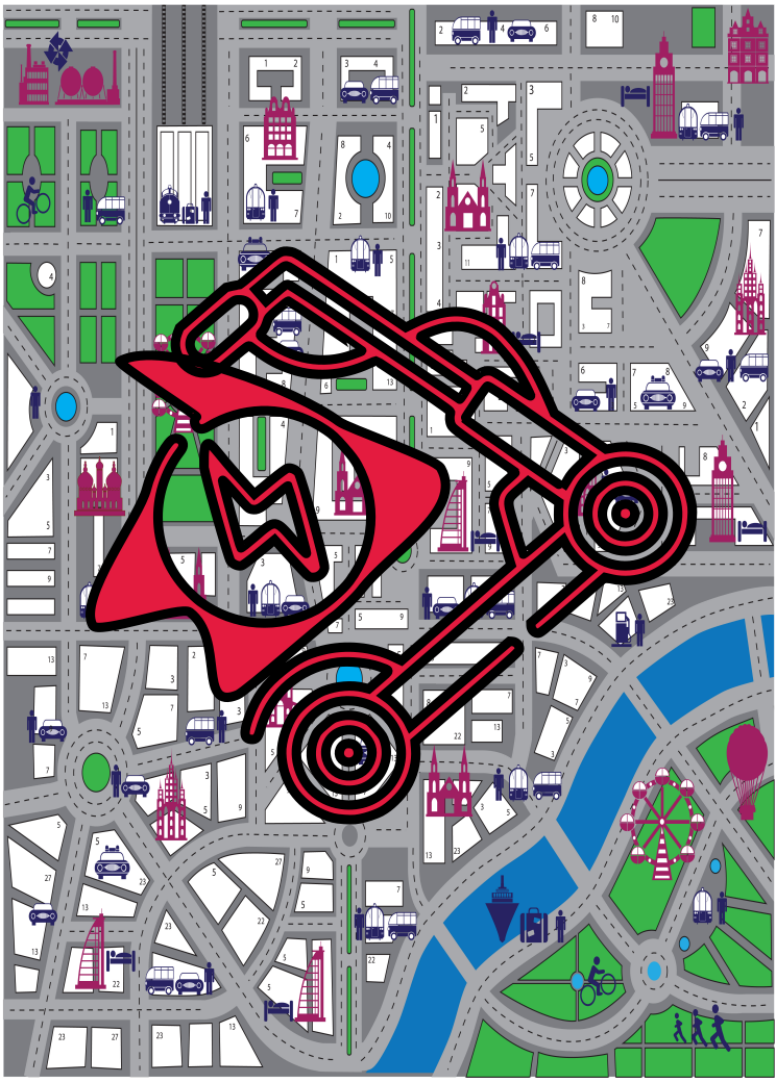
# Elsparkcyklar AB

## System Design Specification



Tomas Zwolinski, Richard Sönnerberg,
Patrik Malvenius, Joel Löfgren
vteam #8
HT 2022
2022-11-25

# Inledning

I detta dokument beskrivs ett system som hanterar uthyrning av elsparkcyklar. Systemet ger kunder möjlighet att skapa användarkonton och hyra cyklar, administratörer ges möjlighet att administrera cyklar, laddstationer, parkeringsplatser, zoner, städer och information om kunder. Systemet innehåller också ett program för cykeln som styr och övervakar denna.

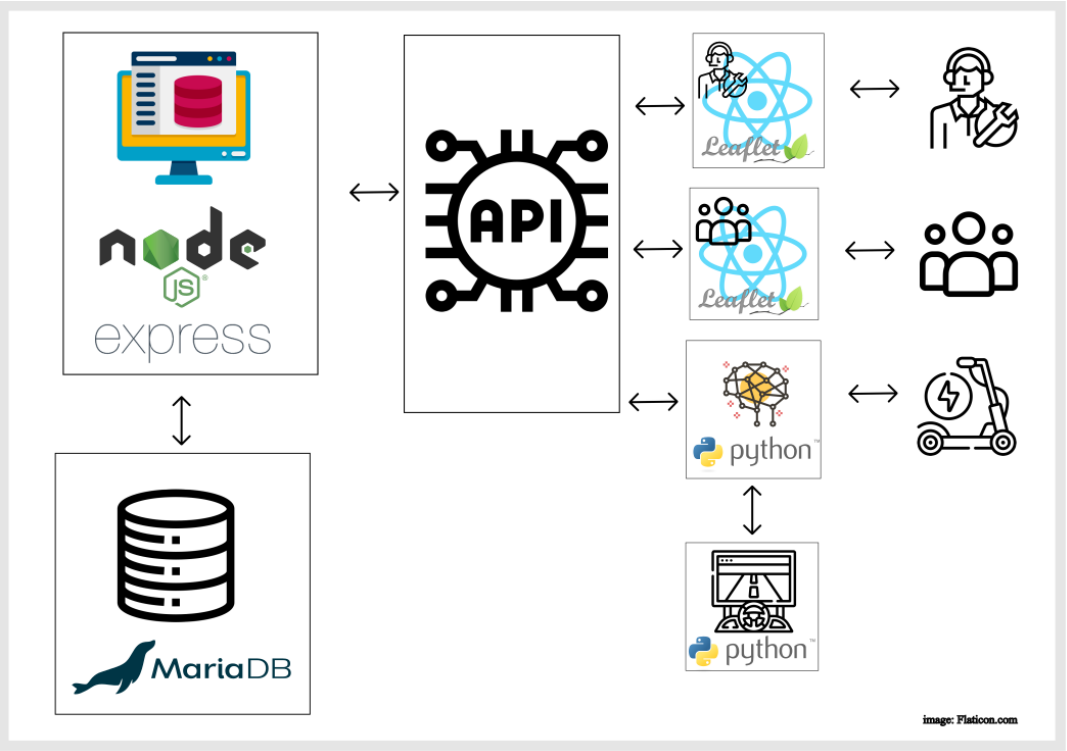
## Bakgrund

Företaget "Svenska Elsparkcyklar AB" har utryckt ett behov av ett system som kan hantera uthyrning av elsparkcyklar i svenska städer. För närvarande är företaget etablerat och har verksamhet i Örebro, Eskilstuna och Vansbro och planerar att expandera till fler med stöd av ett nytt datasystem.

Elsparkcyklar är ett, jämfört med andra fortskaffningsmedel, relativt nytt inslag i vår trafikmiljö. De utgör ett nytt sätt att röra sig och kan med rätt förutsättningar utgöra ett miljövänligt sätt att öka transporteffektiviteten i våra städer. Utmaningarna är dock flera, i en utredning från 2021 påpekar Transportstyrelsen att regelverket kan upplevas som otydligt och att många upplever att de som använder cyklarna inte beter sig korrekt. Transportstyrelsen menar i sin utredning att det framförallt rör sig om att användarna parkerar fel och framför fordonen på ett felaktigt sätt. [1] Sedan 1:a september 2022 får elsparkcyklar inte längre framföras på trottoarer och gångbanor och felparkerade cyklar kan beläggas med avgifter. [2] [3]

Vår förhoppning är att det system som vi presenterar här kommer att kunna lösa en del av dessa utmaningar, inte minst genom att uppmuntra användarna till ett korrekt bruk av cyklarna, att t.ex. parkera cykeln på en parkeringsplats kommer innebära lägre kostnader för användaren, och genom att automatiskt begränsa var och med vilken hastighet cyklarna kan köras.

# Översikt över systemet




*Översikt över systemet*

## Systemets användare

Systemets huvudsakliga användare är kunder (benämns härefter som användare) och administratörer.

Användare har möjlighet att hyra en cykel via en mobilapplikation som också visar status för senaste resan och en historik över gjorda resor. Användare har också tillgång till ett webbgränssnitt där de kan se sina kontodetaljer och en historik över sin användning och betalningar.

Administratörer har möjlighet att via ett webbgränssnitt se status på cyklar och stationer samt få information om kunder. Administratörer kan också lägga till nya cyklar, laddstationer, zoner för parkering, zoner med särskilda regler och städer.

## Systemets delar

Systemet omfattar följande huvudsakliga komponenter:

- Databas med information om cyklar, laddstationer, parkeringszoner, tillåtna zoner att cykla i, användare och administratörer

- Backendmodeller - en komponent bestående av modeller som sköter kopplingen mellan API och databas och gör beräkningar av t.ex. positioner

- API med möjlighet att koppla in anpassade applikationer, grundsystemet levereras med följande applikationer:

    - Administrativt webbgränssnitt där man kan se status för och administrera (ändra, ta bort och lägga till) cyklar, laddstationer, parkeringsplatser, zoner städer och information om kunder

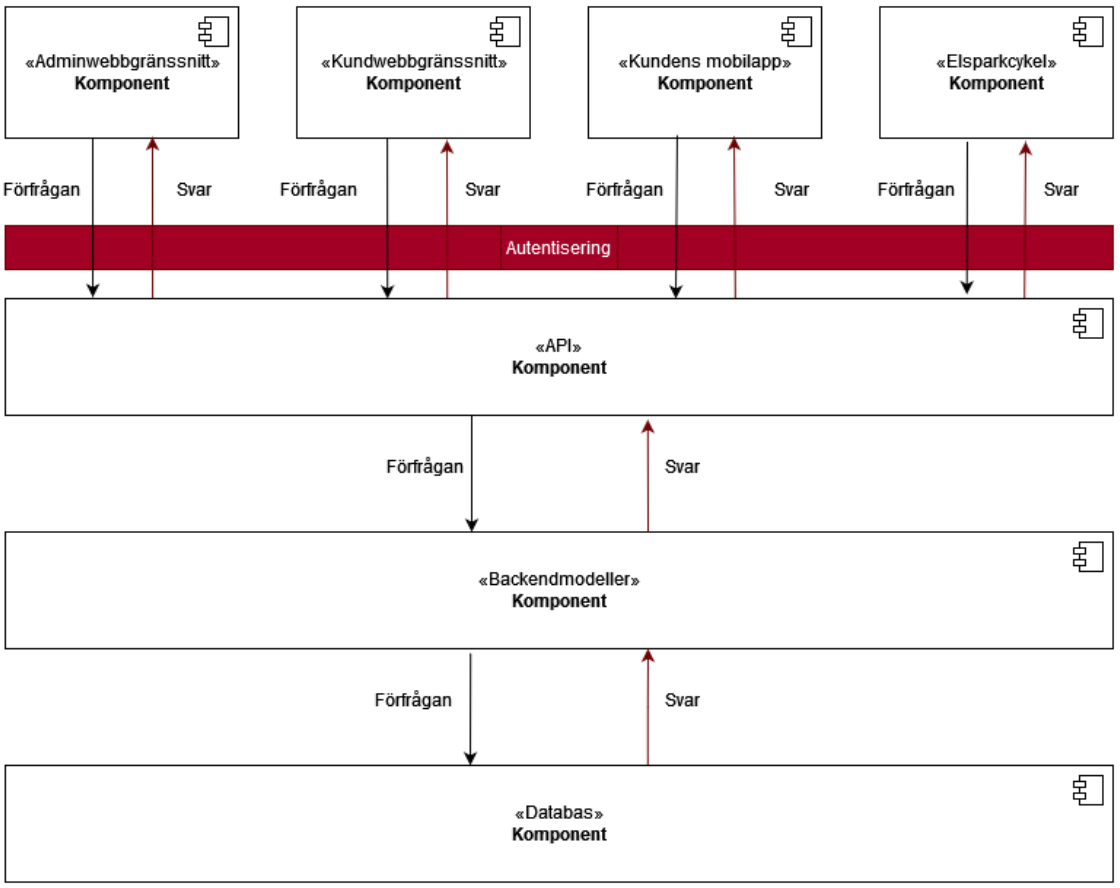
    - Webbgränssnitt för kunden så att denne kan logga in och se sitt konto, historik av utlåning och betalningar

    - Mobilanpassad webbapp för kunden så denne kan se låna/lämna tillbaka cykeln samt se status på senaste resan och historik över gjorda resor

    - Ett cykelprogram som styr och övervakar cykeln (på, av, hastighet, begränsa hastighet, position, behöver service/laddning)

Samtliga applikationer som kopplas mot API:et måste autentisera sin anslutning

Nedanstående diagram visar en översikt över systemets huvudkomponenter samt hur de relaterar till- och kommunicerar med varandra i olika lager. [4]

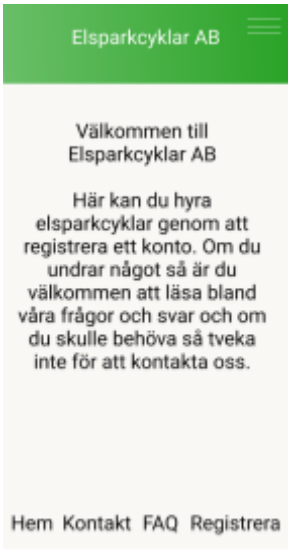



*Översikt över systemets huvudkomponenter*

I följande avsnitt beskriver vi systemets olika delar i detalj.

# Användarens app

I användarens app kan en användare hyra och återlämna elsparkcyklar. Appen är mobilanpassad och byggs med hjälp av JavaScript-biblioteket React. Med React finns möjlighet att bygga upp ett användargränssnitt genom att skriva enskilda komponenter som kopplas samman till ett sammanhängande UI. React kräver inte att man använder någon viss teknologi i resten av systemet och är därför ett bra val för att bygga användargränssnitt i ett modulärt system, där delar ska kunna bytas ut eller läggas till efter behov. Det finns också möjlighet att bygga mobila "native"-applikationer med hjälp av React Native. [5]




*Hemskärm i mobilapp*

Leaflet används för att skapa och hantera kartor och geodata. Leaflet är ett JavaScript bibliotek för att hantera interaktiva kartor. Bibliotekeket är litet och kompakt, jämfört med andra bibliotek för karthantering, t ex OpenLayers, och fungerar väl både för desktop och mobil. Det är därför ett bra val för att hantera kartor i ett system som innehåller applikationer för båda dessa typer av enheter. [6]

Elsparkcyklar som är registrerade i systemet för uthyrning och inte upptagna eller under service kan av en användare väljas för uthyrning via appen. För att kunna hyra en elsparkcykel måste användaren autentisera och identifiera sig, detta görs via OAuth med hjälp av ett GitHub konto. När användaren loggar in i appen visas en kartbild med tillgängliga elsparkcyklar, laddstationer och rekommenderade parkeringsplatser. I vårt program identifieras en elsparkcykel genom att från kartbilden välja elsparcykelns ikon vilket ger användaren möjligheten att välja vald elsparkcykel för uthyrning.

Då en uthyrning påbörjats så låses cykeln upp och användaren kan manövrera elsparkcykeln under hyrtiden.

Användaren kan också, via appen, få information från den hyrda elsparkcykeln i form av:

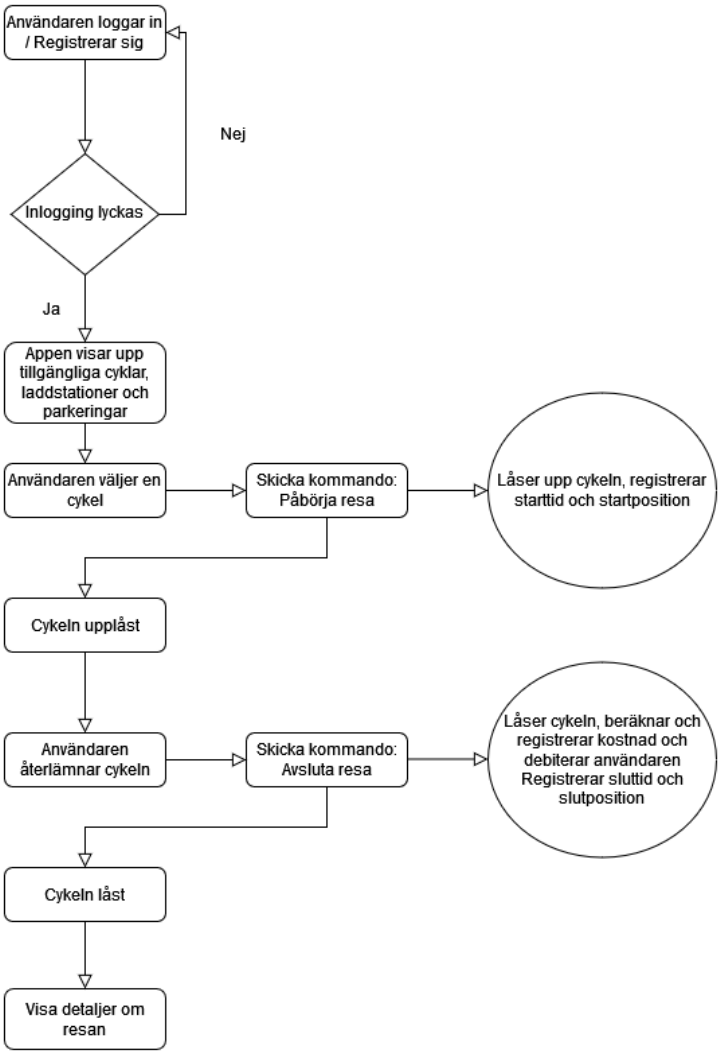
- batterinivå
- behov av service (med anledning av t.ex. punktering, trasiga lampor eller dåliga bromsar)

Då användaren avslutar sin sin resa och återlämnar elsparkcykeln så debiteras användarens konto automatiskt för färden. Färdens kostnad kan variera beroende på:

- hur lång hyrestiden är
- om cykeln flyttats från fri parkering (valfri parkeringsplats utanför rekommenderade parkeringszoner) till en mer önskvärd parkeringsplats
- om cykeln parkeras genom fri parkering

När färden avslutats låses cykeln igen.

Denna bild ger en översikt över flödet för att hyra en elsparkcykel i användarens mobilapp, runda cirklar visar kommandon som utförs i backend.




*Flödet i användarens app vid uthyrning av elsparkcykel.*

# Användarens webbgränssnitt

Denna del av systemet är en desktop-app som, liksom användarens mobilapp, byggs i React. I användarens webbgränssnitt kan en användare logga in för att se och ändra detaljer om sitt konto. Användaren skapar ett konto och loggar in via OAuth med hjälp av sitt Githubkonto.


*Hemskärm i webbläsare på dator*

Det är också möjigt att skapa ett konto på sedvanligt vis genom att uppge:

- förnamn
- efternamn
- adress
- faktureringsadress
- användarnamn
- lösenord

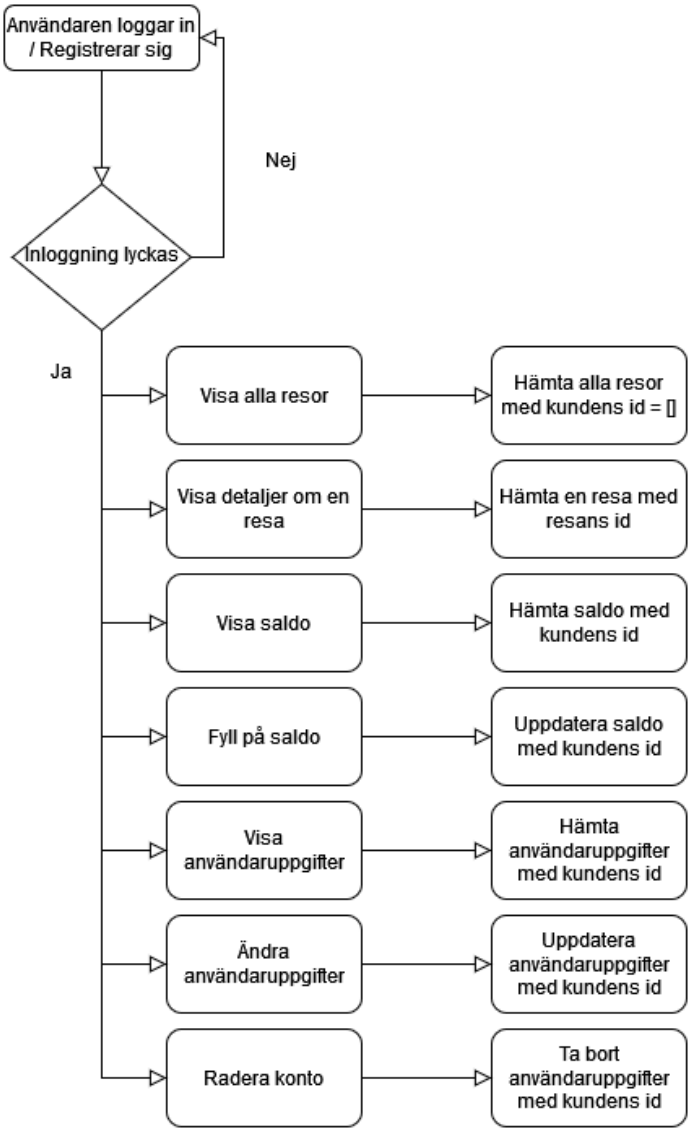
När användaren loggar in har denne möjlighet att ändra detaljer angående kontot eller radera det.

Användaren kan i webbgränssnitttet också se detaljer om sina resor som innefattar:

- ID på resan
- Startdatum
- Starttid
- Slutdatum
- Sluttid
- Startpunkt
- Slutpunkt
- Kostnad för resan

Slutligen kan användaren då denne loggat in på sitt konto också se och fylla på det saldo som används för att bekosta resor med elsparkcykel. Alternativt kan användaren koppla sig till och betala via en betalningstjänst, i så fall dras en avgift varje månad.

Nedanstående diagram visar flödet i användarens webbgränssnitt:




*Överblick över flödet i användarens webbgränssnitt*

# Administratörsgränssnitt

Administratörsgränssnittet byggs i React - ett JavaScript bibliotek för att skapa användargränssnitt. [5] Leaflet används för att skapa och hantera kartor och geodata [6]

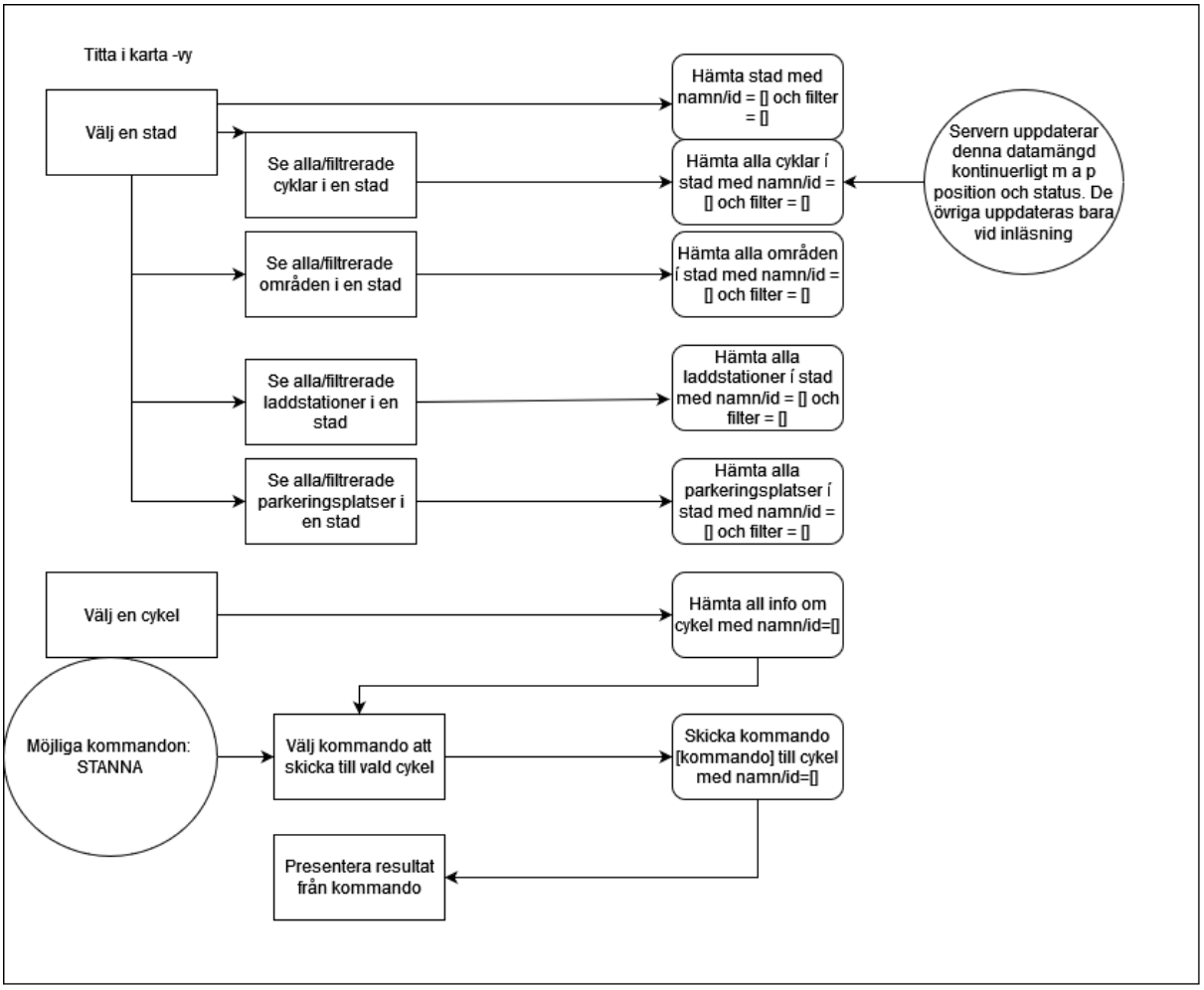
Systemets administratörsgränssnitt används av behöriga användare för att få en översikt över företagets alla resurser:

- Kunder
- Städer man är verksam i
- Cyklar
- Laddstationer
- Parkeringsplatser
- Områden med särskilda bestämmelser

Administratörsgränssnitt innehåller vyer för att inspektera resurser, skicka kommandon till enskilda cyklar, hantera kundinformation, hantera behörigheter, samt uppdatera systemet med ny data, t ex nya städer man etablerar sig i, nya cyklar, nya parkeringsplatser etc.

## Översikt och daglig drift

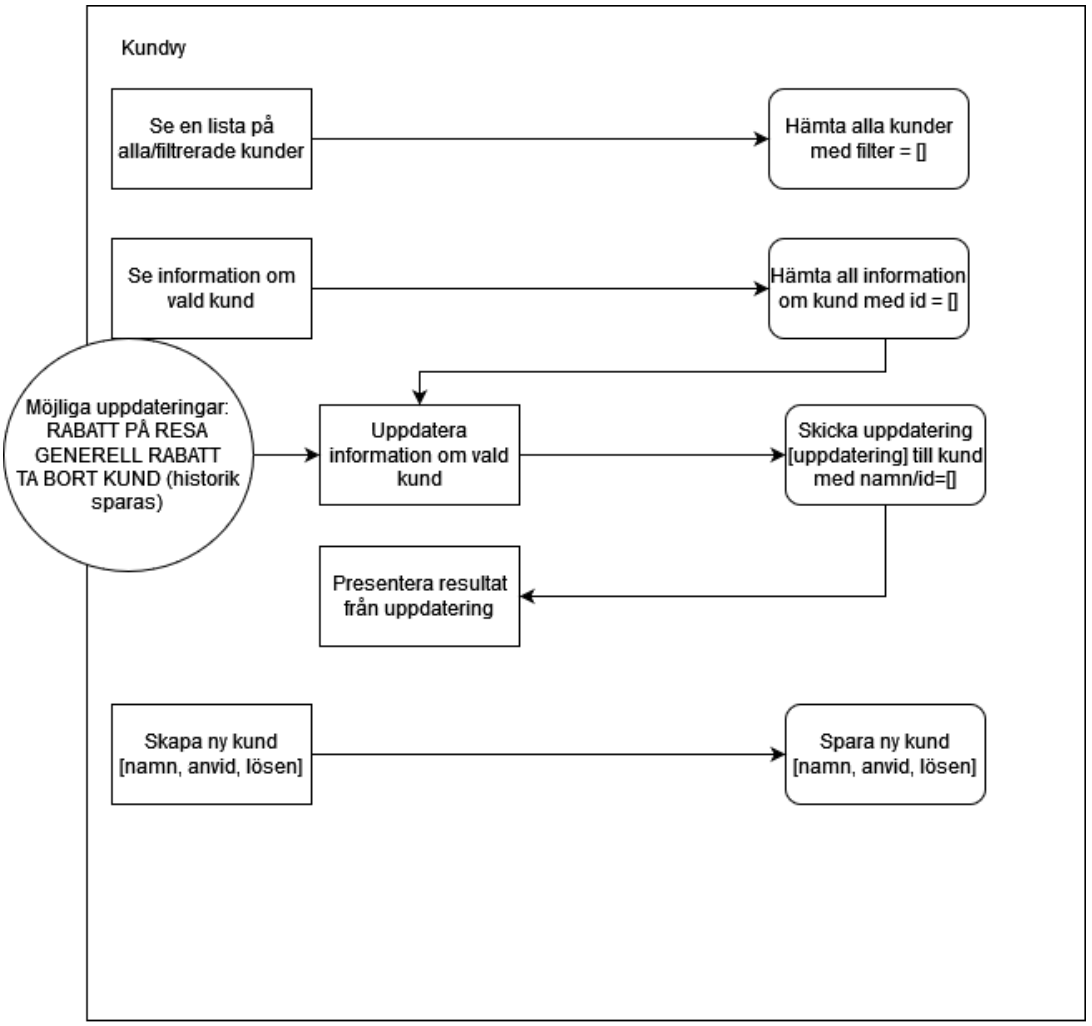
I vyn för översikt och daglig drift presenteras all information kring cyklar, laddstationer, parkeringsplatser och områden med särskilda bestämmelser för varje stad man verkar i. Vyn är kartcentrerad. I kartan kan man se information om aktuell status för alla tillgängliga resurser i vald stad, samt även filtrera kartvyn baserat på resursers typ, identitet eller status. Denna vy används också för att skicka manuella driftkommandon till enskilda cyklar. Det kan t ex vara ett kommando för att stoppa cykeln, om administratören ser behov av det.




*Översiktsvy från systemet*

## Hantering av kunder

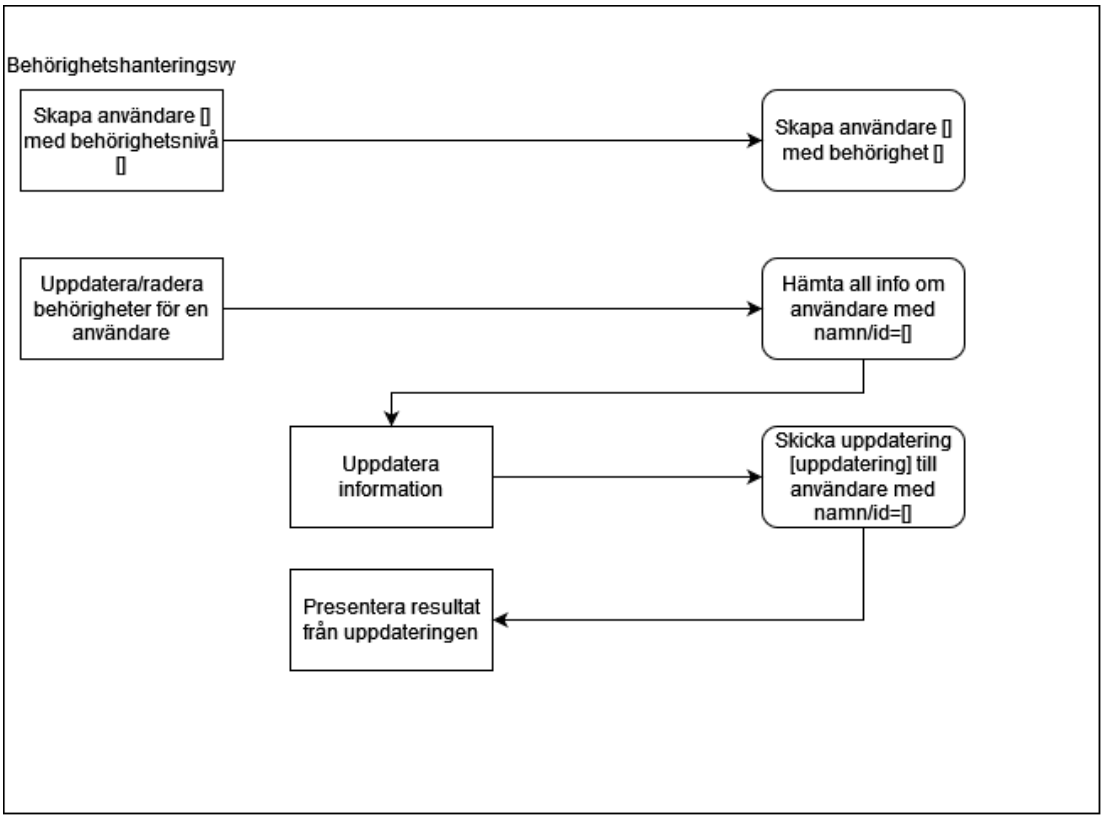
Kundvyn är en klassisk listvy. Här kan man se en lista på alla företagets kunder, som kan filtreras på stad, användarnamn, antal gjorda resor m m. I denna vyn kan administratören också uppdatera information om enskilda kunder, t ex för att ge en kund en generell rabatt eller rabatt för en enskild resa. Administratören kan också skapa upp nya kunder i denna vy, även om detta i normalfallet hanteras av kunden själv.




*Kundvy från systemet*

## Behörighetshantering

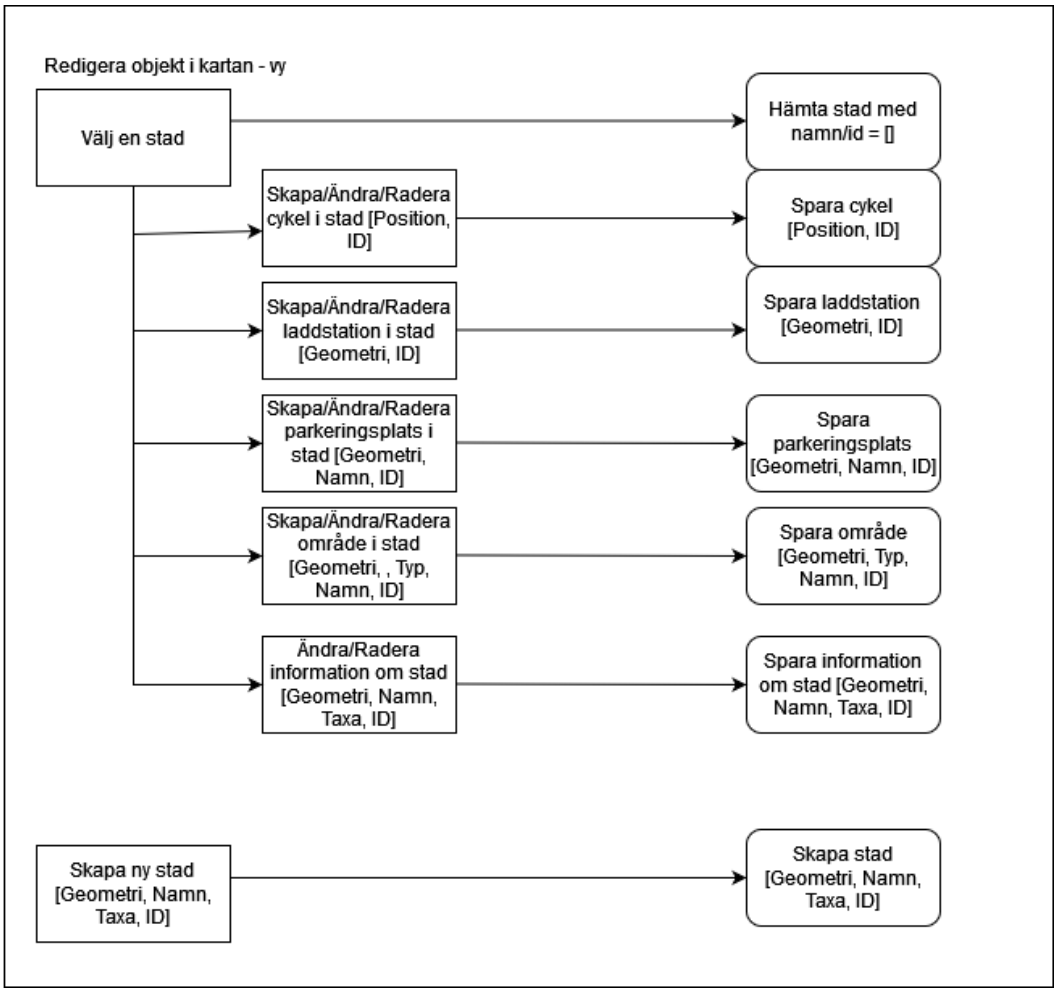
Vyn för behörighetshantering används för att skapa användare och tilldela dessa behörigheter i administratörssystemet. Huvudadministratören anges vid konfiguration av systemet, men alla övriga roller hanteras i detta gränssnitt.




*Behörighetsvy från systemet*

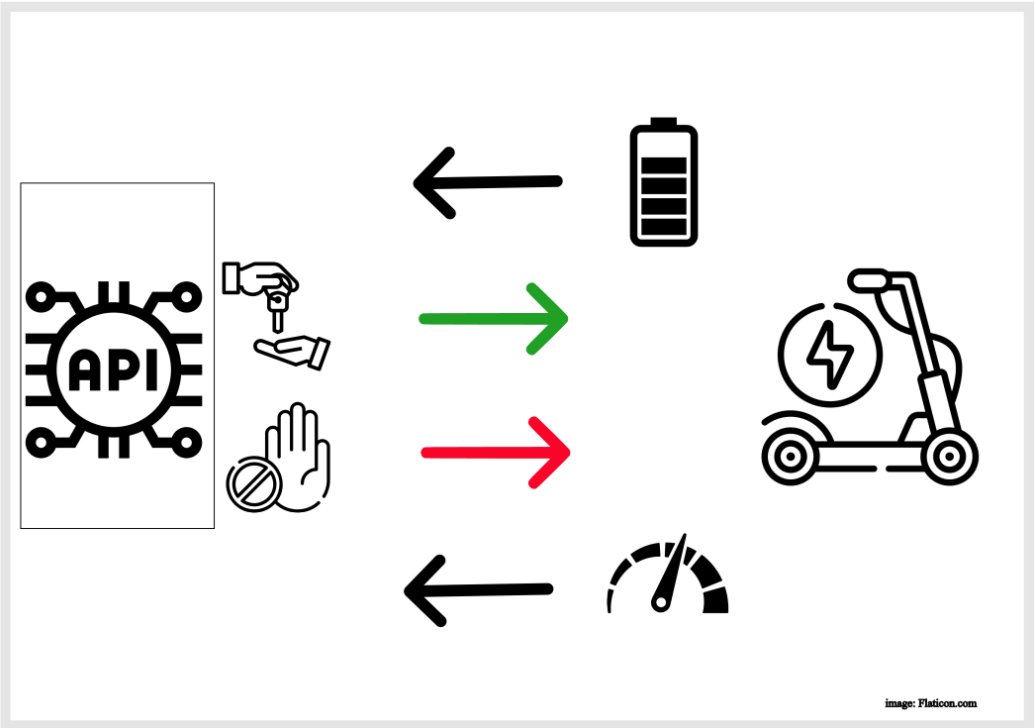
## Geodatahantering

Systemet har kraftfulla och lättanvända funktioner för att hantera nya marknader och nya resurser. I vyn för geodatahantering kan administratören lägga till nya städer för företaget, samt skapa, uppdatera och radera information om enskilda resurser.




*Kartredigeringsvy från systemet*

# Elsparkcykelns mjukvara




*Elsparkcykelns kommunikation med systemet*

En elsparkcykels huvudsakliga uppgift är att hela tiden meddela sin positon och hälsa via API'et. Elsparkcykelns mjukvara byggs i Python, då det i detta programspråk finns många färdiga bibliotek för att hantera integration med sensorer o dyl.

Elsparkcykelns har sensorer som samlar in information som rör dess egen position samt funktionsstatus:

- Batterinivån är låg
- En lampa har gått sönder
- Punktering

Dessa statusindikatorer skickas till backend. Backend matar i sin tur elsparkcykeln med information som rör dess omgivning:

- Uthyrd till en användare
- Användare avslutar hyran
- Begränsa hastighet när den befinner sig i specifika zoner
- Stoppa elsparkcykeln ifall den är utanför tillåtet område.
- Intagen på service
- Service utförd

Det är endast när elsparkcykelns status har blivit ändrad till ”uthyrd” eller på ”service som elsparkcykeln är upplåst och går att köra. Så fort dess status återvänder till ”ledig” så stängs den av och bromsas, och det enda sättat att flytta den är då att fysiskt lyfta upp och bära bort

den. Blir det rörelse på en elsparkcykel som ej är uthyrd skickas då en varning omgående till backend, och sedan med ett tätt intervall tills den återigen står stilla. Detta möjliggör att personal kan hitta eventuellt stulna elsparkcykelar.

Tusentals elsparkcyklar finns i systemet. För att minimera belastningen på API och backend så uppdaterar dom sin position med olika intervall beroende på olika faktorer.

- En uthyrd elsparkcykel i rörelse skickar positionsdata ofta
- En ledig och stillastående elsparkcykel skickar positionsdata sällan
- En elsparkcykel på laddning eller service skickar positionsdata sällan

Varje elsparkcykel sparar också en egen historik över alla sina resor.

- Resans användare
- Resans startposition samt klockslag
- Resans slutposition samt klockslag

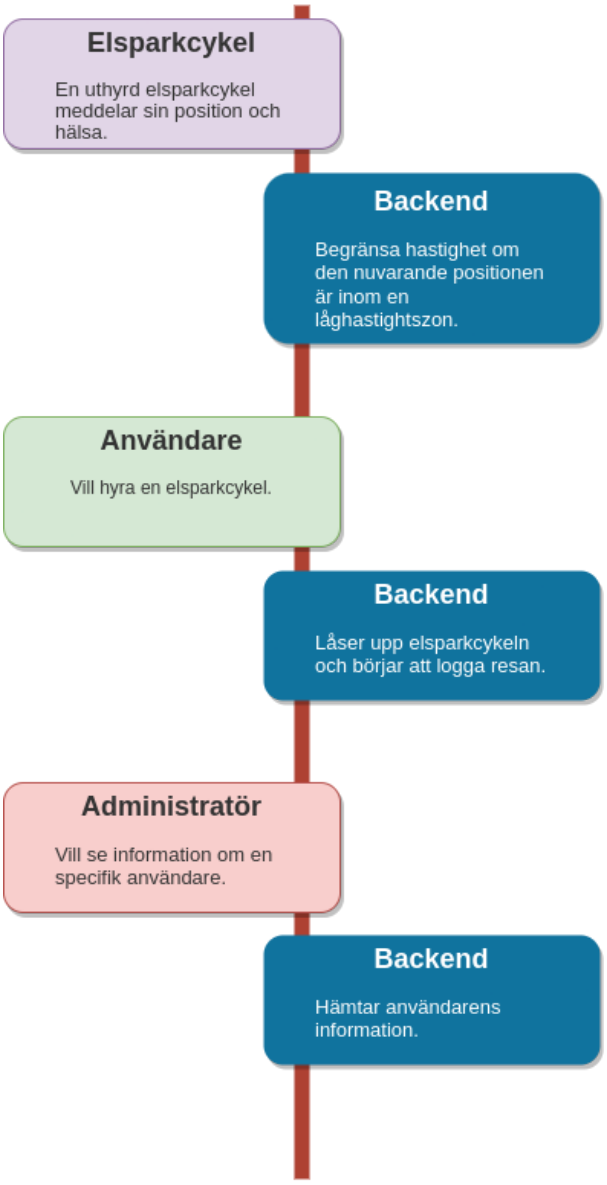
# Backend

De grundläggande kraven för systemets backend är:

- Databasen skall kunna hantera relevant data.
- Systemet skall erbjuda ett väldokumenterat REST API som tredjepartsleverantörer kan använda för att bygga extra tjänster och applikationer.
- REST API:et skall kunna hantera flera olika versioner, t ex genom att använda v1/ som en del i URL:n.
- REST API:et skall hantera autentisering så man kan kontrollera/begränsa belastningen som varje applikation ger.

Med underlag av data från tester med MongoDB och MariaDB, där respektive databas belastades med en stor mängd anrop under kort tid, valdes MariaDB som databas för att hantera all data. [7] Samma tester visade också på stora skillnader i prestanda mellan en FastAPI (Python) och Express (node) server, där Express visade sig vara det bättre valet. Städernas geodata fås från Open Street Map. Det är ett fritt bibliotek som erbjuder den data som behövs. [8]

Kommunikation mellan klienter och backend sker via API:t och där klinten enbart känner till sin egen data och backend har data över alla delar i systemet. Några exempel på kommunikationen är:




*Några övergripande exempel på kommunikationen*

## Backendmodeller

I backend ingår ett lager med modeller som sköter kommunikationen mellan API och databas, alltså skickar in SQL-frågor till databasen och tar emot svar. I dessa modeller genomförs också olika beräkningar. Det kan tex handla om att beräkna kostnaden för en resa, eller beräkna om en cykel befinner sig inom en viss zon. För att göra de nödvändiga geografiska beräkningarna använder vi oss av geodatabibliotek som underlättar detta. Efter att ha undersökt olika bibliotek både för Python och JS kom vi fram till att JS biblioteken tycks vara mer lättanvända. [9] Vi kommer därför använda oss av TurfJS, ett bibliotek som som har flera funktioner som kommer att underlätta för oss att beräkna cyklarnas positioner och göra backendmodellerna effektivare. [9] [10]

## Databas

Information systemets olika entiteter samlas i en databas. En entitet kan t.ex. vara "användare", "stad" eller "administratör". I detta avsnitt beskrivs vilka entiteter som finns i databasen, vilka egenskaper de har och hur entiteterna relaterar till varandra.

Databasen som används är MariaDB. Det är en open source relations-databas. Den är baserad på MySQL databasen och använder traditionell SQL syntax.

De fyra entiteter som utgör grunden för databasen är:


*De fyra huvudsakliga entiteterna*

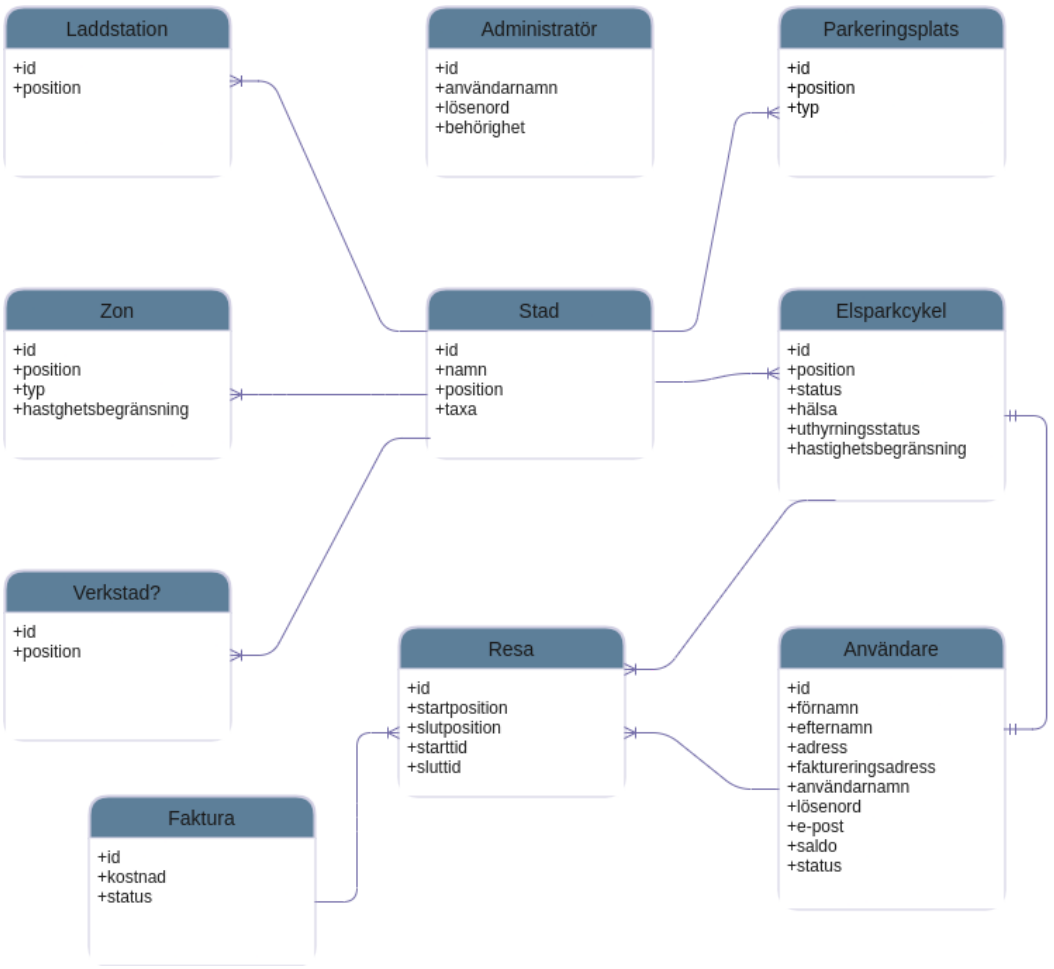
Tabeller vars data är en produkt av olika händelser uppdateras med data från någon av de fyra entiteterna. tex när en användare hyr en elsparkcykel så startas en resa.

- Vilken användare hyrde
- Vilken elsparkcykel som hyrdes
- Elsparkcykelns position när hyran startades
- Elsparkcykelns position när hyran avslutades

Denna resa utgör i sin tur underlaget för en faktura.

- Stadens taxa
- Eventuell rabatt om resan avslutades i en parkeringszon

Nedanstående bild visar samtliga entiteter som förekommer i databasen, deras attribut och inbördes relationer.




*ER-diagram*

## REST-API

Systemets applikationer använder ett API för att kommunicera med systemets backend. API:et i detta system är ett så kallat REST API. Klienterna ansluter till API:et och autentiserar sig och kan sedan skicka en GET, POST, PUT eller DELETE förfrågan för att få, skicka in, uppdatera eller ta bort information i systemet. Varje förfrågan får ett svar i JSON-format som klienten sedan kan använda, t.ex. för att visa ut information till användaren. [11] För att veta vilka routes vi behöver ha i REST-API:et så har följande substantiv identifierats (dessa stämmer överens med entiteterna i databasen):

**Substantiv**

- Användare
- Administratör
- Huvudadministratör
- Elsparkcykel
- Laddstation
- Stad
- Zon
- Verkstad

**Verb**

De identifierade substantiven behöver också utföra handlingar, därav följande verblista:

- En användare ska hyra en elsparkcykel
- En användare ska registrera ett konto
- En användare ska logga in på sitt konto
- En användare ska lämna tillbaka en elsparkcykel
- En administratör ska registrera ett konto
- En administratör ska logga in på sitt konto
- En huvudadministratör ska registera en administratör
- En elsparkcykel ska köra
- En laddstation ska innehålla elsparkcyklar
- En stad ska innehålla elsparkcyklar, zoner etc.
- En verkstad ska kunna utföra service på elsparkcyklar

Listan nedan visar exempel på routes som API:et stödjer. I appendix 1 finns en lista över samtliga routes som API:et stödjer.

| URL | GET | POST | PUT | DELETE |
|---|---|---|---|---|
| /users/ | Visa alla användare | Skapa en ny användare | ❌ | ❌ |
| /users/{id} | Visa en användare med id {id} | ❌ | Modifiera en användare med id {id} | Ta bort en användare med id {id} |
| /users/{id}/position | Visa positionen för en användare med id {id} | ❌ | ❌ | ❌ |
| /users/{city}/ | Visa alla användare som för tillfället hyr en elsparkcykel i stad {city} | ❌ | ❌ | ❌ |
| /users/online/ | Visa alla användare som för tillfället är online | ❌ | ❌ | ❌ |

**Dokumentation**

För att underlätta för tredjepartsleverntörer att bygga externa tjänster och applicationer är REST-API:et väldokumenterat.

**Versioner**

REST-API'et har byggts för att vara framtidssäkert där uppdateringar och tillägg hanteras med versionsnummer som en del i URL:en.

**Autentisering**

Alla applikationer som använder REST-API:et måste autentisera sig med hjälp av JSON Web Tokens för att kontrollera att endast endpoints som rör applikationen finns tillgängliga.

#### Godkänd autentisering

En applikation för administratörer kan se alla användare i systemet.

GET ../v1/users/

```
{
   "data": [
      {
         "id": 1,
         "name": "John Doe",
         ...
      },
      {
         "id": 2,
         "name": "Jane Doe",
         ...
      }
   ]
}
```

#### Misslyckad autentisering

En applikation för användare kan **inte** se alla användare i systemet.

GET ../v1/users/

```
{
   "errors": {
      "status": 401,
      "source": "/users",
      "title": "Authorization failed",
      "detail": "No valid API key provided."
   }
}
```

# Tester

Alla i systemet ingående komponenter kommer under utvecklingen av systemets genomgå unit- och integrationstester

# CI/CD

För att säkerställa att alla tester körs som de ska, och att ingen ny kod skapas som bryter beteendet i befintlig kod, bygger vi ett flöde för CI/CD i GitHub.

# Simulering

Innan systemet färdigställs för slutleverans kommer systemet utsättas för ett virtuellt stresstest. I detta test simulerar vi 5000 cyklar som hyrs ut, framförs, och återlämnas i vardera stad systemet skall användas i. Simuleringsprogrammet kommer att använda sig av förgenererade rutter längs städernas vägnät för att simulera ett realistiskt användarbeteende. Resultatet av denna simulering sammanställs och infogas i leveransrapporten.

# Driftsättning

Systemet kan driftas i valfri miljö då vi redan vid utvecklingen implementerar containerization genom Docker.

# Referenser


[1] Transportstyrelsen. "Utredning behov av förenklade regler för eldrivna enpersonsfordon". Internet: https://www.transportstyrelsen.se/globalassets/global/publikationer-och-rapporter/vag/slutrapport-utredning-regler-eldrivna-enpersonsfordon.pdf [2022-11-18].

[2] M. Eskilsson. "Även för elsparkcyklister är reglerna en bra utveckling". Jönköpingsposten (Sep. 3, 2022), Ledare s. 2.

[3] "Trafikförordning (1998:1276)". Internet: https://rkrattsbaser.gov.se/sfst?bet=1998:1276 [2022-11-18].

[4] M. Richards. "Software architecture patterns". Internet: https://www.oreilly.com/content/software-architecture-patterns/ [2022-11-18].

[5] "React". Internet: https://reactjs.org/ [2022-11-19]

[6] "Leaflet - a JavaScript library for interactive maps". Internet: https://leafletjs.com/ [2022-11-19]

[7] "Requests/sek". Internet: https://github.com/virtuella-team/vteam/tree/tzLocal [2022-11-19]

[8] "Open Street Map" . https://www.openstreetmap.org/#map=5/62.994/17.637 [2022-11-19]

[9] "Hantering av geodata i databas/backend". Internet: https://github.com/virtuella-team/vteam-sds/blob/main/teknisk-analys-geo/teknisk_analys_geo.md [2022-11-22].

[10] "Turf.js". Internet: https://turfjs.org/ [2022-11-19].

[11] [F. Ximenes and F. Juvenal. "How to design a RESTful API architecture from a human-language spec". Internet: https://www.oreilly.com/content/how-to-design-a-restful-api-architecture-from-a-human-language-spec/ [2022-11-19]

# Appendix 1 - lista över routes

Detta appendix innehåller de routes som systemets REST-API stödjer.

### Lista över routes

| URL | GET | POST | PUT | DELETE |
|---|---|---|---|---|
| /users/ | Visa alla användare | Skapa en ny användare | ❌ | ❌ |
| /users/{id} | Visa en användare med id {id} | ❌ | Modifiera en användare med id {id} | Ta bort en användare med id {id} |
| /users/{id}/position | Visa positionen för en användare med id {id} | ❌ | ❌ | ❌ |
| /users/{city}/ | Visa alla användare som för tillfället hyr en elsparkcykel i stad {city} | ❌ | ❌ | ❌ |
| /users/online/ | Visa alla användare som för tillfället är online | ❌ | ❌ | ❌ |
| /users/offline/ | Visa alla användare som för tillfället är offline | ❌ | ❌ | ❌ |
| /administrators/ | Visa alla administratörer | Skapa en ny administratör | ❌ | ❌ |
| /administrators/{id} | Visa en administratör med id {id} | ❌ | Modifiera en administratör med id {id} | Ta bort en administratör med id {id} |
| /mainAdministrators/ | Visa alla huvudadministratörer | Skapa en ny huvudadministratör | ❌ | ❌ |
| /mainAdministrators/{id} | Visa en huvudadministratör med id {id} | ❌ | Modifiera en huvudadministratör med id {id} | Ta bort en huvudadministratör med id {id} |
| /cities/ | Visa alla städer | Skapa en ny stad | ❌ | ❌ |
| /cities/{id} | Visa en stad med id {id} | ❌ | Modifiera en stad | Ta bort en stad med id {id} |

| | | | | |
|---|---|---|---|---|
| | | | med id {id} | |
| /cities/{id}/zones/ | Visa alla zoner i stad med id {id} | ❌ | ❌ | ❌ |
| /cities/{id}/electricScooters/ | Visa alla elsparkcyklar i stad med id {id} | ❌ | ❌ | ❌ |
| /cities/{id}/chargingStations/ | Visa alla laddstationer i stad med id {id} | ❌ | ❌ | ❌ |
| /cities/{id}/workshops/ | Visa alla verkstäder i stad med id {id} | ❌ | ❌ | ❌ |
| /electricScooters/ | Visa alla elsparkcyklar | Skapa en ny elsparkcykel | ❌ | ❌ |
| /electricScooters/rented | Visa alla elsparkcyklar som just nu är uthyrda | ❌ | ❌ | ❌ |
| /electricScooters/stale | Visa alla elsparkcyklar som just nu inte är uthyrda | ❌ | ❌ | ❌ |
| /electricScooters/serviced | Visa alla elsparkcyklar som just nu är under service | ❌ | ❌ | ❌ |
| /electricScooters/{id} | Visa en elsparkcykel med id {id} | ❌ | Modifiera en elsparkcykel med id {id} | Ta bort en elsparkcykel med id {id} |
| /electricScooters/{id}/position | Visa positionen för en elsparkcykel med id {id} | ❌ | ❌ | ❌ |
| /electricScooters/{id}/batteryLevel | Visa batterinivån för en elsparkcykel med id {id} | ❌ | ❌ | ❌ |
| /electricScooters/{id}/stop | ❌ | Stoppa elsparkcykeln | ❌ | ❌ |
| /chargingStations/ | Visa alla laddstationer | Skapa en ny laddstation | ❌ | ❌ |

| | | | | |
|---|---|---|---|---|
| /chargingStations/{id} | Visa en laddstation med id {id} | ❌ | Modifiera en elsparkcykel med id {id} | Ta bort en elsparkcykel med id {id} |
| /zones/ | Visa alla zoner | Skapa en ny zon | ❌ | ❌ |
| /zones/{id} | Visa en zon med id {id} | ❌ | Modifiera en zon med id {id} | Ta bort en zon med id {id} |
| /workshops/ | Visa alla verkstäder | Skapa en ny verkstad | ❌ | ❌ |
| /workshops/{id} | Visa en verkstad med id {id} | ❌ | Modifiera en verkstad med id {id} | Ta bort en verkstad med id {id} |
| /parkingZones/ | Visa en alla parkeringplatser | Skapa en ny parkeringsplats | ❌ | ❌ |
| /parkingZones/{id} | Visa en parkeringplats med id {id} | ❌ | Modifiera en parkeringsplats med id {id} | Ta bort en parkeringsplats med id {id} |
| /trips/ | Visa alla resor | Skapa en ny resa | ❌ | ❌ |
| /trips/{id} | Visa en resa med id {id} | ❌ | Modifiera en resa med id {id} | Ta bort en resa med id {id} |
| /trips/{userId} | Visa alla resor med användar-id {användar-id} | ❌ | ❌ | ❌ |