System Design Specification

2021

—

Webbaserade ramverk och designmönster - Grupp 13
(Benaris Hajduk)
Daniel Lundgren
Hanna Johansson
Jesper Stolt

Innehållsförteckning

Bakgrund

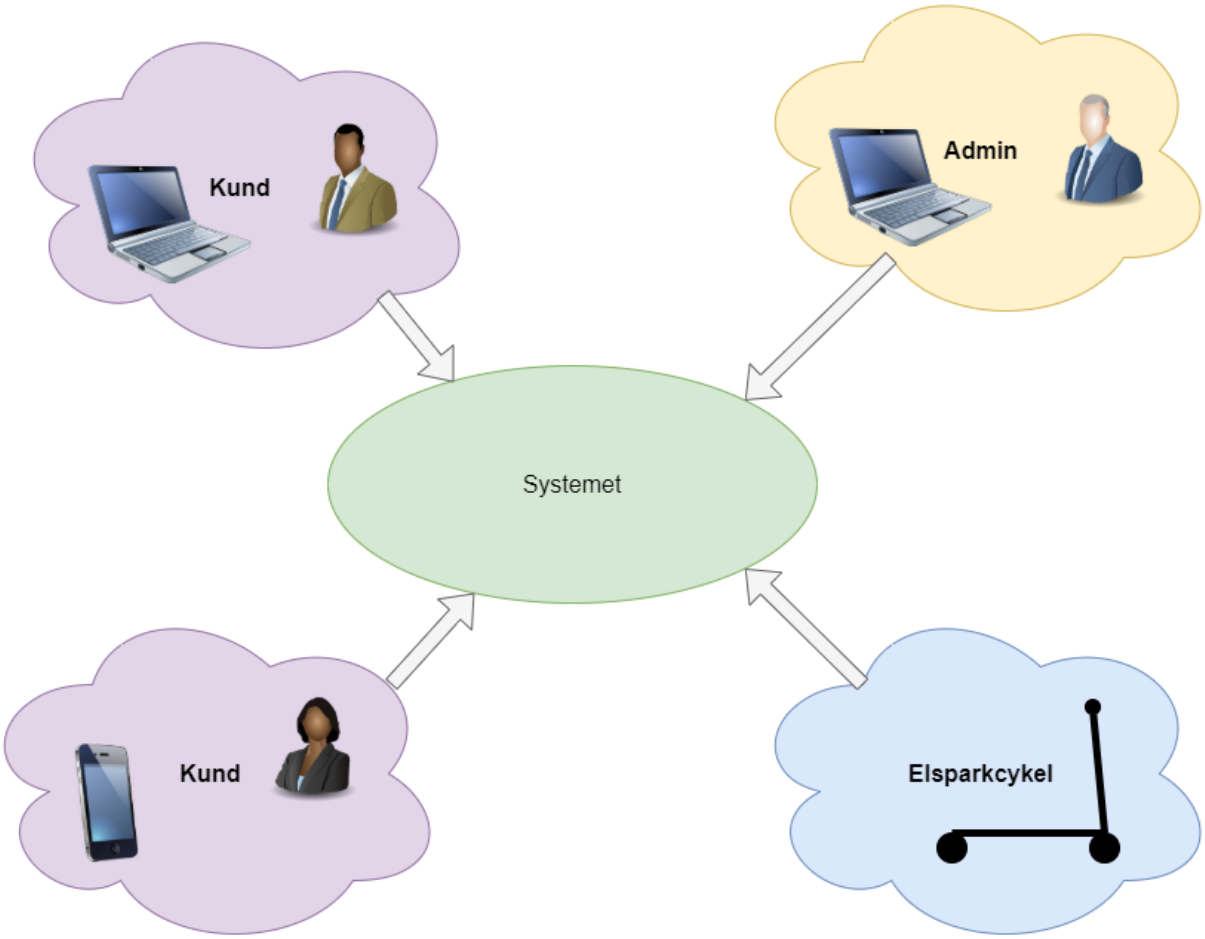
På uppdrag av Svenska elsparkcyklar AB ska det skapas ett system för att hyra och hantera elsparkcyklar. Systemet innefattar en app och webbklient för kunden, en administrativ webbklient och ett program för cykeln.

Via kundens app kan kunden hyra elsparkcyklar och via webbklienten kan kunden registrera sig och hantera sina resor. Den administrativa webbklienten ger en översikt över kunder, cyklar, parkeringsplatser, laddstationer och även möjligheten att hantera cyklarnas position. Cykelns program integrerar med cykeln och uppdaterar dess data och position. För att knyta ihop det behövs en backend som har ett API [1]. Detta API kommer sedan integrera med en databas där all nödvändig data lagras.

Översikt

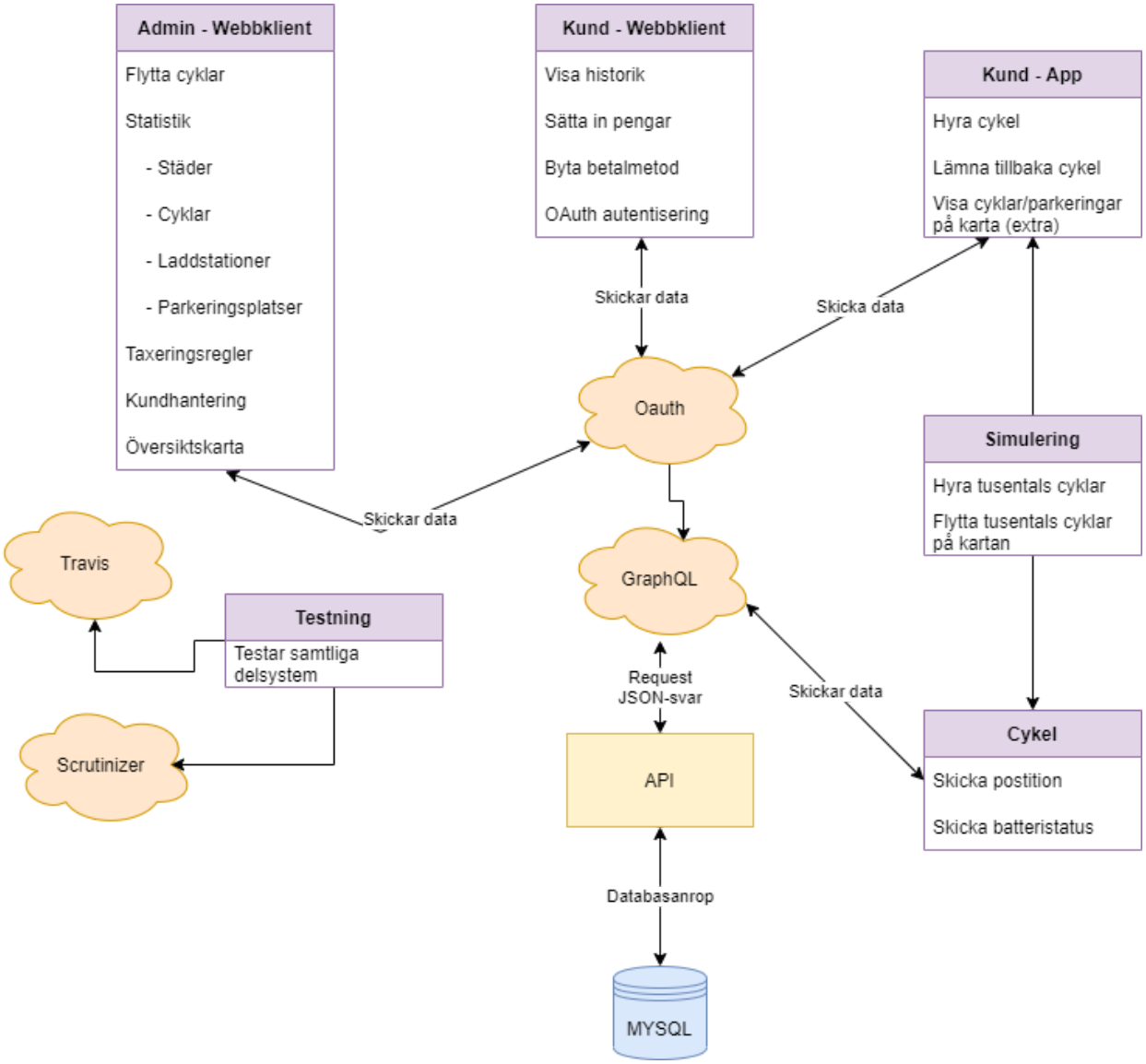
För den här delen av rapporten redogörs en övergripande bild av systemet och dess delar. Med övergripande så menas vilka delar som finns med och hur dessa jobbar med varandra.

En övergripande bild av systemets användare



Bilden ovan visar delarna som har kontakt med systemet. Dels är det kunder via telefoner eller webbplatser. En administratör via en webbplats samt cykeln via dess program.

En översikt över systemets respektive delar



Bilden ovan visar övergripande hur respektive del samverkar med varandra.

Delsystem

Beskrivning av varje delsystem som kommer innefattas av systemet.

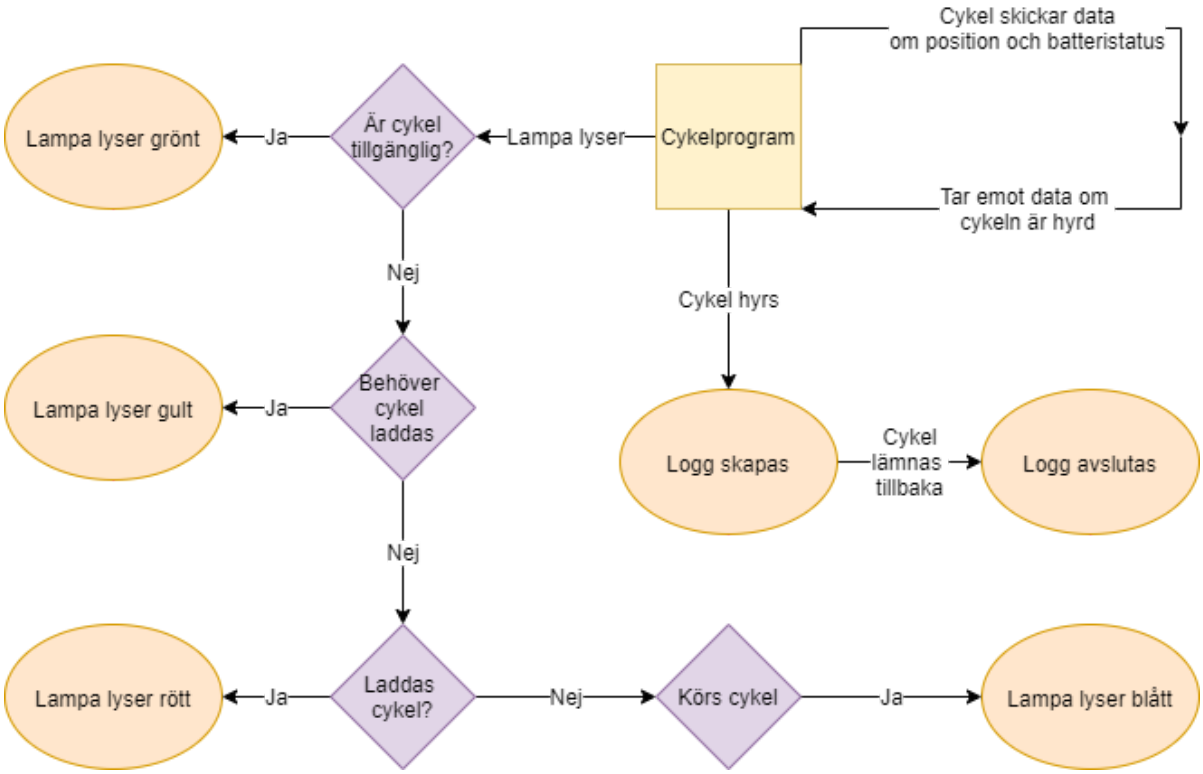
Cykelns program

Cykelns program kommer att skapas med JavaScript och Cordova. Cordovas plugin “Device Motion” kommer användas för att skapa rörelsehastighet, GPS-plugin kommer användas för att skapa position och cykelns program kommer att ansluta och kommunicera med servern via RESTful API.

När kunden hyr cykeln ska kunden registreras via en app.

- Appen har flera knappar för åtgärder som exempelvis start/stopp, åk, parkering samt laddning. Så kunden klickar på knappen som motsvarar hans handlingar (åk, parkering, laddning).
- Appen har också flera lampor/färger som indikerar cykelns status. Vid grön lampa finns cykel tillgänglig att hyra, vid blå används cykeln, vid rött laddas cykeln, vid gult behöver cykeln laddas och vid lila befinner sig cykeln på parkering.
- När en kund lämnar tillbaka cykeln måste kunden klicka på att stänga av så att cykeln inte längre går att använda.
- Appen kommer att skicka sin status och position till servern med regelbundna intervall som ställs in på servern.

Bild över cykelns programs funktioner och delar



En övergripande bild över hur cykelns program funkar. Den visar vad som påverkar färgen på lamporna på cykeln samt hur cykeln pratar med andra delar av systemet.

API

Tabell över alla entiteter och hur dem ska kunna manipuleras

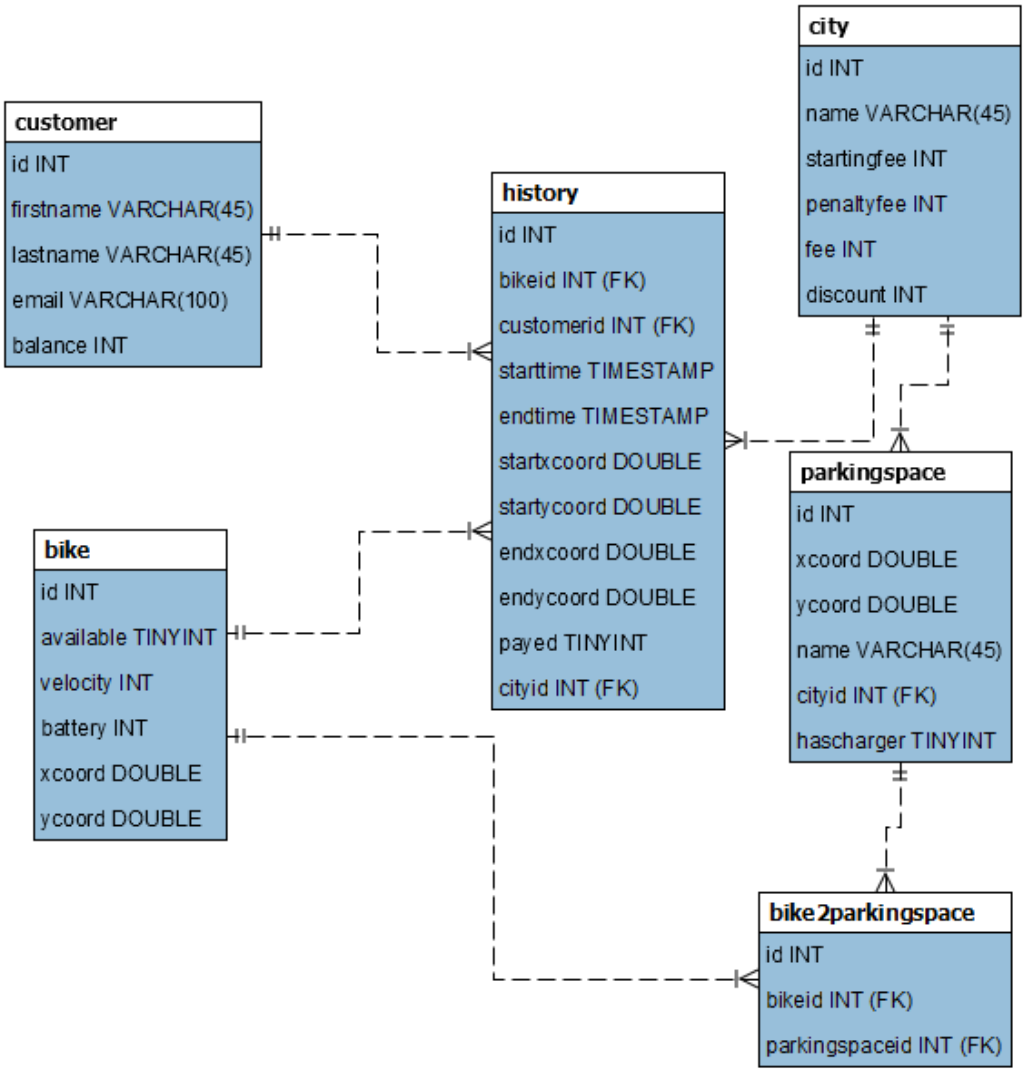
Entitet	Create	Read	Update	Delete
customer	X	X	X	X
bike		X	X	
city		X		
parkingspace		X		
bike2parkingspace		X		
history	X	X	X	

Systemets API [1] utvecklas för att undvika osäker hantering av databasen och för att underlätta för alla systemets andra delar med att manipulera all relevant data. För att spara på tid skapas det inte CRUD-funktioner [2] för alla olika entiteter. Det skulle inte heller vara nödvändigt att i framtiden låta systemets delar att använda sig av CRUD mot alla entiteter. Bästa exempel på det här är entiteten “bike2parkingspace”. Den här entiteten hanteras helt automatiskt av API:et som kollar kopplingar mellan “bike” och “parkingspace” och tar bort eller lägger till beroende på resultat. Därmed behöver användaren av API:et endast kunna läsa av den här entiteten.

För utveckling av systemets API har verktyget GraphQL [3] valts. Främsta anledningen till det här valet var att man med hjälp av GraphQL automatiskt kan kolla kopplingar mellan olika entiteter, hämta tillhörande data och sedan packa ihop denna data till ett enda paket innan det skickas till användaren. Det är en stor fördel jämfört med REST-modellen [4] där användaren hade behövt kolla kopplingarna själv och hämta även oönskad data som sedan kan hanteras efter det tagits emot. Andra anledningen till valet är att all data samlas under en enda endpoint [4]. Hade valet hamnat på REST-modellen hade alla entiteter varit utspridda på varsitt endpoint. Sista anledningen är inkluderingen av GraphiQL som är ett grafiskt gränssnitt för att pröva olika förfrågningar till API:et. Det här underlättar för användaren som snabbt kan se resultatet av sin förfrågan.

Databas

Bild över databasens olika tabeller och dess kopplingar

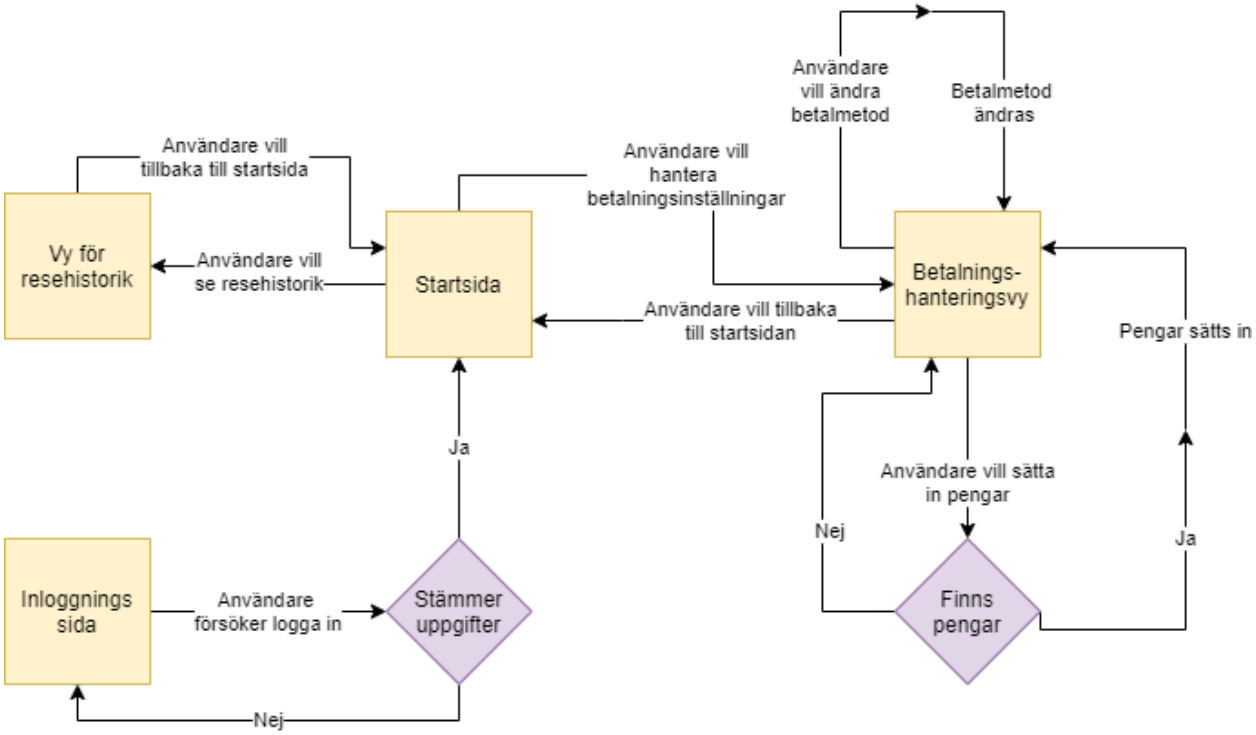


För skapandet av databasen kommer programmet MySQL Workbench [5] att användas. MySQL Workbench gör det mycket smidigt att skapa en uppkoppling mot databasen, skriva välstrukturerade SQL-förfrågningar och sedan spara dessa i SQL-filer. En annan funktion i programmet som är till stor hjälp är ritandet av eer-diagram [6]. Utifrån de diagram som ritas i MySQL Workbench kan det genereras SQL-kod som underlättar vid skapandet och eventuellt återskapande av databasen.

Slutkundens webbklient

Klienten kommer att skapas med hjälp av Javascript och Mithril som ramverk. Valet där faller på att underhållet kan då underlättas då samma språk används på flera ställen. Det hade fungerat lika bra med Angular eller Vue. För att användare ska kunna autentisera sig används OAuth enligt krav från beställaren.

Bild över webbklientens olika delar och funktioner



Bilden visar en översikt över delarna slutkundens webbklient innefattar.

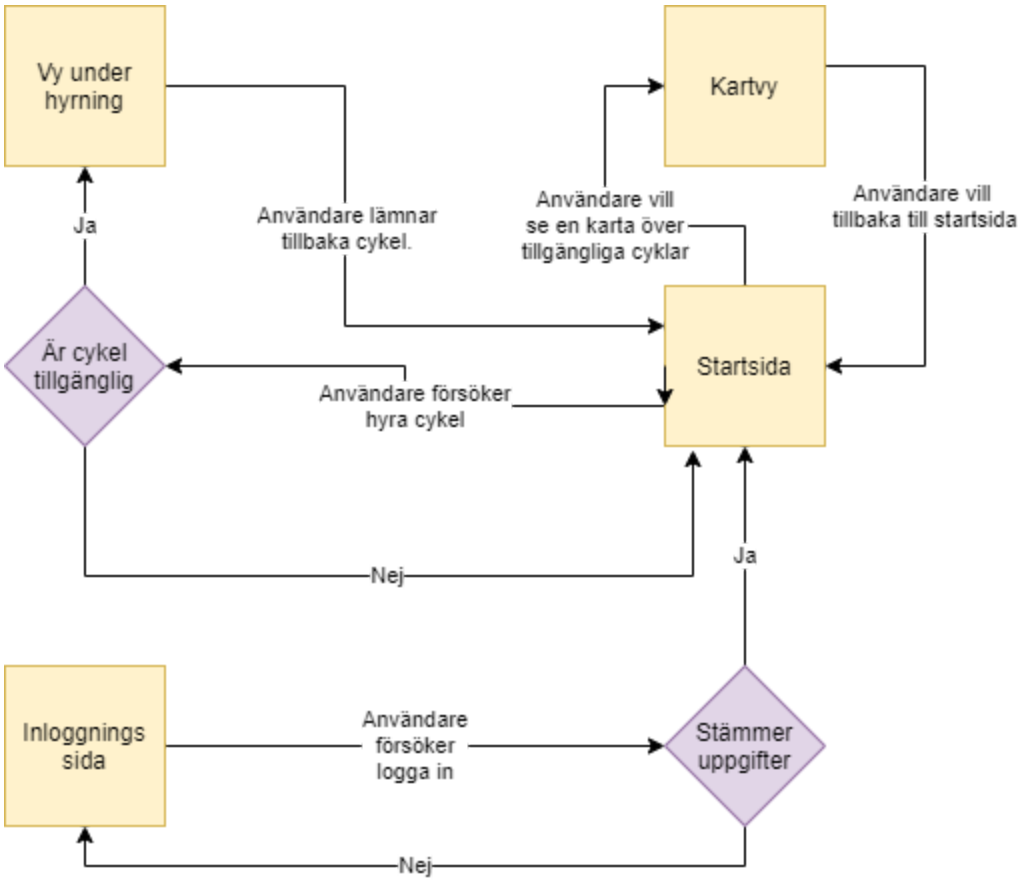
Extra

Finns tid ska karta skapas med hjälp av leaflet där resor från historiken visas.

Slutkundens app

Appen utvecklas i JavaScript med hjälp av Cordova för att kunna driftsätta det på telefon. Huvudanledningen till att använda just Cordova är att man får möjlighet att komma åt telefonens inbyggda hjälpmedel, så som GPS och Kamera. Som ramverk används Mithril. För att skapa produktionsveriionen kommer SASS och Webbpack att användas som stöd. För verifiering kommer OAUTH att användas till stöd, vilket är krav från kund.

Bild över appens olika delar och funktioner



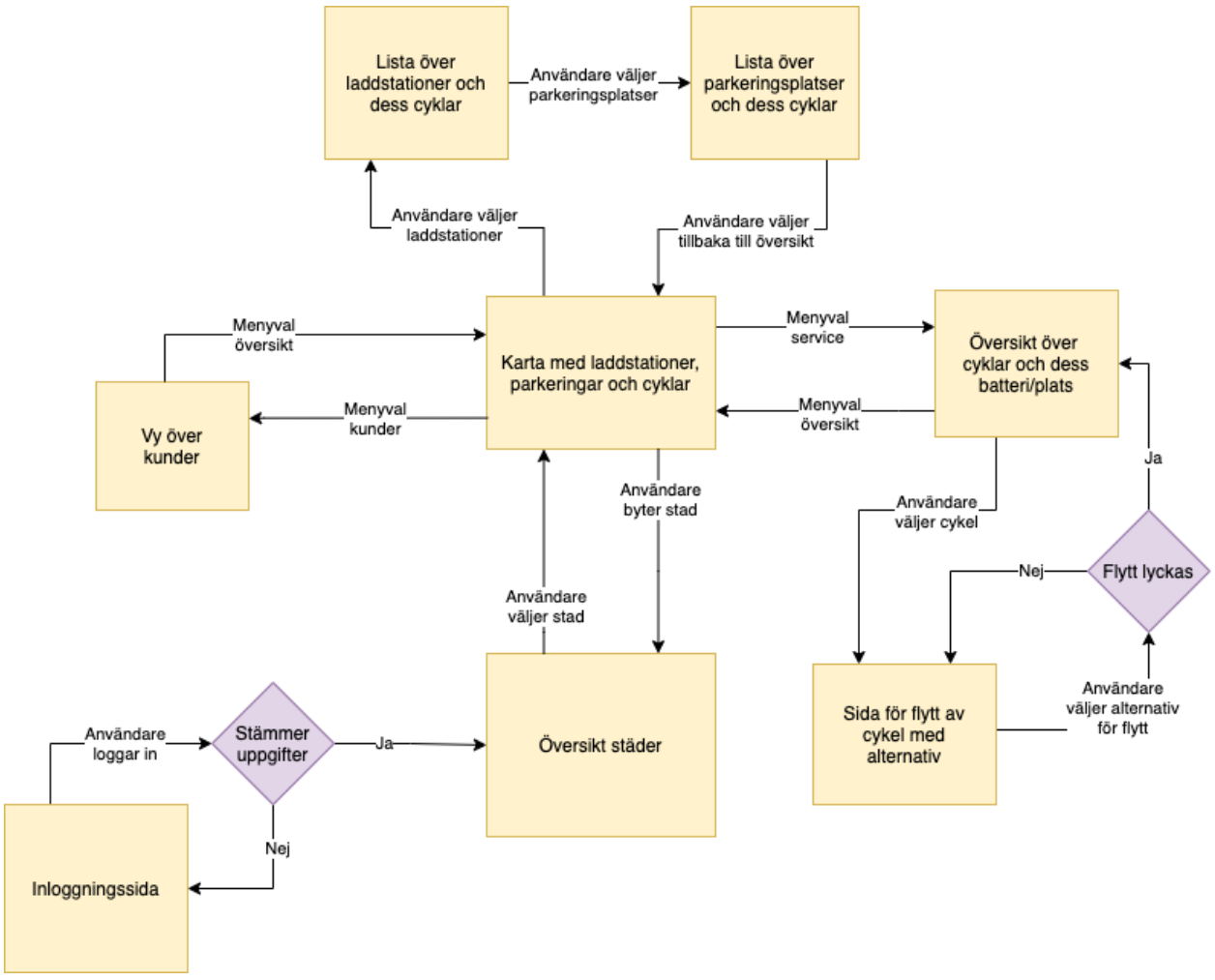
Bilden visar en översikt av appens funktioner och delar.

Extra

Finns tid så kommer Leaflet användas för att skapa kartor och plug in för att kunna läsa cykel id som QR med hjälp av telefonens kamera.

Administrativ webbklient

Då den administrativa webbklienten ska, enligt kravspecifikationen från kund, erbjuda status för cyklar, stationer och kunder samt visa karta med positioner kommer det att utvecklas med JavaScript och ramverket Mithril. Då vi tidigare har utvecklat applikationer som har integrerat med API och möjligheten till karta föll valet naturligt på JavaScript och Leaflet för att kunna använda sig av tidigare kunskaper och kunna uppfylla kundens krav inom tidsramen för projektet.

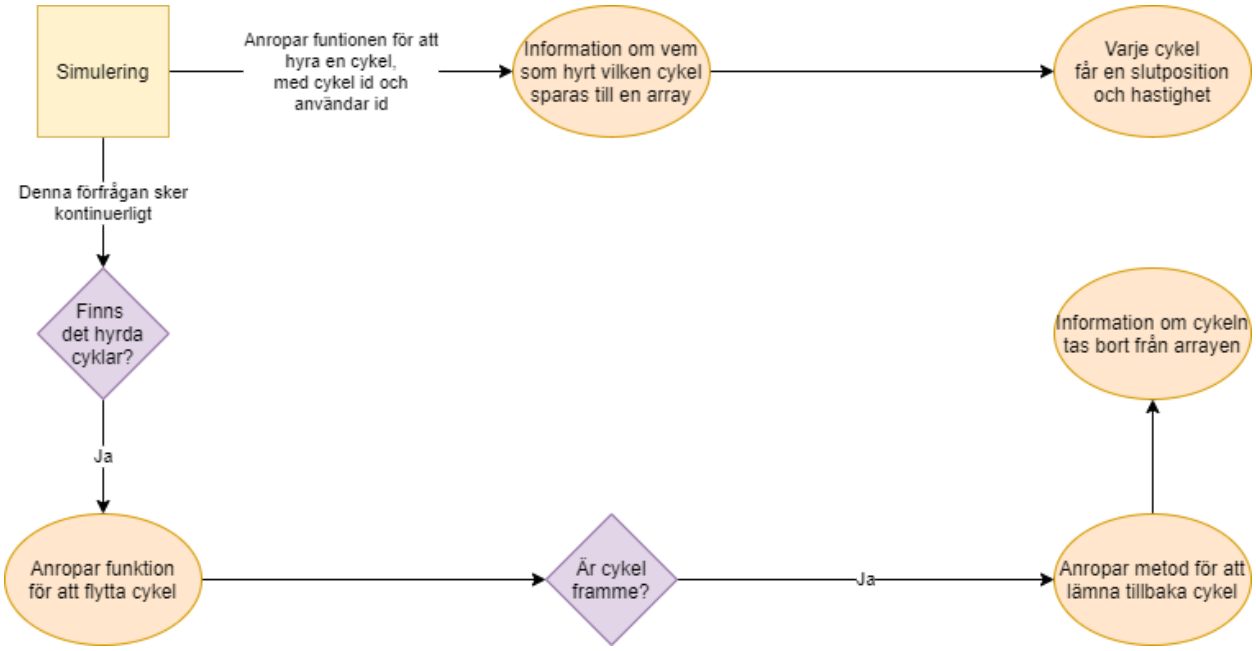


Bilden visar en översikt av den administrativa webbklientens delar och funktioner.

Simulering

För att kunna testa att systemet fungerar som tänkt behöver vi simulera uthyrning av cyklar och att dessa då faktiskt rör sig på kartan över var cyklar finns. För att kunna lösa det kommer vi ta hjälp av funktioner som finns dels för kundens app, där det är möjligt att hyra en cykel och lämna tillbaka den. Men istället för att behöva skapa tusentals instanser av användare, kommer parameters skickas med för användarnummer, för att simulera att en viss användare hyr en cykel. På samma sätt kommer det skickas in parameter till en viss cykel för att ändra cykelns position så att det ser ut som att dom hyrda cyklarna rör på sig.

Övergripande bild över hur simuleringen ska genomföras



För att kunna simulera att systemet körs, behövs det i simulerings-klienten en array som håller koll på vilka cyklar som är hyrda så att dess position kan uppdateras vid bestämda intervall.

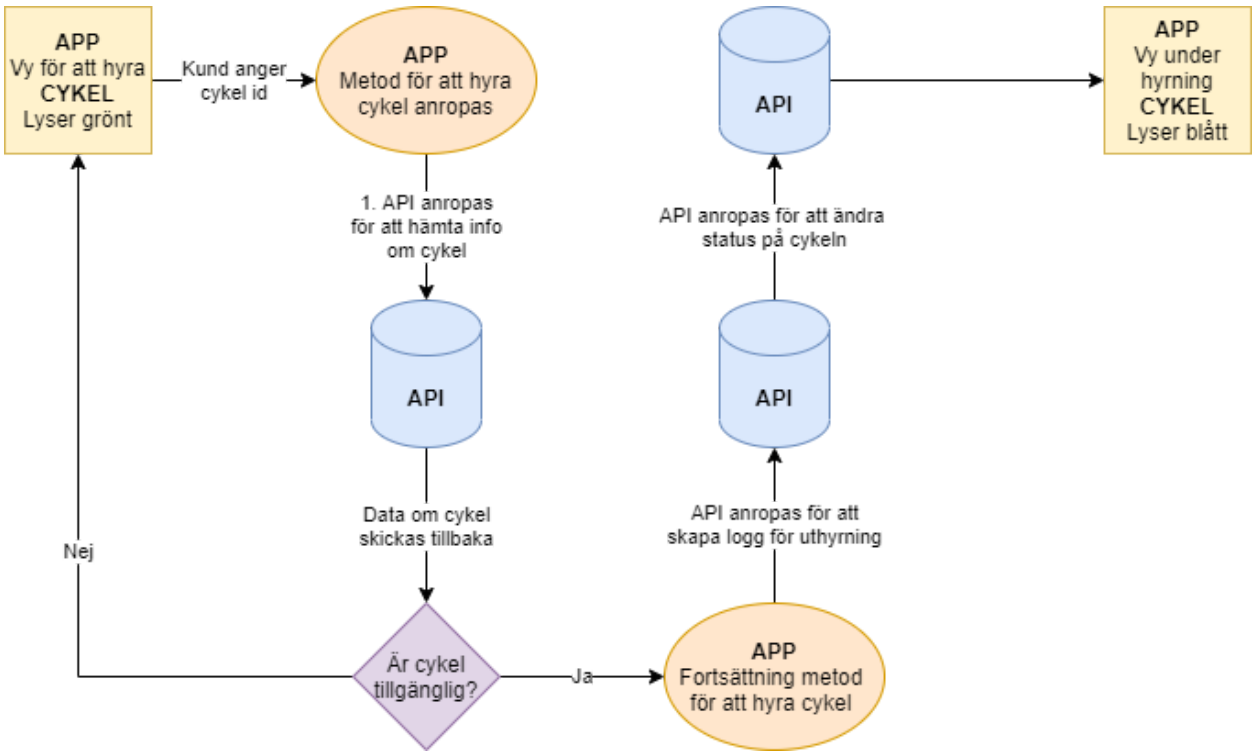
Use cases

En ingående redogörelse för use-cases som vi anser är viktigast.

Slutkund ska kunna hyra elsparkcykel

För att hyra en cykel är det främst två delar av systemet som behöver samverka. Det är appen som användaren hyr cykeln ifrån. Den behöver sedan i olika delar av processen för att hyra en cykel anropa systemets API. Dels så behöver den information om elsparkcykeln. Den behöver även kunna uppdatera cykelns status för cykeln och skapa en logg för uthyrningen. Den sista delen av systemet som är inblandad är cykeln, vars färg på lampa kommer gå från grön till blå, när uthyrningen är genomförd.

Övergripande bild över stegen som ingår i hyrning av en elsparkcykel



Bilden visar vilka delar av systemet som samverkar i vilken del av kedjan för att hyra en cykel.

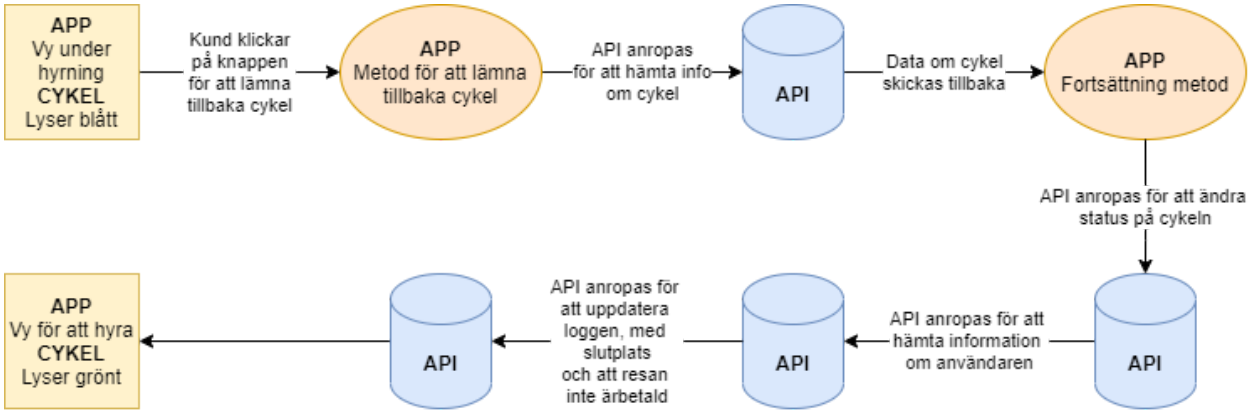
Slutkund ska kunna lämna tillbaka elsparkcykel

För att lämna tillbaka en cykel är det samma komponenter som i systemet som arbetar som vid hyrning. Dels är det appen som skickar att cykeln ska lämnas tillbaka, dels är det API som tar emot flertalet anrop från appen för att hämta och uppdatera data i diverse databastabeller och dels är det cykeln som med sin status ändrar färg på lampan från blå till grön vid avslutad uthyrning. Det finns tre olika scenarion som kan ske vid återlämning och dessa är beroende av vad kunden har för betalningsmetod och hur mycket pengar kunden har på kontot. Dessa redogörs för här nedan.

- Kunden har valt månadsbetalning.
- Kunden har valt direktbetalning, men har för lite pengar på kontot för att kunna betala resan.
- Kunden har valt direktbetalning och har tillräckligt med pengar för att betala resan.

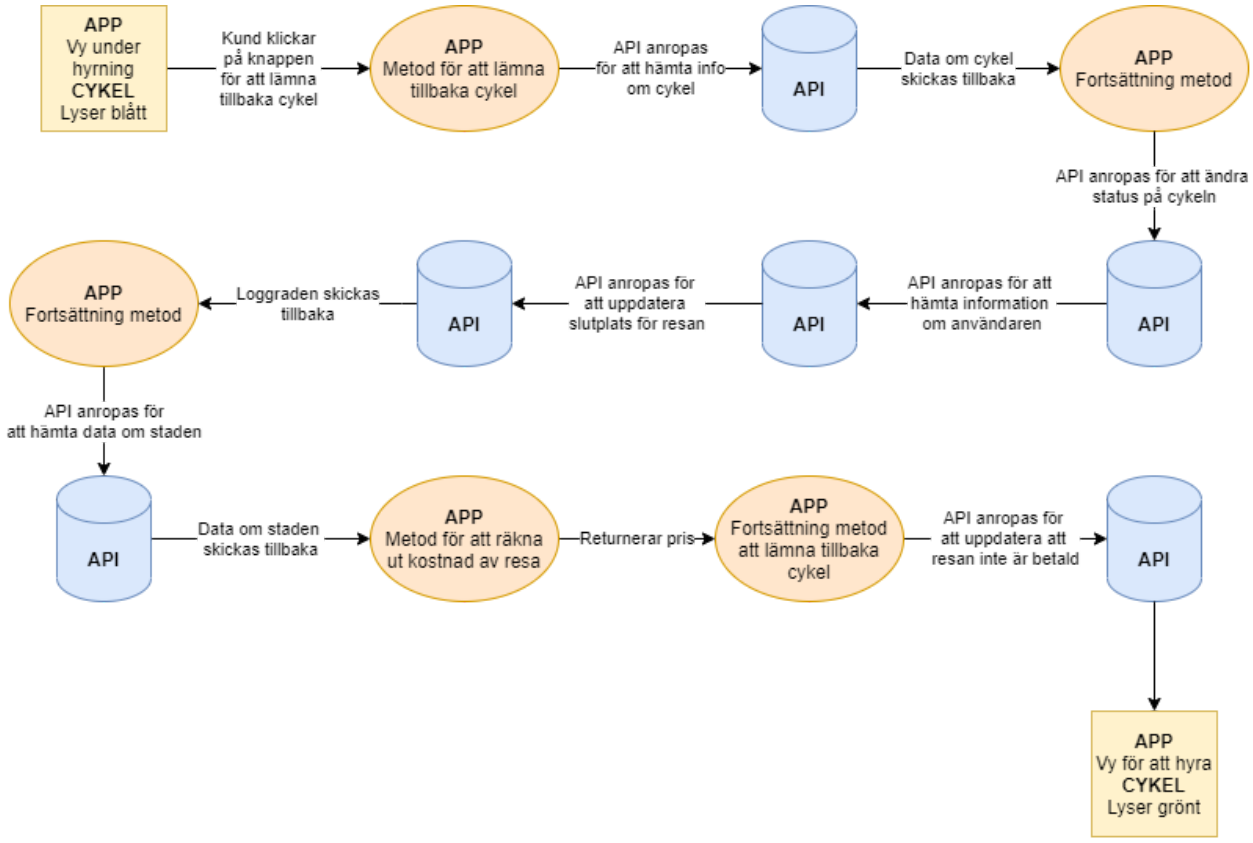
För mer ingående genomgång av respektive fall finns en bild över respektive fall nedan.

Bild över tillbakalämning av cykel med månadsbetalning



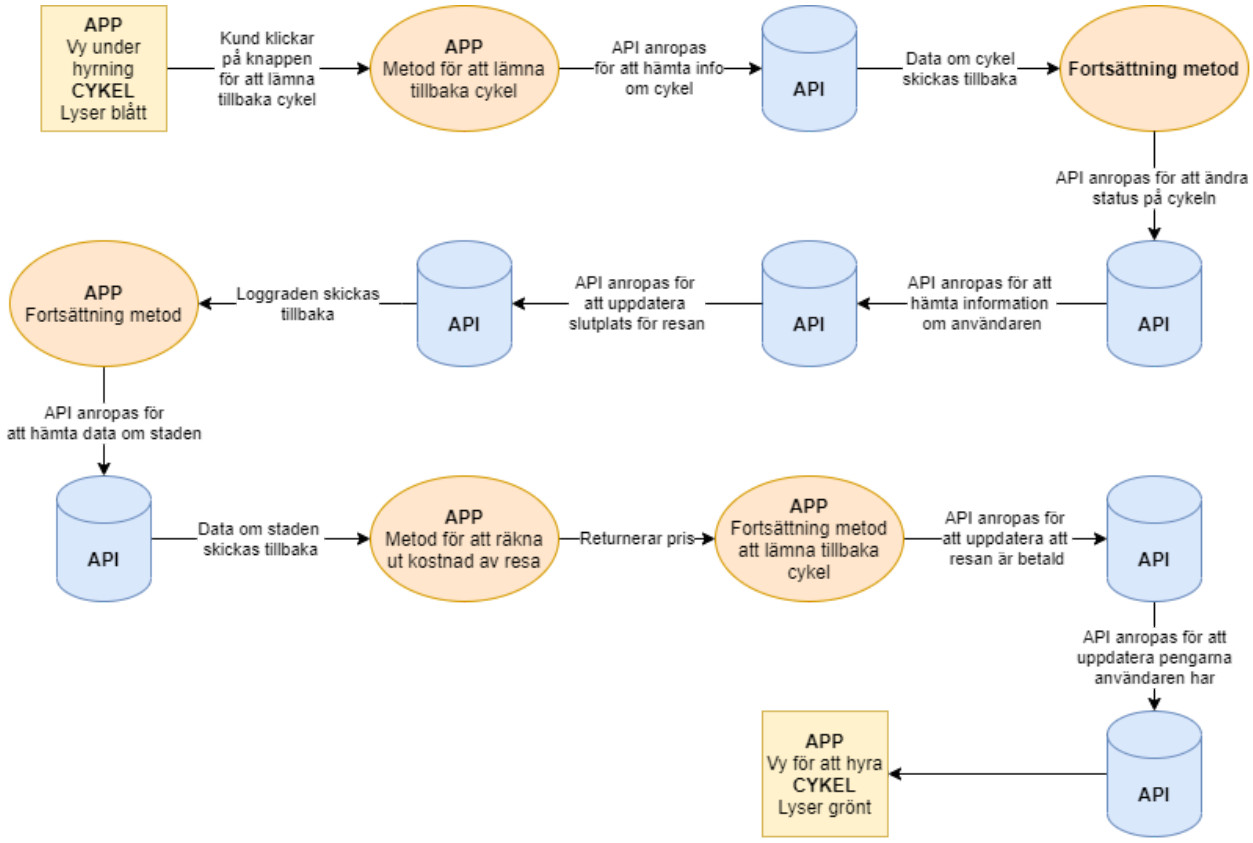
För återlämning av cykel med månadsbetalning är flödet relativt kort, då enbart cykeln status samt loggen behöver uppdateras.

Bild över tillbakalämning av cykel med direktbetalning och för lite pengar



För att lämna tillbaka en cykel om man har direktbetalning behöver uppdateringen av logg-tabellen i databasen göras i två steg. Först måste man lägga till slutposition för att få en sluttid. Den används sedan för att räkna ut pris på resan. När man har för lite pengar på kontot så uppdateras sedan loggen med att resan inte är betald.

Bild över tillbakalämning av cykel med direktbetalning

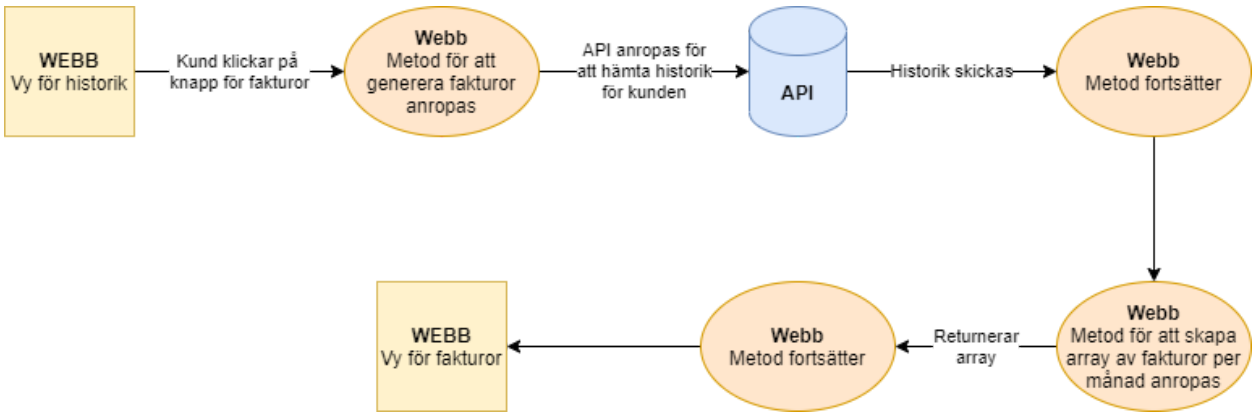


Likt direktbetalning när man inte har tillräckligt med pengar så behöver logg-tabellen i databasen uppdateras i två steg när man har tillräckligt med pengar. Skillnaden är att man andra gången uppdaterar med att resan är betald och att det då uppdateras pengarna kunden har på kontot.

Slutkund ska kunna betala resa

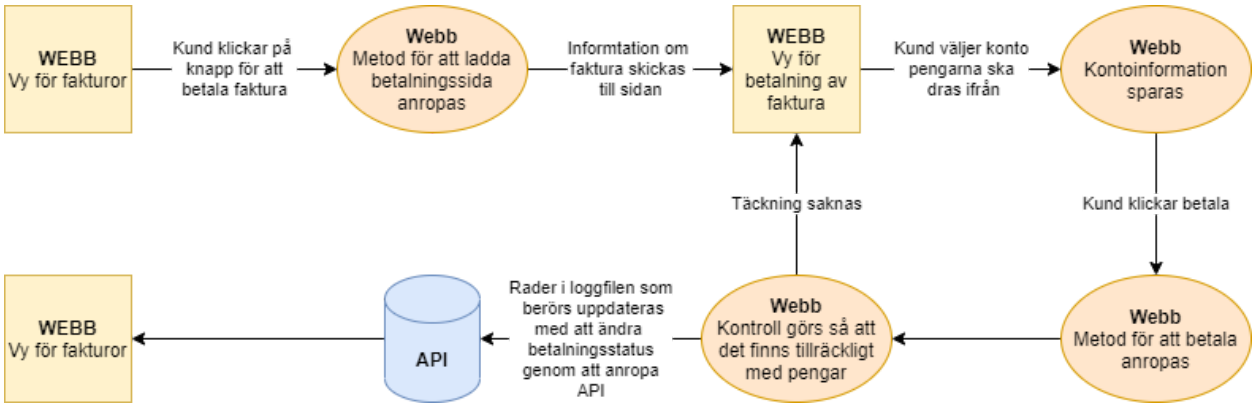
För att betala en resa finns det två scenarion, dels direktbetalning, vilket har redogjorts för under föregående kapitel när cykel lämnas tillbaka. Det andra är månadsbetalning vilket ska redogöras för i detta stycke. Varje månad genereras en faktura där en totalsumma för månadens resor summeras, dels med detaljer för varje resa, men även översikt med totalsumma och betalningsinformation. För att kunna genomföra det, behövs anrop mot databasen för att hämta loggraderna för en användare och sedan filtrera ut dessa per månad och bara visa raderna som inte är betalda. Dessa skapar sedan materialet för fakturan.

Bild över generering av faktura



För att generera fakturor behöver man hämta data för en användares logg från API och sedan filtrera ut det per månad.

Bild över betalning av faktura



För att kunna betala en faktura behöver man välja faktura man vill betala och vilket konto man vill betala det ifrån, därefter görs en koll ifall det finns tillräckligt med pengar. Är fallet så, uppdateras fakturans rader till betalt med hjälp API anrop.

Flytt av cykel via administrativ webbklient

För att kunna genomföra flytt av cykel behöver systemet hämta information från API för att fastställa cyklarnas position inför flytt. När cykeln har fått en vald plats behöver systemet integrera med API för att uppdatera cykelns position och även om cykeln hamnar på en laddningsstation och därmed får status laddas.

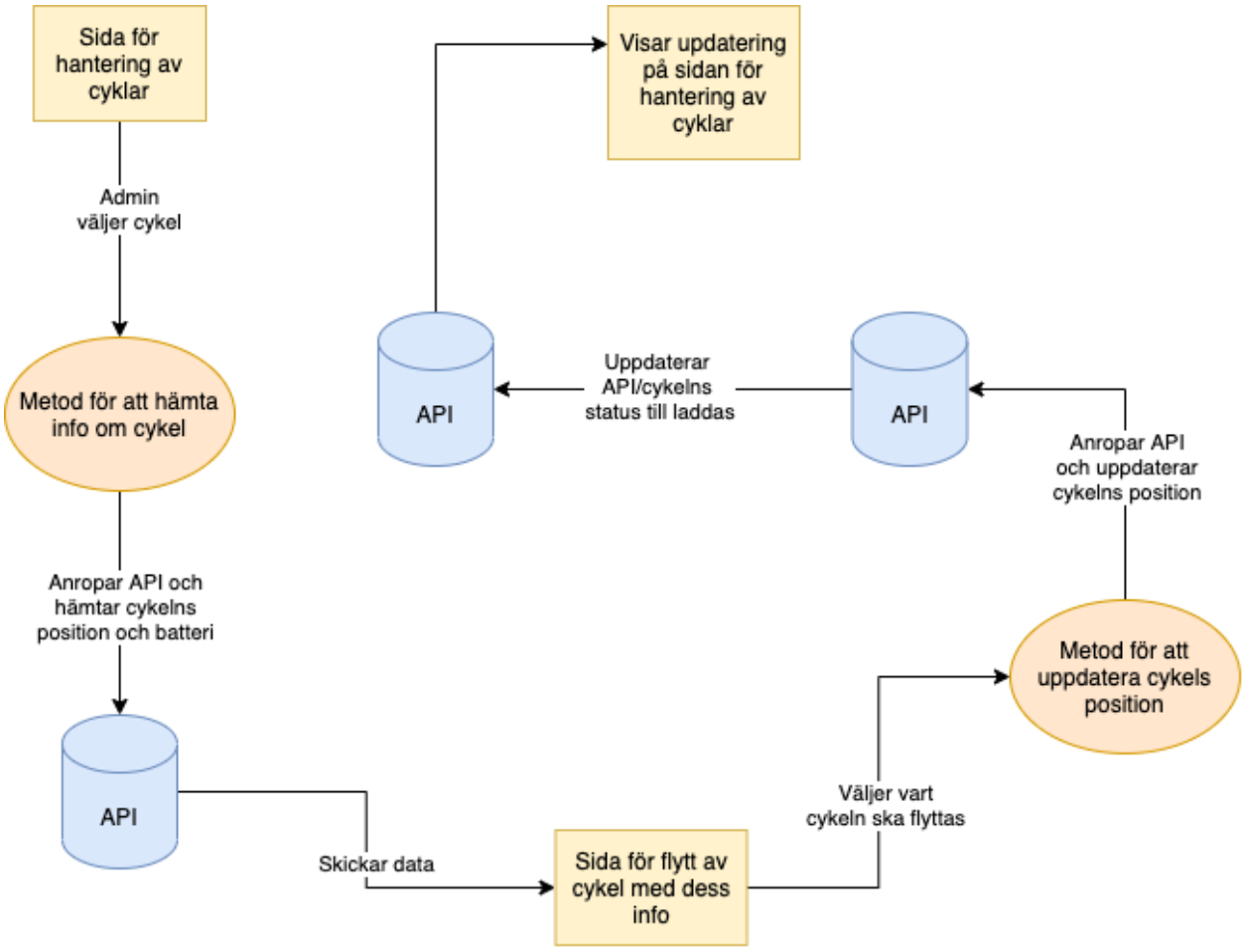


Bild för systemets integration med API vid flytt av cykel.

Referenser

[1] Petr Gazarov 2019-12-19 What is an API? In English, please.

https://www.freecodecamp.org/news/what-is-an-api-in-english-please-b880a3214a82/

[2] Alexandra Altvater 2017-05-02 What are CRUD Operations: How CRUD Operations Work, Examples, Tutorials & More

https://stackify.com/what-are-crud-operations/

[3] 2019-01-08 What is GraphQL?

https://www.redhat.com/en/topics/api/what-is-graphql

[4] Tehreem Naeem 2020-01-28 REST API Definition: What is a REST API (RESTful API)?
https://www.astera.com/type/blog/rest-api-definition/

[5] MySQL Workbench

https://www.mysql.com/products/workbench/

[6] J Riyana 2020-03-24 Drawing ER and EER Diagrams & Relational Mapping

https://medium.com/nerd-for-tech/drawing-er-and-eer-diagrams-mapping-4965e2b3cc3e