och en tredje sladd för kommandopulserna. Dessa kopplas ihop med Arduinoenheten som kodas för att styra servomotorn som man önskar. Under projektets gång har vi testat att använda två olika servomotorer med varierande resultat.



Figur 8: Första servomotorn som vi använde

3 Metod

Till en början kretsade arbetet kring inlärning av de program och programspråk som användes under projektets gång. Därefter påbörjades utvecklingen av plattformen, men då kommunikationen skedde på distans så var det viktigt för oss att vara uppdaterade med varsin kod, därav valet att använda oss av versionshanteringsprogrammet Git.

3.1 Minsta möjliga produkt

I ett mjukvaruprojekt specificeras ofta så kallad minsta möjliga produkt, den engelska termen är MVP (Minimum Viable Product). Med detta menas en mjukvara som har tillräckligt med funktioner för att i ett tidigt utvecklingsstadium locka till sig användare som anammar olika trender före alla andra, så kallade *early adopters*[15]. Minsta möjliga produkt för detta projekt är ett system som består av en mobilapplikation där användaren på egen hand kan skriva in sitt registreringsnummer alternativt kan fotografera sin registreringsskylt, denna information skickas vidare till en server som i sin tur kommunicerar med grinden till parkeringen. Detta kortfattade system anser vi vara minsta möjliga produkt för detta projekt och som uppfyller de grundläggande behoven för ett första utkast av en komplett produkt för återkoppling och framtida produktutveckling.

3.2 Testning

För att säkerhetsställa att bildigenkänningen i OpenALPR fungerar och kan verifiera registreringsskyltar utfördes testning på exempelbilder. Som namnet antyder så är OpenALPR öppen mjukvara och källkoden går att ladda ner och använda sig utav för att integrera funktionaliteten i befintligt program. Men detta är mer komplext och vi har istället valt att använda den förkompilerade exekverbara versionen som går att hämta och installera via projektsidan på GitHub [16]. Programmet går att använda via CLI (Command-Line Interface), det vill säga att man kör programmet via kommandoprompten och anger vilken fil som ska hanteras och olika programspecifika flaggor, exempelvis att det är en EU-registreringsskylt man vill identifiera.

4 Genomförande

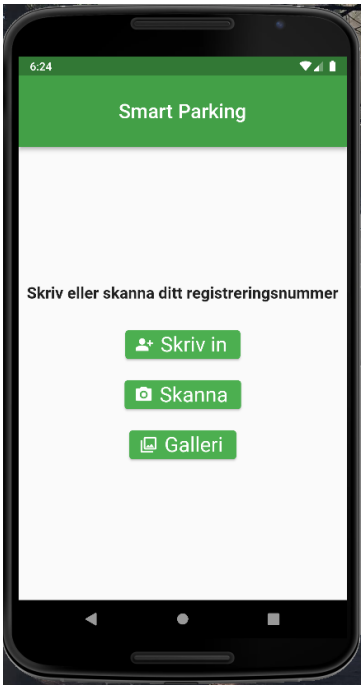
Detta examensarbete har resulterat i en färdig produkt för ett slutet parkeringssystem. Förenklat så kan man fördela systemet i tre delar: mobilapplikationen som agerar användargränssnitt som användaren kan utnyttja, en server i backend som sköter kommunikationen mellan applikationen och själva parkeringen, samt Arduinoenheten som kontrollerar servomotorn.

4.1 Mobilapplikationen SmartParking

Den här delen kommer beskriva vår mobilapplikation och alla dess funktioner.

SmartParking har utvecklats i programmet Android Studio och ska teoretiskt sett funka både på Android och iOS-enheter. Vi skriver teoretiskt då mobilapplikationen endast har testats på två Androidenheter, en Samsung Galaxy S7 Edge och en Samsung Galaxy S21. Det var dessa mobiltelefoner som vi hade tillgång till samt emulatorn i Android Studio för att testa mobilapplikationen.

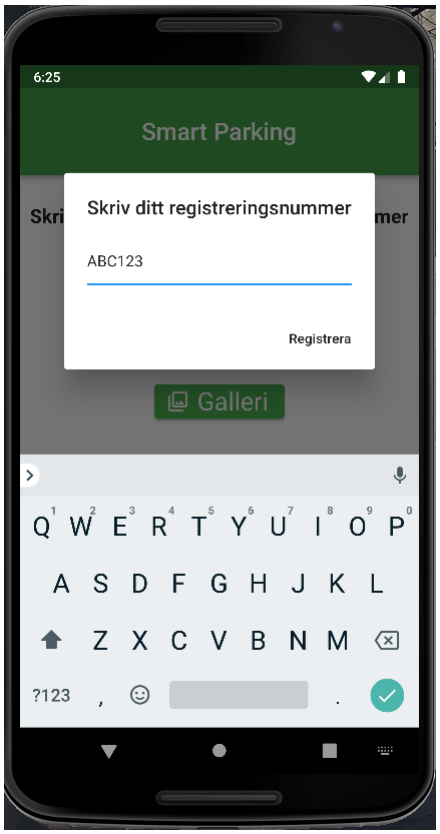
SmartParking applikationen består av en huvudsida och när applikationen startas hamnar användaren direkt på denna sida. Här finns tre alternativ, i form av tre knappar, för att registrera sitt registreringsnummer till tjänsten.



Figur 9: SmartParkings huvudsida

Längst upp i applikationen finns en *app bar*, även kallad *action bar*, som är en återkommande komponent hos Androidapplikationer. Här presenteras tjänstens namn ”SmartParking” på en grön bakgrund, som vi har valt att ha som temafärg. Under applikationens *app bar* finns huvuddelen som innehåller texten ”Skriv eller skanna ditt registreringsnummer” och tre knappar. Dessa knappar är följande:

- Skriv in: Användaren får upp en pop up ruta med beskrivningen ”Skriv ditt registreringsnummer”, tangentbordet startas automatiskt och ska skriva in sitt registreringsnummer. Slutligen trycker man på knappen ”Registrera” för att skicka registreringsnumret till tjänstens server som behandlar datan. Som återkopplande information visas det skickade registreringsnumret längst ner i applikationen.

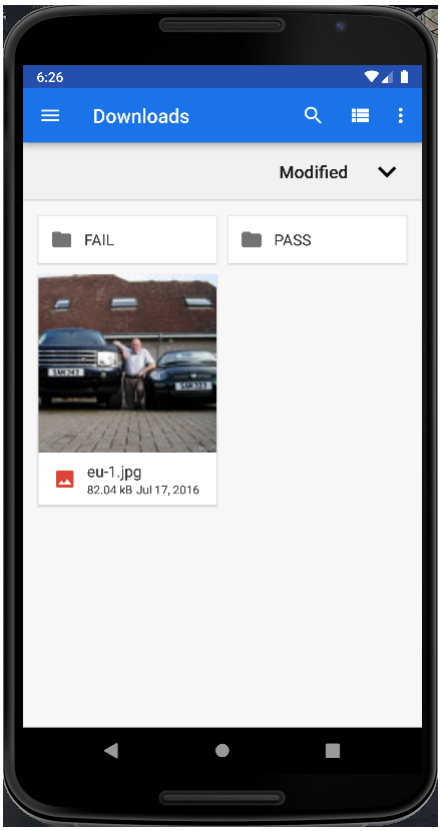


Figur 10: Knappen ”Skriv in”

- Skanna: Mobilens kameraapplikation startas och användaren kan fotografera sin registreringsskylt. När man tagit bilden så måste man acceptera den, därefter skickas den till servern för bildbehandling. Vid första användning så måste man ge applikationen rättighet att använda mobilens kamera.

Figur 11: Knappen ”Skanna”

- Galleri: Om användaren väljer att trycka på denna knapp så startas mobilens bildgalleri och får möjligheten att välja en bild som redan finns sparad lokalt eller i en molntjänst. När man väljer bilden så skickas den automatiskt till servern för bildigenkänning.



Figur 12: Knappen ”Galleri”

4.2 Server

Servern eller backend av vår applikation är som tidigare nämnt skrivet med NodeJS som plattform och ExpressJS som ramverk ovanpå det, med koppling till MongoDB samt OpenALPR för bildigenkänning. Servern använder en global .env (environment) fil, i vilket man enkelt kan konfigurera om man ska starta servern i produktion eller utvecklingsmiljö, om SSL (Secure Sockets Layer) ska användas och i så fall var de certifikatfilerna finns på servern och vilken port samt adress som databasen ska använda sig utav. Beroende på vad för konfiguration man gjort så kommer servern starta upp på HTTP eller HTTPS och därefter ligga och lyssna efter kommunikation.

```
if(process.env.USE_SSL) {
  var fs = require("fs");
  var privateKey = fs.readFileSync(process.env.SSL_PRIVATE_KEY, "utf8");
  var certificate = fs.readFileSync(process.env.SSL_CERTITIFCATE, "utf8");
  var credentials = {key: privateKey, cert: certificate};
  require("https").createServer(credentials, app).listen(process.env.PORT, () => {
      console.log(`Server running HTTPS on port ${process.env.PORT}`);
  });
} else {
  require("http").createServer(app).listen(process.env.PORT, () => {
    console.log(`Server running HTTP on port ${process.env.PORT}`);
  });
}
```

Figur 13: Initiering av HTTP/HTTPS Server

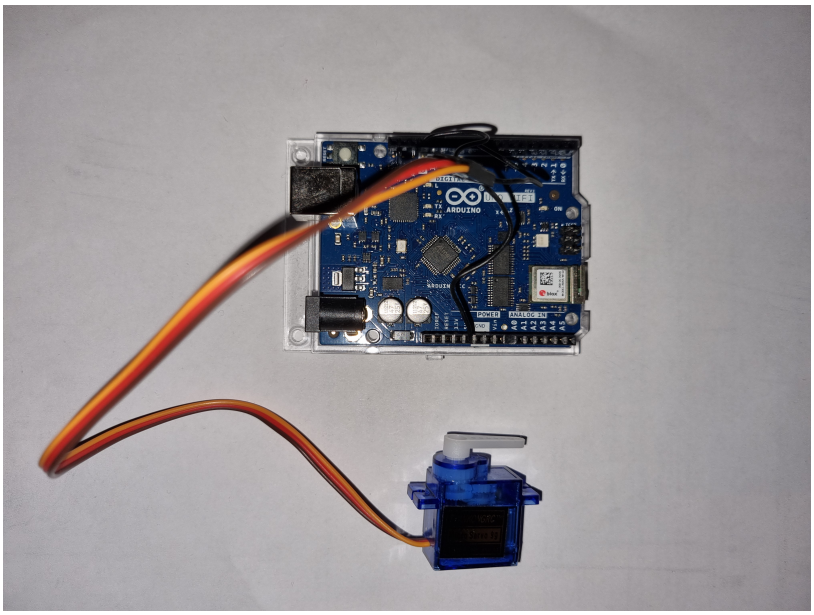
För att kommunicera med servern behöver man, utöver att veta vad för adress, port och protokoll den lyssnar på, även veta vilken rutt man ska skicka sina meddelanden till. I vårt fall har vi satt upp så att den lyssnar på `/parking`, det vill säga exempelvis `https://www.example.com/parking` om man startat servern via HTTPS och har domänen `example.com`. Men det räcker inte enbart att skicka något godtyckligt paket till `/parking`, utan det krävs även att man använder rätt HTTP kommando, vilket här är att skicka meddelandet som ett POST-meddelande. Det POST-meddelandet kan sedan antingen innehålla en bild på en registreringsskylt, alternativt en JSON text innehållandes attributet RegNumber följt av ett registreringsnummer, exempelvis `{ "RegNumber":"ABC123" }`.

Om servern mottar en bild, så skickar den först vidare bilden till OpenALPR som försöker utläsa vad det är för registreringsnummer. Därefter skickas det vidare till parkeringsfunktionen på samma sätt som om ett registreringsnummer hade mottagits som text. Den kollar om det finns något fordon parkerat med det registreringsnumret, om det inte gör det så läggs det till i databasen med nuvarande klockslag som starttid, flaggar att det fordonet är parkerat och returnerar den informationen till avsändaren, samtidigt som det skickar iväg

ett POST-meddelande till Arduinon att den ska öppna grinden. Om ett fordon redan står registrerat som parkerat med det registreringsnumret så flaggas det istället som att det inte längre är parkerat och en sluttid sätts för att därefter skicka den informationen, inklusive starttid, i retur och samtidigt meddelar Arduinon att den ska öppna grinden.

4.3 Arduino UNO

Efter att användaren har skickat sitt registreringsnummer till servern och denna behandlar datan så mottar slutligen Arduino UNO-enheten order om att öppna grinden. Arduinon har varit kopplad med en USB-kabel till en dator för strömåtkomst och koduppladdning, samt varit uppkopplad till ett WiFi-nätverk för att kunna kommunicera med servern.



Figur 14: Arduino UNO ihopkopplad med servomotorn

Nedan följer en detaljerad genomgång av de viktigare delarna av koden till projektets Arduinoenhet vars uppgift är att ha en öppen klient för kommunikation med tjänstens server samt för att styra servomotorn vid förfrågan.

Listing 1: Inkludering av bibliotek och variabeldeklaration

```
#include <SPI.h>
#include <Servo.h>
#include <WiFiNINA.h>

char ssid[]        = "SmartParking";  // The network SSID
char pass[]        = "password"; // The network password

int status         = WL_IDLE_STATUS; // Connection status
Servo servo_9; // Initializes the servomotor

WiFiServer server(80); // Server socket
WiFiClient client = server.available();
```

Programmet börjar med att vi inkluderar nödvändiga bibliotek för seriell kommunikation, kontroll av servomotor och WiFi-uppkoppling. Därefter deklarerar vi variabler för att kunna koppla Arduinoenheten till ett specifikt WiFi-nätverk. Slutligen, initierar vi servomotorn till in-/utgången med nummer 9 på Arduinoenheten samt initierar WiFi-servern till portnummer 80.

Listing 2: Setupfunktionen

```
void setup() {
  // Connects the servomotor to pin 9
  servo_9.attach(9, 500, 2500);

  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  while (!Serial) {
    ; // Wait for serial port to connect.
      // Needed for native USB port only
  }

  enable_WiFi();
  connect_WiFi();

  server.begin();
  printCurrentNet();
  printWifiData();
}
```

Arduinoenhetens Setupfunktion som körs en gång när Arduinoenheten startas. Programmet kopplar servomotorn till Arduinon samt anger dess min- och maxhastighet, i detta fall 500ms samt 2500ms. Därefter skapas seriell kommunikation med datorn på portnummer 9600 och WiFi-uppkoppling aktiveras.

Listing 3: Loopfunktionen

```
void loop() {
  // Check the network connection once every 10 seconds:
  delay(10000);

  client = server.available();
  if(client){
    receiveDataFromServer();
  }
}
```

Arduinoenhetens Loopfunktion som körs var tioende sekund. Så länge det finns en uppkopplad serverklient så körs funktionen receiveDataFromServer().

Listing 4: Kommunikation mellan SmartParkings server och Arduinoenhet

```
void receiveDataFromServer(){
  if (client) {
    Serial.println("New client");
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      while(client.available()) {
        char c = client.read();
        Serial.write(c);
        if (c == '\n' && currentLineIsBlank) {
          while(client.available())
          {
             Serial.write(client.read());
             servomotorGate();
          }
          Serial.println();

          Serial.println("Sending response");
          // Send a standard http response header
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println();
          client.println("<HTML><BODY>Opening the gate</BODY></HTML>");
          client.stop();
        }
        else if (c == '\n') {
          // You're starting a new line
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          // You've gotten a character on the current line
          currentLineIsBlank = false;
        }
      }
    }
    Serial.println("Disconnected");
  }
}
```

Det är i funktionen receiveDataFromServer() som kommunikation mellan tjänstens server och Arduinoenheten sker. När servern skickar ett HTTP POST-meddelande till Arduinoklienten så skrivs detta ut på den seriella monitorn på datorn och funktionen servomotorGate() körs.

Listing 5: Funktion som kontrollerar servomotorn

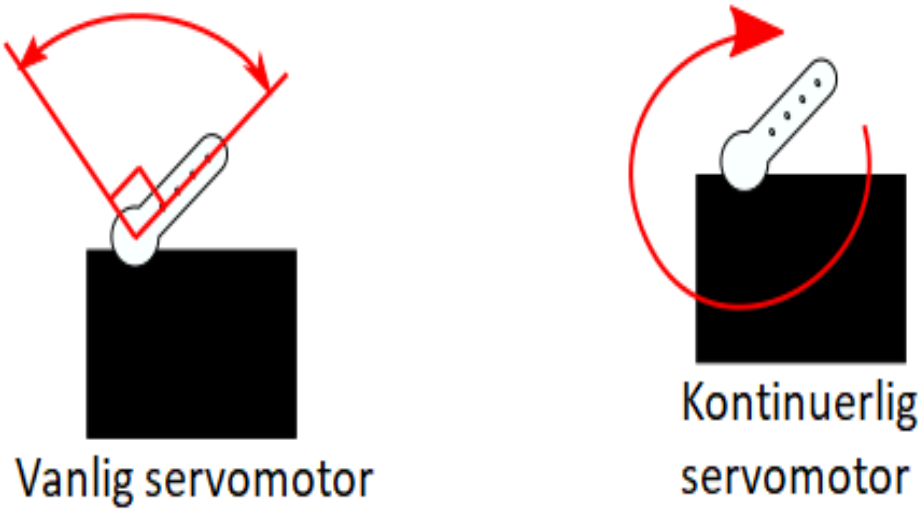
```
// Runs the servomotor that controls the gate
void servomotorGate(){
  int position = 0;
  for (position = 0; position <= 90; position += 1) {
    servo_9.write(position);
    Serial.println("Opening the gate");
  }
  delay(5000); // Wait for 5000 millisecond(s)
  for (position = 90; position >= 0; position -= 1) {
    servo_9.write(position);
    Serial.println("Closing the gate");
  }
}
```

Denna funktion öppnar och stänger servomotorn som ska simulera en riktig parkeringsgrind. Startpositionen sätts till 0 och därefter ska servomotorn röra sig till 90 grader stegvist. Efter 5 sekunder så stängs grinden och servomotorn går tillbaka till startpositionen 0 grader.

4.4 Servomotor

Ovannämnda funktion servomotorGate() sätter nuvarande position till 0, roterar från 0 till 90 grader, väntar 5 sekunder och slutligen roterar från 90 till 0 grader. Dessa 5 sekunder må vara en kort stund men detta sattes endast för testningssyfte. I en komplett produkt så borde man godtyckligt skala upp tiden till minst 30 sekunder men helst även ha med minst en kamera eller sensor som kan avgöra om bilen har passerat grinden för att säkerhetsställa så att ingen bil fastnar.

Som tidigare nämnt finns det flera olika typer av servomotorer men de som vi har undersökt och letat information om är servomotorer med kontinuerlig rotation (kallas ibland för full rotation eller 360° servomotor). Medan en vanlig servomotor endast snurrar över ett smalt område, med exakt kontroll över positionen, har en kontinuerlig rotationsservo en axel som snurrar kontinuerligt, med kontroll över dess hastighet och riktning [17]. När vi i början av projektet fick tillgång till en servomotor så förbryllade det oss varför servomotorn kontinuerligt snurrar fram och tillbaka istället för att köra 90 graders öppning och stängning. Efter mycket informationssökning så insåg vi just att det finns två olika typer av servomotorer och att vi hade fått en så kallad 360° servomotor som man inte kan kontrollera antalet grader på, endast hastighet och riktning. Under slutveckorna så fick vi en ny servomotor men även denna visade sig vara en kontinuerlig servomotor, vilket betyder att vi aldrig lyckades få till den rörelse som vi hade velat.

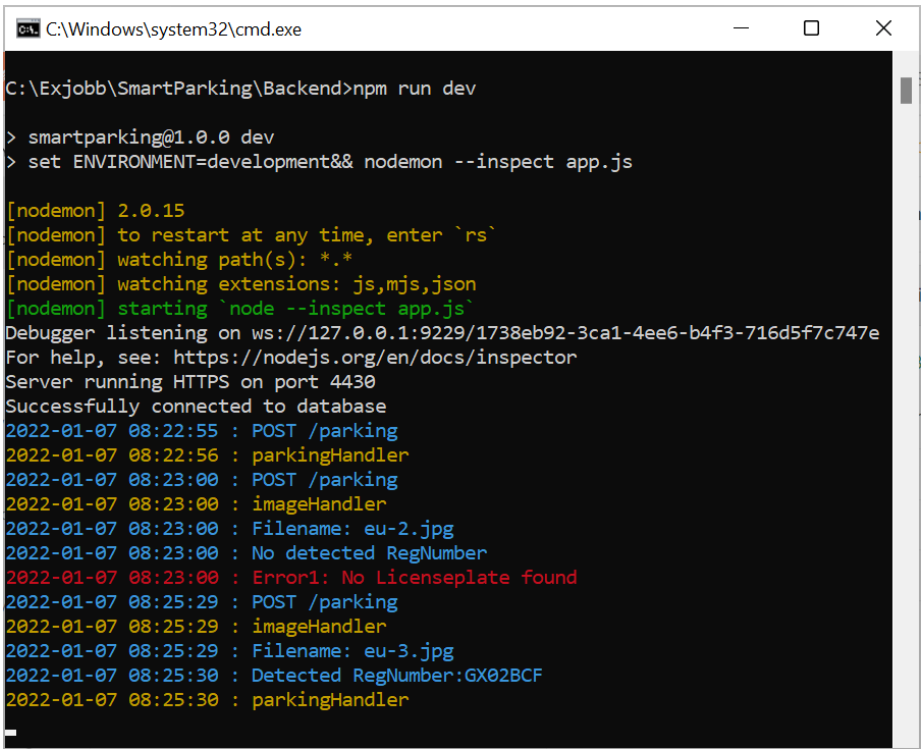


Figur 15: Vanlig och kontinuerlig servomotor

4.5 Ytterligare funktionalitet

Följande är några punkter på ytterligare funktionalitet som vi utvecklat likväl mjukvara vi använt oss utav för att testa och säkerställa att vi hållit en hög kodstandard genom projektet.

Vi har skapat en simpel logging mekanik i backend, som om man startar servern i utvecklingsläge skriver ut meddelanden i lämplig färg beroende på vad servern gör, medan om man startar servern i produktionsläge endast visar kritiska meddelanden. Detta utan att behöva ha skrivit in en kontroll vid varje utskrift till konsollen om man är i ena eller andra läget. Där skulle man även kunnat skriva med så de meddelandena skrevs ut till en fil för eventuell felsökning när tjänsten är igång för användare som eventuellt stöter på problem.



Figur 16: Backend startad med lite exempel på logging

Vi använde oss även utav SonarQube för att analysera backend kodbasen, kanske framförallt för att testa på kodanalysverktyg snarare än att vi hade en stor kodbas med buggar och dåligt skriven kod som behövde gås igenom. Det var initialt lite komplicerat att få det att fungera, men när det väl var igång så kunde vi helt klart se nyttan i att använda liknande verktyg, framförallt i större projekt. Framförallt var dess förklaring på varför en viss bit kod var felaktig eller dålig (så kallad *code smell*) och hur man istället borde göra väldigt användbart för att bli bättre kodare.

Postman är även ett program som är värt att nämna som vi använde för att i ett tidigt skede testa och felsöka serverfunktionaliteten i både backend såväl som Arduinon. Med detta program kunde vi enkelt skicka JSON-meddelanden över POST och även bilder för att användas av i bildigenkänningen på backend.

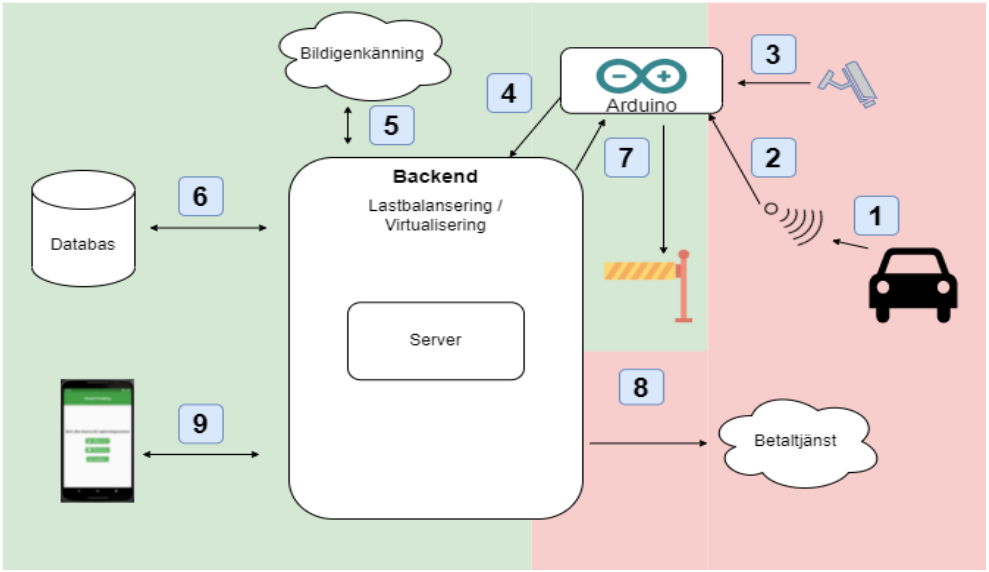
Slutligen vill vi också nämna att vi initialt övervägde att använda oss utav Docker eller Kubernetes för backend, som är virtualiseringsplattformar för att smidigt kunna köra servermjukvaran i en sluten testmiljö vilket också ger oss fördelen att hela vår backend blir portabel. Men dessa två plattformar är till viss del beroende av ens installerade operativsystem, så vi valde istället att skapa en mer fullständig virtuell testmiljö genom det i Windows inbyggda Hyper-V, som förvisso tar mer lagringsplats och inte är lika effektivt vad gäller processorkraft och arbetsminne, men vi båda är mer bekväma med då vi använt det förut. Virtualisering medförde att vi utan att behöva oroa oss för att vår kod och den konfigurering som behövde göras påverkar våra datorer som vi använder till vardags utan alla ändringar är smidigt isolerade i sin egen arbetsmiljö och vi kan även i framtiden enkelt bara starta upp den virtuella maskinen igen och starta upp vår plattform med några enkla klick.

5 Resultat

I detta kapitel nämns vad som har åstadkommits samt vad som ej utvecklades. Vi analyserar även data på hur väl mobilapplikationen tillsammans med OpenALPR lyckades identifiera registreringsnummer i bilder.

5.1 Slutprodukten SmartParking

I slutändan lyckades vi implementera majoriteten av de viktigaste funktionaliteter som vi i det inledande skedet av projektet beslutade att ha med. I nedanstående flödesschema är symbolerna med grön bakgrund de funktionaliter som med framgång implementerades i plattformen SmartParking medan de med röd bakgrund är de funktionaliter som av diverse anledningar inte implementerades. Siffran i flödesschemat representerar siffran i nedanstående listor.



Figur 17: Flödesschema på slutprodukten SmartParking

Lista över implementerade funktionaliteter:

4. Arduinon skickar bilden till backend.
5. Backend skickar bild till bildigenkänningsmjukvaran/molntjänsten, som returnerar registreringsnummer i textformat.
6. Backend kollar i databasen om registreringsnumret redan existerar, och i så fall om det står som parkerat eller inte.
   (a) Om registreringsnumret inte var registrerat eller inte stod som parkerat, så uppdaterar backend databasen att det registreringsnumret nu har påbörjat parkering.

(b) Om registreringsnumret stod som parkerat, så uppdaterar backend databasen att det registreringsnumret nu inte längre är parkerat.

7. Backend meddelar Arduinon att öppna grind för in-/utfart som då öppnar grinden.

9. Man kan också använda mobilapplikationen för att manuellt skicka antingen registreringsnumret som text alternativt bild på registreringsskylten och processen fortsätter likt ovan.

Vi anser att de mest signifikanta funktionaliteterna har implementerats för att uppfylla projektets minsta möjliga produkt (MVP) som specificerades i början av projektet.

Lista över ej implementerade funktionaliteter:

1. Ett fordon närmar sig en parkeringsgrind och upptäcks av en sensor.

2. Sensorn skickar en signal till Arduinon att den upptäckt något.

3. Arduinon tar en bild med kamera.

8. Backend meddelar betaltjänst att användare X ska betala för parkering.

5.2 Testresultat av bildigenkänning

Av de 29 bilder vi valde att testa med lyckades programmet identifiera korrekt registreringsnummer i 9 av fallen, vilket då ger en träffsäkerhet på 31%. Det är möjligt att det skulle gå att få bättre resultat genom att modifiera bilderna att ha högre eller lägre kontrast eller liknande förändringar, men det är utanför syftet med det här examensarbetet och vi nöjde oss med det resultatet. Nedan ser vi två figurer där man ser vilken bild som skickats in och vad för resultat programmet returnerade. I ena fallet lyckades den att korrekt identifiera vilket registreringsnummer det var på bilden och man kan se dess förtroende för andra alternativ som den kanske tror är rätt. I den andra figuren ser vi hur programmet misslyckades att identifiera något registreringsnummer.


```
C:\Windows\system32\cmd.exe
C:\Exjobb\SmartParking\Backend\openalpr_64>alpr.exe -c EU C:\Exjobb\SmartParking\Backend\licenseplates\PASS\s_cmd262.jpg

plate0: 10 results
    - CMD262      confidence: 89.4197
    - MD262       confidence: 78.0417
    - CM0262      confidence: 77.2508
    - GMD262      confidence: 76.9674
    - 6MD262      confidence: 76.615
    - CMO262      confidence: 76.2909
    - CMD26Z      confidence: 73.5697
    - CM262       confidence: 72.3544
    - M0262       confidence: 65.8728
    - MO262       confidence: 64.9129
```

Figur 18: OpenALPR lyckas identifiera rätt registreringsnummer


```
C:\Windows\system32\cmd.exe
C:\Exjobb\SmartParking\Backend\openalpr_64>alpr.exe -c EU C:\Exjobb\SmartParking\Backend\licenseplates\FAIL\s_tbj538.jpg

No license plates found.
```

Figur 19: OpenALPR misslyckas att identifiera ett registreringsnummer

6 Analys och Diskussion

Vi lyckades i stora drag med vad vi hade som mål med vårt examensarbete. En mobilapplikation som kan skicka antingen text eller bild till en server som därefter med hjälp av bildigenkänning och databas hanterar parkering av fordon, samt en Arduino som med en servomotor öppnar en grind på kommando.

Det var dock ett flertal punkter som vi inte lyckades med alternativt inte lyckades så väl med. Exempelvis saknas det någon form av betalfunktion eller rent av någon tydligare form av kommunikation till användaren när dennes fordon är parkerat eller inte. Förvisso är en riktig betalfunktion ett helt eget examensarbete sett till projektstorlek, men man skulle kunna ha någon form av respons till användaren när denne vill avsluta sin parkering och där se hur länge man stått parkerad, vad kostnaden blev och visa en knapp för användaren att betala med. Det är visserligen inga avancerade funktioner att skapa då vi har löst kommunikation mellan de olika enheterna, men ändock något som kanske borde ha varit med.

Vi hade även en hel del problem med Arduinon, delvis tog det en del felsökande och testande innan vi tillslut lyckades få den att ansluta till ett trådlöst nätverk, men framförallt hade vi problem med servomotorn. Vi fick den aldrig att rotera så mycket som vi specificerade i koden att den skulle göra och trots att vi fick låna en andra motor för att felsöka med och vi spenderade många sena kvällar så lyckades vi aldrig lösa problemet. Om det berodde på vår Arduino eller om det var på grund utav servomotorerna lyckades vi aldrig klura ut, men med tanke på att vi testade med två motorer så var det troligtvis något litet som vi hade missat i koden som gjorde att den inte agerade korrekt.

Därutöver är användningen av OpenALPR inte det mest optimala, delvis då den bara lyckades identifiera rätt registreringsskylt i en tredjedel av fallen, men också att det inte är så lärorikt att använda ett befintligt program till vilket vi bara skickar en bild och förhoppningsvis får korrekt registreringsnummer tillbaka i retur. Av ekonomiska skäl så antar vi att det inte finns en väl fungerande applikation som är gratis som har den funktionen och kan utföra det med hög träffsäkerhet, utan det är dyra licenser alternativt molntjänster som tar betalt per identifiering man får använda sig utav. Alternativt att man använder grunderna i exempelvis OpenCV och Tesseract såsom OpenALPR har gjort och på så vis skapa en egen bildigenkänning som presterar bättre, kanske med någon form av maskininlärning. Men även det är nog mer än ett examensarbete helt i sig och därför nöjde vi oss med OpenALPR som i varje fall korrekt lyckades identifiera rätt registreringsnummer i en tredjedel av de exempelbilder vi använt oss utav.

6.1 Etik

De etiska och moraliska frågor som kan uppstå kopplade till det här systemet är relativt få, men är ändå värda att lägga en tanke kring. Framförallt användning av kameror och det kopplat med automatik är något man får ha i åtanke om man skulle ta vårt system, eller en vidareutvecklad variant utav det, i drift. Ifall man använt sensor kopplat till Arduinon som i sin tur tar en bild per automatik när den känner av rörelse så behöver man säkerställa att det är på just ett fordon och inte eventuellt en människa. Likväl att kameran enbart tar en bild på registreringsskylten och inte på eventuellt förare och passagerare. Därutöver skulle datan kunna användas för att kartlägga en besökare om man exempelvis äger flera parkeringar i samma stad eller rent av i flera städer. Detta är dock inget man i dagsläget behöver oroa sig fullt så mycket över då vi enbart stödjer att användaren manuellt fyller i registreringsskylt alternativt tar en bild och bifogar med sin mobil och att vi inte heller lagrar någon historisk data.

6.2 Miljö

Forskning visar att bilförare spenderar onödig tid vid letande av parkeringsplats [18]. Detta resulterar i att den moderna människan förbrukar allt mer tid i bilen, tid som man istället skulle kunna lägga på andra aktiviteter. Genom att förenkla och effektivisera själva parkeringsmomentet för en bilanvändare så sparar man både tid (att leta efter en ledig parkeringsplats) och resurser (bränsleförbrukning vid parkering). Plattformen SmartParking, som består av en server, mobilapplikation samt en energisnål Arduino, borde allt som allt vara en energismart lösning [19]. Jämfört med den energiåtgång som bilar drar vid letande av parkeringsplats så borde vår plattform, utan några konkreta testningar att styrka detta påstående, vara en miljövänligare smart lösning som är till för att hjälpa bilföraren.

7 Slutsats

Syftet med examensarbetet var att skapa ett system för att förenkla parkering med hjälp av en applikation i användarens mobil eller rent av automatisera det hela genom att nyttja kameror och bildigenkänning. Resultatet som vi kom fram till var att det går utmärkt att använda både mobil och automatisering med relativt små och billiga komponenter. Förvisso kräver det en del teknisk expertis för att få bildigenkänningen att fungera tillräckligt bra eller så får man använda sig av dyra betaltjänster som löser det problemet. Om bildigenkänningen skulle fallera så kvarstår fortfarande möjligheten att manuellt fylla i vad för registreringsnummer bilen har, servern godkänner och grinden öppnas för att parkera fordonet.

Det finns ytterligare funktionalitet som man hade kunnat ha med i detta system. Exempelvis finns det ingen säkerhet i att rätt fordon åker ut i fallet av att man själv manuellt skriver in registreringsnummer i applikationen. Man skulle också kunna använda systemet till att begränsa åtkomst till exempelvis en parkering genom att ha en lista på godkända registreringsnummer som användarens fordon matchas mot. Annan funktionalitet skulle vara att ha en kamera kopplad till Arduinon som automatiskt läser av fordon vid in- och utpassage, likväl använda sensorer som känner av när ett fordon passerat grinden och den med säkerhet kan stängas igen.

Referenser

[1] MongoDB (2022)
What is a Document Database?, https://www.mongodb.com/document-databases

[2] MongoDB Manual (2022)
BSON Types, https://docs.mongodb.com/manual/reference/bson-types/

[3] MongoDB (2022)
What is the MEAN Stack?, https://www.mongodb.com/mean-stack

[4] NodeJS About (2022)
About NodeJS, https://nodejs.org/en/about/

[5] W3 Techs (2022)
Historical quarterly trends in the usage statistics of web servers, https://w3techs.com/technologies/history_overview/web_server/ms/q

[6] Hackernoon (2022)
The Advantages of Using Node.js: Caching, Scalability, and a Rich Ecosystem, https://hackernoon.com/the-advantages-of-using-nodejs-caching-scalability-and-a-rich-ecosystem-q23t33c1

[7] RestfulAPI (2022)
What is REST, https://restfulapi.net/

[8] RestfulAPI (2022)
HTTP Methods, https://restfulapi.net/

[9] CodeAcademy (2022)
What is CRUD?, https://www.codecademy.com/article/what-is-crud

[10] SonarQube (2022)
About SonarQube, https://www.sonarqube.org/about/

[11] OpenCV (2022)
20 years of innovations, https://opencv.org/anniversary/20/

[12] DBpedia (2022)
About Tesseract, https://dbpedia.org/page/Tesseract_(software)

[13] git-scm (2022)
About, https://git-scm.com/about

[14] GitHub (2022)
Features, https://github.com/features

[15] ProductPlan (2022)
Minimum Viable Product, https://www.productplan.com/glossary/minimum-viable-product

[16] OpenALPR Git (2022)
OpenALPR GitHub Project, https://github.com/openalpr/openalpr

[17] SparkFun (2022)
Continuous Rotation Servo Trigger Hookup Guide, https://learn.sparkfun.com/tutorials/continuous-rotation-servo-trigger-hookup-guide/all

[18] Nolimo entreprenad (2022)
Nya parkeringslösningar – bra för miljö och ekonomi, https://https://www.xn–byggfretagstockholm-u6b.com/nya-parkeringslosningar-bra-for-miljo-och-ekonomi

[19] Sebastian Sabou (2022)
A survey about power consumption for Arduino, https://oaji.net/articles/2021/3162-1616582077.pdf

Appendix A - SmartParking applikation

Listing 6: main.dart

```
import 'dart:io';
import 'dart:async';
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'package:http_parser/http_parser.dart';
import 'package:image_picker/image_picker.dart';


void main() => runApp(MaterialApp(
  home: Home(),
  // Turns off the 'DEBUG' sign in the upper right corner
  debugShowCheckedModeBanner: false,
));


class Home extends StatefulWidget {
  @override
  State<Home> createState() => _HomeState();
}


class _HomeState extends State<Home> {

  // Declaration of image variable
  File? _image;
  final ImagePicker _picker = ImagePicker();

  // Take a picture with the camera app
  Future cameraImage() async {
    final image = await _picker.pickImage(source: ImageSource.camera);
    setState(() {
      _image = File(image!.path);
      uploadImage();
    });
  }

  // Pick the picture from the gallery
  Future galleryImage() async {
    final image = await _picker.pickImage(source: ImageSource.gallery);
    setState(() {
```

```
    _image = File(image!.path);
    uploadImage();
  });
}


// Upload camera photo to server
Future uploadImage() async {
  final uri = Uri.parse("https://www.tamazin.com:4430/parking");
  var request = http.MultipartRequest('POST', uri);
  var takenPicture = await http.MultipartFile.fromPath
  ("image", _image!.path);
  request.files.add(takenPicture);

  var response = await request.send();
  response.stream.transform(utf8.decoder)
  .listen((value) { print(value); });
  if(response.statusCode == 200){
    print('Image uploaded!');
  } else{
    print('Image not uploaded');
  }
}

// Send Data to the Server
String onValue = '';
postData() async{
  try{
    var response = await http.post(Uri.parse
    ("https://www.tamazin.com:4430/parking"),
        body: {"RegNumber": onValue}
    );
    print(response.body);
  } catch(e){
    print(e);
  }
}

// Pop-up dialog
Future createAlertDialog(BuildContext context) async {

  TextEditingController customController = await TextEditingController();

  return showDialog(context: context, builder: (context){
    return AlertDialog(
      title: Text('Skriv ditt registreringsnummer'),
```

```
        content: TextField(
          autofocus: true,
          textCapitalization: TextCapitalization.characters,
          controller: customController,
        ),
        actions: <Widget>[
          MaterialButton(
            elevation: 5.0,
            child: const Text('Registrera'),
            onPressed: () {
              Navigator.of(context).pop(customController.text.toString());
            },
          )
        ]
      );
    });
}

@override
Widget build(BuildContext context) {
  return GestureDetector(
  // Dismisses the keyboard when pressing on the background
    onTap: () {
      FocusScope.of(context).requestFocus(new FocusNode());
    },
    child: Scaffold(
      appBar: AppBar(
        toolbarHeight: 90,
        title: const Text(
          'SmartParking',
          style: TextStyle(
            fontSize: 27.0,
            //fontFamily: '', // change the font?
          ),
        ),
        centerTitle: true,
        backgroundColor: Colors.green[600],
      ),
      body: Center(
        child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              const Text(
                  'Skriv␣eller␣skanna␣ditt␣registreringsnummer',
                  style: TextStyle(
                    fontSize: 20,
```

```
      fontWeight: FontWeight.bold,
    )
),

const SizedBox(height: 30.0),

// Write button
Builder(builder: (context){
  return ElevatedButton.icon(
      onPressed: () {
        createAlertDialog(context).then((onValue){
          if (onValue == null || onValue.isEmpty) return;
          setState(() => this.onValue = onValue);
          postData();
          SnackBar mySnackbar = SnackBar
          (content: Text('Hello␣$onValue'));
          ScaffoldMessenger.of(context)
          .showSnackBar(mySnackbar);
        });
        print('(dummy␣test)␣skriv␣in');
      },
      icon: const Icon(Icons.person_add_alt_rounded),
      label: const Text('Skriv␣in'), // testa
      style: ButtonStyle(
        backgroundColor:
        MaterialStateProperty.all(Colors.green[500]),
        textStyle:
        MaterialStateProperty.all(
        const TextStyle(fontSize: 26)),
      )
  );
}),

const SizedBox(height: 15.0),

// Camera button
//_image == null ? Text("") : Image.file(_image!),
ElevatedButton.icon(
    onPressed: cameraImage,
    icon: const Icon(Icons.camera_alt_rounded),
    label: const Text('Skanna'),
    style: ButtonStyle(
      backgroundColor: MaterialStateProperty.all(
      Colors.green[500]),
      textStyle: MaterialStateProperty.all(
      const TextStyle(fontSize: 26)),
```

```
                    ),
                ),

                const SizedBox(height: 15.0),

                // Gallery button
                //_image == null ? Text("") : Image.file(_image!),
                ElevatedButton.icon(
                    onPressed: galleryImage,
                    icon: const Icon(Icons.collections_outlined),
                    label: const Text('Galleri'),
                    style: ButtonStyle(
                      backgroundColor: MaterialStateProperty.all(
                      Colors.green[500]),
                      textStyle: MaterialStateProperty.all(
                      const TextStyle(fontSize: 26)),
                    )
                )
              ]
          ),
        ),
      ),
    );
  }
}
```

Appendix B - Server

```
require("dotenv").config(); //Used for loading .env variables
require("./config/database").connect();
const express = require("express");

const app = express();
app.use(express.json({ limit: "50mb" }));

//Enable file upload, required for POST-ing files
const fileUpload = require('express-fileupload');
app.use(fileUpload({
  createParentPath: true
}));
app.use(express.urlencoded({extended: true}));
app.use(express.json());

//Fallback to port80 if no port declared in .env
if (!process.env.PORT) process.env.PORT = 80;

if(process.env.USE_SSL) {
  var fs = require("fs");
  var privateKey = fs.readFileSync(process.env.SSL_PRIVATE_KEY, "utf8");
  var certificate = fs.readFileSync(process.env.SSL_CERTITIFCATE, "utf8");
  var credentials = {key: privateKey, cert: certificate};
  require("https").createServer(credentials, app).listen(process.env.PORT, () => {
    console.log(`Server running HTTPS on port ${process.env.PORT}`);
  });
} else {
  require("http").createServer(app).listen(process.env.PORT, () => {
    console.log(`Server running HTTP on port ${process.env.PORT}`);
  });
}

app.use('/parking', require("./routes/parking"));

app.use("*", (req, res) => {
  return res.status(404).json({
    success: "false",
    message: "Page not found",
    error: {
      statusCode: 404,
      message: "You reached a route that is not defined on this server",
    },
  });
});

module.exports = app;
```

Listing 7: app.js

```
var express = require('express');
var request = require('request');
var router = express.Router();
const ParkingDB = require("../model/parking");
const util = require('util');
const exec = util.promisify(require('child_process').exec);
var logger = require("../common/logger").Logger;
const path = require('path');
const fs = require('fs');

router.post("/", async (req, res) => {
    logger.debug("POST /parking")

    try {
        if(req.files) {
            if (req.files.file || req.files.image) {
                return res.status(200).json(await imageHandler(req.
                    files));
            } else {
                throw new Error("Invalid file/image attachment");
            }
        } else if(Object.keys(req.body).length > 0) {
            return res.status(200).json(await parkingHandler(req.
                body["RegNumber"].trim()));
        } else {
            throw new Error("No valid file AND empty body!");
        }
    } catch (error) {
        logger.error("Error1: " + String(error).split(":").at(-1).
            trim());
        return res.status(500).json({
            "success"   : "false",
            "message"   : "Error",
            "error"     : {
                "statusCode"  : 500,
                "message"     : String(error).split(":").at(-1).
                    trim(),
            },
        });
    }
});

router.use("*", (req, res) => {
    logger.debug("* /parking")
    return res.status(404).json({
            "success": "false",
            "message": "Page not found",
            "error": {
                "statusCode": 404,
                "message": "You reached a route that is not defined
                     on this server",
        },
    });
});

async function parkingHandler(RegNumber){
    logger.info("parkingHandler");
```

```
53      try {
54          let isParked = false;
55          let startTime = null;
56          let endTime = null;
57
58          //Check DB if RegNumber/vehicle is already registered
59          const existingVehicle = await ParkingDB.findOne({ "
                RegNumber" : RegNumber }).exec();
60
61          if (existingVehicle)
62          {
63              //Check if the vehicle is parked
64              if (existingVehicle.isParked) {
65                  //Vehicle is parked, so user wants to leave parking
66                  startTime = existingVehicle.startTime.
                        toLocaleString(process.env.LOCALE);
67                  endTime = new Date().toLocaleString(process.env.
                        LOCALE);
68
69                  //Update db entry, end parking
70                  await ParkingDB.findByIdAndUpdate(existingVehicle.
                        _id, {
71                      "isParked"  : isParked,
72                      "endTime"   : endTime
73                  });
74              } else {
75                  //Vehicle is not parked, so user wants to park
                        vehicle
76                  isParked = true;
77                  startTime = new Date().toLocaleString(process.env.
                        LOCALE);
78
79                  //Update db entry, start parking
80                  await ParkingDB.findByIdAndUpdate(existingVehicle.
                        _id, {
81                      "isParked"  : isParked,
82                      "startTime" : startTime
83                  });
84              }
85          } else {
86              //RegNumber is not registered, so user wants to park
                    vehicle
87              isParked = true;
88              startTime = new Date().toLocaleString(process.env.
                    LOCALE)
89              await ParkingDB.create({
90                      "RegNumber" : RegNumber,
91                      "isParked"  : isParked,
92                      "startTime" : startTime,
93                      "endTime"   : endTime,
94              });
95          }
96
97          var clientServerOptions = {
98              uri: 'http://arduino.tamazin.com/open/',
99              body: JSON.stringify("Open sesame!"),
100             method: 'POST',
```

```
            headers: {
                'Content-Type': 'application/json'
            }
        }

        //Message the arduino to open the gate
        request(clientServerOptions, function (error, response) {
            if(error){
                logger.error("Error opening gate: " + error);
            } else {
                logger.info("Opened Gate");
            }
        });

        //Parking/Stop parking was succesful, set flag and return
            data as object
        return {
            RegNumber,
            isParked,
            startTime,
            endTime
        };
    } catch (error) {
        logger.error("Error5: " + error);
        res.status(500).send({error: error});
    }
}

async function imageHandler(incomingFile) {
    logger.info("imageHandler");
    try {
        const file = incomingFile.file || incomingFile.image;
        const fileTmpPath = path.join(__dirname, '..', 'tmp', file.
            name);

        // Store recieved file in temp location
        file.mv(fileTmpPath, (err) => { if (err) { throw new Error(
            "Error while saving file to temp location!") } });

        logger.debug("Filename: " + file.name);
        // Send file to OpenALPR
        const { stdout } = await exec("imageRecognition.bat " +
            fileTmpPath);

        // Delete temp file
        fs.unlinkSync(fileTmpPath);

        let openALPRREsult = JSON.parse(stdout);
        let RegNumber;
        if (openALPRREsult.results.length) {
            RegNumber = openALPRREsult.results[0].plate;
            logger.debug("Detected RegNumber:" + RegNumber);
        } else {
            logger.debug("No detected RegNumber");
            throw new Error("No Licenseplate found");
        }

```

```
        return parkingHandler(RegNumber);
    } catch (error) {
        // Propagate error up the chain
        throw Error(error);
    }
}

module.exports = router;
```

Listing 8: routesparking.js

```
const mongoose = require("mongoose");

const { MONGO_URI } = process.env;

// Connecting to the database
exports.connect = () => {
  mongoose
    .connect(MONGO_URI)
    .then(() => {
      console.log("Successfully connected to database");
    })
    .catch((error) => {
      console.log("database connection failed. exiting now...");
      console.error(error);
      process.exit(1);
    });
};
```

Listing 9: configdatabase.js

```
const mongoose = require("mongoose");

const parkingSchema = new mongoose.Schema({
  RegNumber: { type: String, unique: true },
  isParked: { type: Boolean, default: false },
  startTime: { type: Date, default: null },
  endTime: { type: Date, default: null }
});

module.exports = mongoose.model("parking", parkingSchema);
```

Listing 10: modelparking.js

```
var Logger = (exports.Logger = {});

Logger.info = function(msg) {
    if(process.env.ENVIRONMENT == "development") {
        var message = new Date().toLocaleString(process.env.LOCALE)
            + " : " + msg;
        console.log('\x1b[33m%s\x1b[0m', message);
    }
};

Logger.debug = function(msg) {
    if(process.env.ENVIRONMENT == "development") {
```

```
12          var message = new Date().toLocaleString(process.env.LOCALE)
                + " : " + msg;
13          console.log('\x1b[36m%s\x1b[0m', message);
14      }
15  };
16
17  Logger.error = function(msg) {
18      if(process.env.ENVIRONMENT == "development") {
19          var message = new Date().toLocaleString(process.env.LOCALE)
                + " : " + msg;
20          console.log('\x1b[31m%s\x1b[0m', message);
21      }
22  };
```

Listing 11: commonlogger.js

Appendix C - Arduino UNO

```
1
2  #include <SPI.h>
3  #include <Servo.h>
4  #include <WiFiNINA.h>
5
6
7  char ssid[]        = "SmartParking";  // The network SSID
8  char pass[]        = "password"; // The network password
9
10 int status         = WL_IDLE_STATUS; // The Wifi radio's connection
       status
11 Servo servo_9; // Initializes the servomotor
12
13 WiFiServer server(80); // Server socket
14 WiFiClient client = server.available();
15
16
17 void setup() {
18   // Connects the servomotor to pin 9
19   servo_9.attach(9, 500, 2500);
20
21   // Initialize serial and wait for port to open:
22   Serial.begin(9600);
23   while (!Serial) {
24     ; // Wait for serial port to connect. Needed for native USB
           port only
25   }
26
27   enable_WiFi();
28   connect_WiFi();
29
30   server.begin();
31   printCurrentNet();
32   printWifiData();
33 }
34
35
36 void loop() {
37   // Check the network connection once every 10 seconds:
38   delay(10000);
39
40   client = server.available();
41   if(client){
42     receiveDataFromServer();
43   }
44 }
45
46
47 // Enables the WiFi module on the Arduino board
48 void enable_WiFi(){
49    // Check for the WiFi module:
50   if (WiFi.status() == WL_NO_MODULE) {
51     Serial.println("Communication with WiFi module failed!");
52     // Don't continue
53     while (true);
```

```
  }

  // Check if the latest Firmware version is installed
  String fv = WiFi.firmwareVersion();
  if (fv < WIFI_FIRMWARE_LATEST_VERSION) {
    Serial.println("Please upgrade the firmware");
  }
}


// Connects to the WiFi network
void connect_WiFi(){
  // Attempt to connect to Wifi network:
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to WPA SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network:
    status = WiFi.begin(ssid, pass);
    // Wait 10 seconds for connection:
    delay(10000);
  }

  // Now the arduino is connected, so print out the data:
  Serial.print("You're connected to the network: ");
  Serial.println();
}


// Information about the network
void printCurrentNet() {
  // Print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // Print the MAC address of the router you're attached to:
  byte bssid[6];
  WiFi.BSSID(bssid);
  Serial.print("BSSID: ");
  printMacAddress(bssid);

  // Print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("Signal strength (RSSI): ");
  Serial.println(rssi);

  // Print the encryption type:
  byte encryption = WiFi.encryptionType();
  Serial.print("Encryption Type: ");
  Serial.println(encryption, HEX);
  Serial.println();
}


// Information about the IP and MAC address
void printWifiData() {
  Serial.println("Your board's IP and MAC address: ");
  // Print your board's IP address:
```

```
111   IPAddress ip = WiFi.localIP();
112   Serial.print("IP Address: ");
113   Serial.println(ip);
114
115   // Print your MAC address:
116   byte mac[6];
117   WiFi.macAddress(mac);
118   Serial.print("MAC address: ");
119   printMacAddress(mac);
120   Serial.println();
121 }
122
123
124 // Finds the MAC adress for your Arduino board
125 void printMacAddress(byte mac[]) {
126   for (int i = 5; i >= 0; i--) {
127     if (mac[i] < 16) {
128       Serial.print("0");
129     }
130     Serial.print(mac[i], HEX);
131     if (i > 0) {
132       Serial.print(":");
133     }
134   }
135   Serial.println();
136 }
137
138
139 void receiveDataFromServer(){
140   if (client) {                     // If you get a client
141     Serial.println("New client");   // Print a message out the
          serial port
142     // An HTTP request ends with a blank line
143     boolean currentLineIsBlank = true;
144     while (client.connected()) {    // Loop while the client's
          connected
145       while(client.available()) {   // While there's bytes to read
            from the client
146         char c = client.read();     // Read a byte
147         Serial.write(c);            // Print it on the serial
              monitor
148         // If you've gotten to the end of the line (received a
              newline
149         // character) and the line is blank, the http request has
              ended,
150         // so you can send a reply
151         if (c == '\n' && currentLineIsBlank) {
152
153           // Here is where the POST data is.
154           while(client.available())
155           {
156              Serial.write(client.read());
157              servomotorGate();
158           }
159           Serial.println();
160
161           Serial.println("Sending response");
```

```
            // Send a standard http response header
            client.println("HTTP/1.1 200 OK");
            client.println("Content-Type: text/html");
            client.println();
            client.println("<HTML><BODY>Opening the gate</BODY></HTML
                >");
            client.stop();
          }
          else if (c == '\n') {
            // You're starting a new line
            currentLineIsBlank = true;
          }
          else if (c != '\r') {
            // You've gotten a character on the current line
            currentLineIsBlank = false;
          }
        }
      }
      Serial.println("Disconnected");
  }
}


// Runs the servomotor that controls the gate
void servomotorGate(){
  int position = 0;
  for (position = 0; position <= 90; position += 1) {
    servo_9.write(position);
    Serial.println("Opening the gate");
  }
  delay(5000); // Wait for 5000 millisecond(s)
  for (position = 90; position >= 0; position -= 1) {
    servo_9.write(position);
    Serial.println("Closing the gate");
  }
}
```

Listing 12: smart_parking.ino