Systemdesignspecifikation

Svenska Elsparkcyklar AB

2023-11-07

Vteam Grupp 7 - Bästa gruppen

Linda, Déspina, David & Adam

Innehållsförteckning

Inledning

Detta System Design Specification har skapats för att beskriva arkitekturen av webbapplikationen. Det är en detaljerad dokumentation som beskriver de tekniska och funktionella kraven för systemet vi ska bygga.

Bakgrund

Företaget Svenska Elsparkcyklar AB har behov av ett nytt system som kan hantera uthyrningen av el-sparkcyklar inom stadsområden. Företaget är i nuläget aktiva i 3 olika städer och planerar att expandera deras verksamhet till flera städer med stöd av ett nytt datorsystem.

Projektet har sin bakgrund i kommunens strävan att hantera problem och oordning som uppstått på grund av att antalet el-sparkcyklar har ökat. Dessa sparkcyklar har blivit en anledning till klagomål och oro från invånarna på grund av oegentligt parkerade fordon, överträdelse av trafikregler samt övergivna el-sparkcyklar som orsakar problem i stadsmiljön.

Det är här vi kommer in, vi har fått i uppdrag att designa deras nya system. Vi har som mål att skapa ett system som förenklar hanteringen av elsparkcyklarna, kundinteraktioner och samtliga aspekter av verksamheten. För att adressera problematiken ovan och säkerställa att elsparkcyklar används i enlighet med gällande lagar och föreskrifter ska fokus läggas på cykelns program. Detta program kommer att innefatta loggning av varje resa som görs, med insamling av start- och slutpunkt (inklusive plats och tid) samt information om användaren. Syftet med detta projekt är därmed att skapa en lösning som möjliggör bättre övervakning och reglering av elsparkcyklar för att förbättra trafik- och stadsmiljön.

Systemarkitektur

När vi utvärderade olika arkitektoniska alternativ för vår applikation, genomförde vi en noggrann diskussion i projektgruppen där vi vägde för- och nackdelar av både Layered Architecture och Microservices Architecture.

Layered Architecture är en arkitektonisk modell där en applikation är strukturerad i olika lager som ansvarar för specifika uppgifter. Detta ger en hierarkisk struktur.[3]

Microservices Architecture är en annan arkitektonisk modell där en applikation är uppdelad i små oberoende tjänster som kommunicerar med varandra över nätverket. [3]

Vår bedömning är att vi inte förväntar oss behöva införa nya teknologier i framtiden, vilket innebär att denna aspekt inte utgör en betydande nackdel för Layered Architecture i vår kontext. Vi noterade också fördelen med enkelheten som Layered Architecture erbjuder genom att följa en tydlig hierarkisk struktur. Vi valde därmed Layered Architecture som vår arkitektoniska modell. Vi tror att detta val även kommer att möjliggöra en effektiv och smidig utvecklingsprocess.

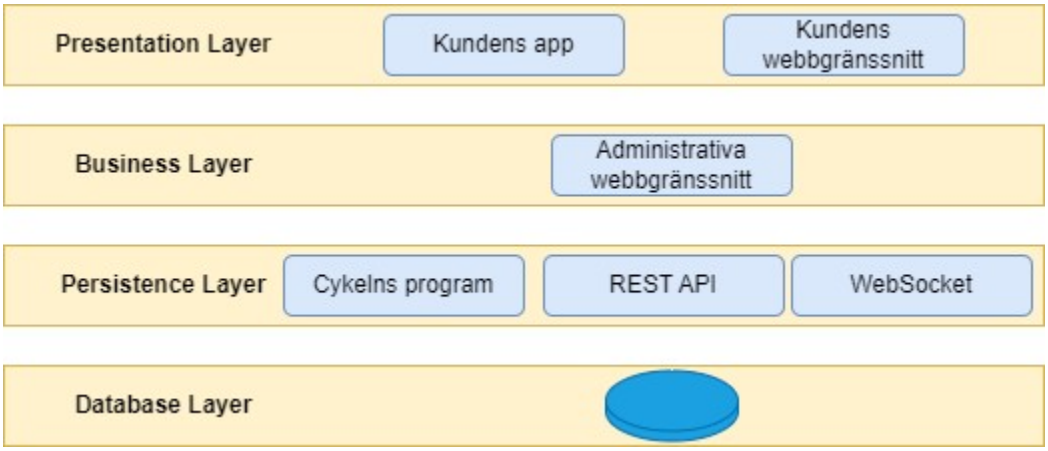

Bild 1. Översikt av systemet.

Databas

Datan kommer att sparas i en relationsdatabas för att underlätta hämtning och lagring av data med mycket korsreferenser.

För att stödja denna datan ämnar vi ha bastabeller för kunder, elsparkcyklar, zoner och administratörer samt resor och parkeringar i associativa tabeller. Nedan finns ett första utkast i ett ER-diagram.

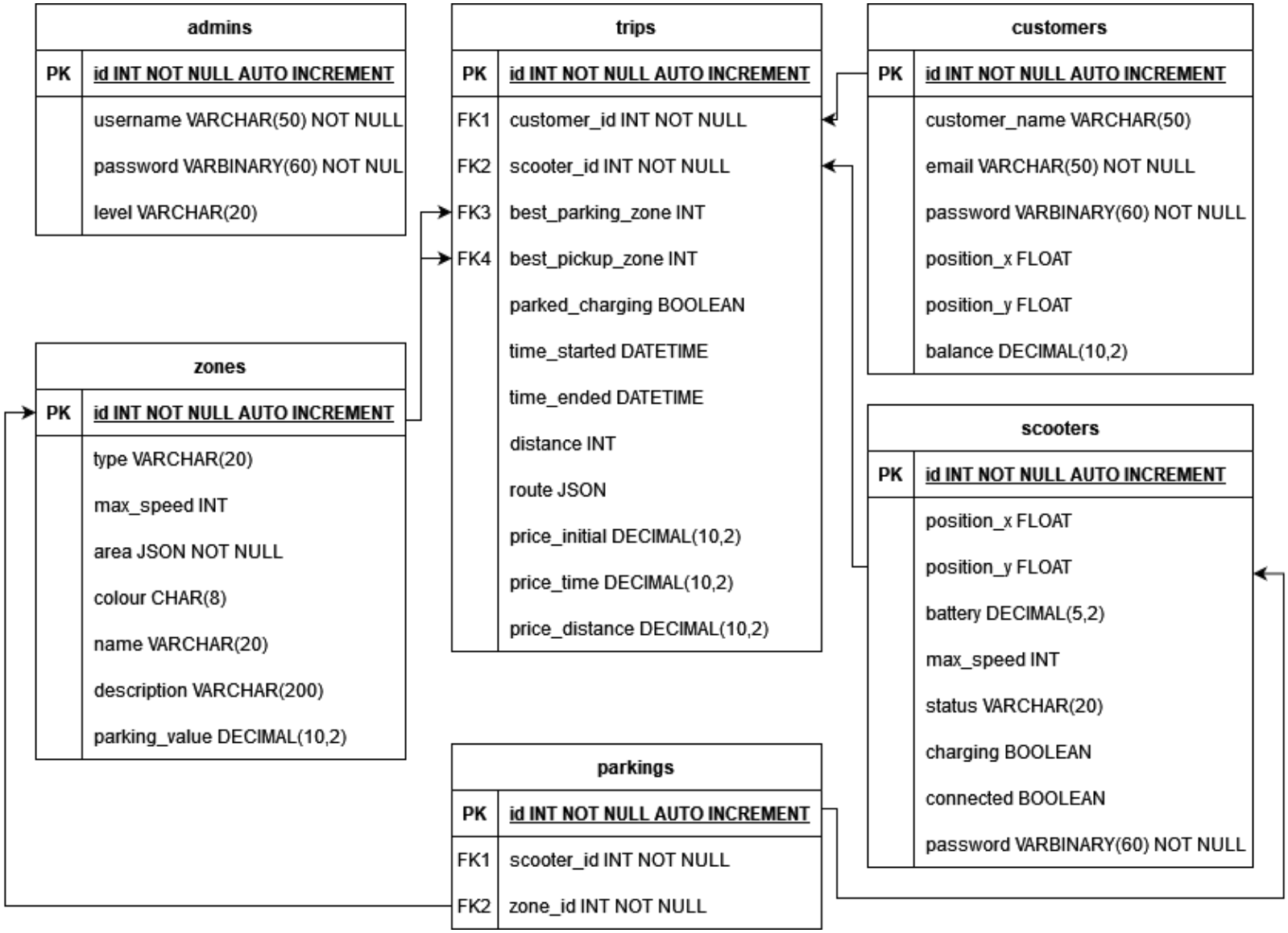

Diagram 1. ER-diagram över databasen

Tekniska val

- **Back-end**: Vi har valt att använda Node.js och Express, tillsammans med MariaDB som vår databas.
- **Språk och verktyg:** Vi kommer att använda oss av Typescript för att förbättra utvecklingen för samtliga medlemmar.
- **Kodstandard:** För att upprätthålla en konsekvent kodstandard kommer vi att använda ESLint med standardpaket för Typescript.
- **Kollaborationsverktyg:** För att enkelt kunna dela kod samt uppdelning av ansvarsområden så har vi bestämt oss att vi ska använda GitHub och GitHub-Projects. Dessutom använder vi oss av ett Azure Container Registry för att kunna arbeta med samma Docker-images på ett enkelt vis.

Motivering till val

Vi har valt *Node.js*, *Express* och *MariaDB* eftersom vi har goda erfarenheter med denna verktygs kombination. Vi har tidigare sett framgångsrika resultat med kombinationen, och det är även vad alla i teamet kände sig mest bekväma med.Vi hade ett flertal val vi kunde ha gjort av stacks som inkluderar Python/django, PHP/Symphony ,MySQL men för gruppen så kändes det bäst med Node.js stacken. Vi valde även *TypeScript* eftersom det förenklar utvecklingsprocessen i en större grupp genom att tydligt deklarera datatyper och de förväntade värdena. Detta kommer att hjälpa oss att tidigt identifiera fel, undvika dem och förbättra kodens kvalitet.

GitHub använder vi för att dela vår kod mellan varandra, vi har preliminärt valt att dela upp repon inom olika ansvarsområden vi tror detta kommer hjälpa oss att undvika konflikter i merge etc under den praktiska kodningen av projektet.

Vi har valt att använda oss av ett *GitHub Project* för att ha en bra uppdelning på vad som både finns att göra och det som vi har gjort. Detta har varit ett bra sätt för oss att dela upp ansvar och uppgifter, det kommer även förenkla för oss att se vilka uppgifter som behöver avklaras och problem som behöver åtgärdas.

För att upprätthålla en kodstandard har vi valt att installera *ESLint* med

standardpaket för Typescript. Vi anser att en tydlig kodstandard är avgörande för projektets framgång då alla i projektet ska ha möjligheten att snabbt förstå vad som händer i koden. Detta kommer hjälpa oss att underhålla koden samt förenkla överlämningen till våra andra gruppmedlemmar.

Test och verifikation

Systemet ska ha lösningar för att testa och verifiera funktionalitet.

Enhetstester

Samtliga undersystem och appar ska ha enhetstester som testar en rimlig del av den funktionella koden, samt testa den på ett ändamålsenligt vis. En förhoppning är en kodtäckning på 90%, men vid särskilda fall kan en täckning så låg som 70% vara godtagbar. Samtliga delar av systemet ska ha en CD-implementation från GitHub Actions eller motsvarande som kör testerna automatiskt.

Verifiering

Systemet ska konstrueras för att lätt kunna verifieras genom simulering. Genom att utgå från ett repo med en docker-compose.yml så ska systemet kunna köras med en exempeldatabas. Det ska då exempelvis gå att logga in på en administratörsinloggning för att se kunder och elsparkcyklarnas position på en karta.

Dessutom kommer det att finnas två mockningstjänster för att möjliggöra simuleringen. Den första är ett script (t.ex. Node-script) som ansluter till scooter-appen via WebSocket och hoppar in istället för de elektroniska system som elsparkcykeln kommer ha i den riktiga implementationen; GPS-system, batteriladdning, hastighetsmätare.

Den andra mocknings tjänsten är ytterligare ett script som kommer fungera som kunderna i simuleringen. De kommer logga in via REST API:t och WebSocket-servern och hyra cyklar och resa runt med dem. Denna tjänst kommer behöva hantera planerande och resande av rutter samt kunna skicka meddelanden till scooter-mock-tjänsten för att berätta att cykeln rör på sig tillsammans med kunden.

Logisk vy

Användaren

Systemet ska stödja för användare att hitta elsparkcyklar på en karta, samt kunna få förslag på närmaste elsparkcykel. Användaren ska kunna låsa upp/checka ut en närliggande elsparkcykel och då låses den upp för användning. Elsparkcykel ska nu vara upplåst fram tills någon av följande inträffar:

- användaren parkerar den
- användaren och cykeln är för långt bort från varandra
- cykeln åker utanför en bestämd zon
- anslutningen mellan något att de relevanta systemen får ett avbrott

Administratör

Administratören ska kunna se alla elsparkcyklar och användare i en karta. Hen ska också kunna skicka ut servicemeddelanden till användarna (såsom trafikinformation, avbrott i någon del av systemet). Dessutom ska administratören kunna begränsa en cykels hastighet eller stoppa den helt. Administratören ska också kunna markera en cykel för upplockning av service. Kartan ska kunna filtreras, till exempel efter batterinivå eller om cykeln är markerad för service.

Utvecklings vy

Utvecklings vyn handlar om hur man hanterar applikationen och vilka metoder och standarder man använder för att göra detta. [1,2]

GitHub

Systemet som helhet kommer utvecklas med hjälp av GitHub Organization utifrån olika repository. Detta för att säkerställa att de olika komponenterna går att utveckla och testa enskilt eller som en helhet. Projektet och dess repositories finns på GitHub under https://github.com/best-scooter. Även GitHub Actions kommer

integreras i projektet för att skapa ett CI/DD-flöde för att säkerställa att de olika utvecklarnas arbete är kontinuerligt synkroniserat.

Docker

Docker/Docker-Compose kommer användas i enlighet med kravspecifikationen för driftsättning. Målsättningen är att de olika komponenterna antingen ska kunna köras och testas enskilt i olika containers, alternativt hela systemet. Bilderna till Docker finns på ett Azure Container Registry under utveckling och kommer att publiceras på DockerHub inför presentationen.

Komponenter

Systemet kommer bestå av följande komponenter:

- Server
    - Databas med kunder respektive cyklar
    - Backendservice/REST API
    - WebSocket server
    - Autentisering (Externt via OAuth provider)
- UI
    - Webbgränssnitt för administratör
    - Webbgränssnitt för kund
    - Mobilapplikation för kund
- Cykelsystem
    - Program för elsparkcyklarna

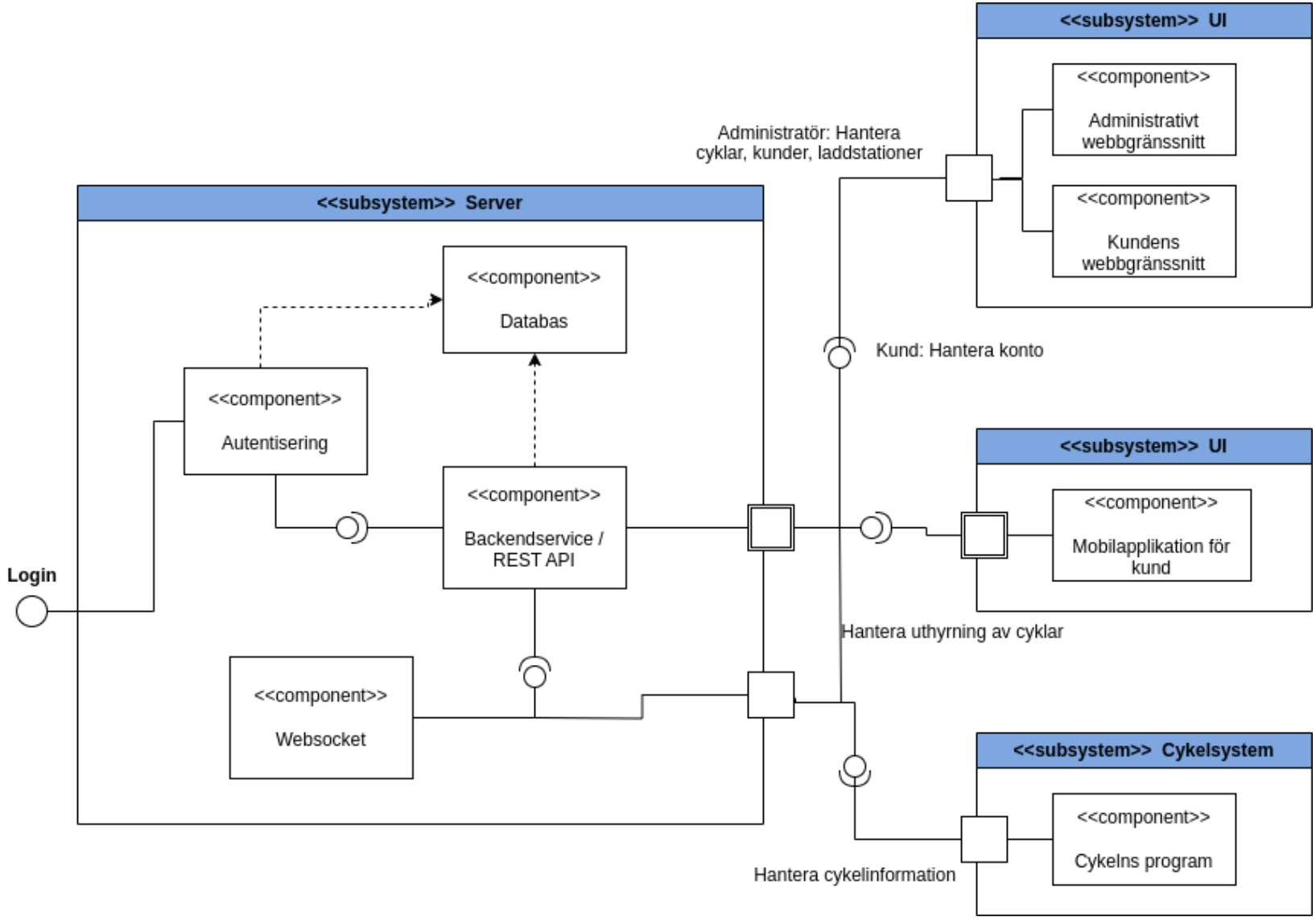

Diagram 2. Komponentdiagram av systemet

Process vy

Systemet består av ett flertal processer som kommunicerar med varandra på flera vägar. Användarna använder sig av ett webbgränssnitt eller en mobilapp för att kommunicera med REST API:t och WebSocket-servern. Administratörer använder sig av ett administratörsgränssnitt som kommunicerar både med REST API:t och WebSocket-servern. REST API:t kommunicerar med databasen, och WebSocket-servern kommunicerar med användarna, administratörerna och elsparkcyklarna. Databasen har enbart kontakt med backend-servern.

Fysisk vy

När det gäller den fysiska implementeringen av systemet kommer databasen att köras på en enskild server. Detsamma gäller även för backend/REST API. För såväl det administrativa webbgränssnittet och kundens webbgränssnitt kommer även dessa att köras på en enskild server.

Kunden kommer utöver webbgränssnittet även ha tillgång till en applikation som kommer att vara anpassad att köras på användarnas smartphones.

Varje elsparkcykel kommer köra sina egna program på enskilda enheter för att på så sätt kunna kommunicera relevant information till systemet som helhet.

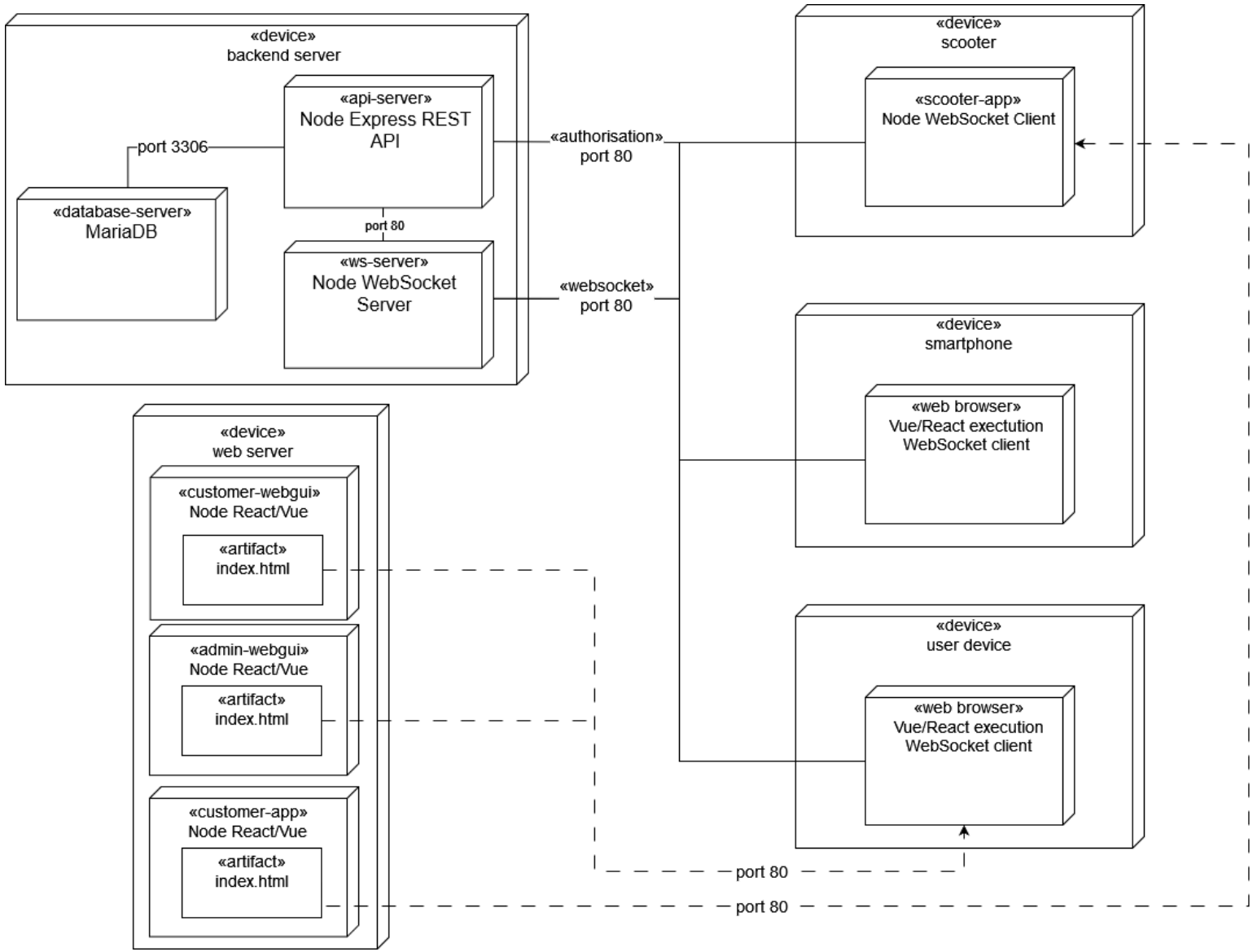

Diagram 3. Driftsättning Diagram

Scenarier

Kunden ska kunna

1. **Logga in och se kundinformation:**
    a. En kund loggar in på webbplatsen och får tillgång till detaljer om sitt konto, inklusive personlig information och aktuell saldo.
2. **Skapa konto och logga in via OAuth:**
    a. En ny kund registrerar sig genom att skapa ett konto på webbplatsen. Kunden kan också logga in med hjälp av OAuth för en säker autentisering.
3. **Visa resehistorik:**
    a. Kunden kan se en detaljerad historik över sina tidigare resor, inklusive information som start- och slutdestination, varaktighet och total kostnad.
4. **Fylla på konto (prepaid) eller månadsbetalning:**
    a. Kunden har möjlighet att antingen fylla på sitt konto med pengar, så kallat prepaid, eller ställa in automatisk månatlig betalning via en betaltjänst.
5. **Hyra en cykel:**
    a. Kunden öppnar appen, väljer en tillgänglig cykel och genomför uthyrning. Systemet reserverar cykeln för kunden under den valda tiden.
6. **Köra cykeln under hyrtiden:**
    a. Under hyrtiden har kunden full åtkomst till den hyrda cykeln och kan använda den för sina resor. Kunden kan också pausa hyrtiden vid behov.
7. **Lämna tillbaka cykeln:**
    a. Efter avslutad resa lämnar kunder tillbaka cykeln på en godkänd parkeringsplats. Systemet registrerar avslutad resa och beräknar kostnaden baserat på hyrtid och eventuella extra avgifter.

Administratör ska kunna

1. **Visa alla städer och cyklar:**
    a. Administratören öppnar det administrativa webbgränssnittet och ser en lista över alla tillgängliga städer och cyklar i systemet.
2. **Hantera laddstationer och cyklar:**
    a. Administratören väljer en specifik stad och ser en översikt över alla laddstationer inom staden. Även antalet cyklar på varje laddstation kan ses och man kan flytta en cykel från en laddstation till en annan.
3. **Hantera accepterade parkeringsplatser:**
    a. Administratören granskar de accepterade parkeringsplatserna i varje stad och kan lägga till eller ta bort platser. Om det behövs, kan de även ändra antalet platser.
4. **Hantera kunder:**
    a. Administratören har möjlighet att se en lista över alla kunder i systemet och kan utföra olika åtgärder som att lägga till en ny kund, uppdatera kundinformation eller ta bort en kund från systemet.
5. **Följa upp cyklarnas position:**
    a. Administratören kan spåra var varje cykel är parkerad i realtid. Det inkluderar information om cykeln är på en laddstation, en accepterad parkeringsplats eller fri parkering.
6. **Hantera resor och avgifter:**
    a. Administratören kan se en lista över alla resor som kunder har gjort. De kan också hantera avgifter för varje resa, inklusive den fasta avgiften, rörlig taxa per tidsenhet och eventuell extra avgift för fri parkering.
7. **Optimera startavgift för fri parkering:**
    a. Om en kund tar en cykel från fri parkering och lämnar den på en definierad parkering, kan användaren se och hantera en reducerad startavgift för den specifika resan.
8. **Översikt över laddstationer och cyklar på karta:**
    a. Administratören får en visuell översikt över alla laddstationer, cyklar och deras aktuella positioner på en karta. Vid behov kan man genomföra åtgärder.

Cykeln ska kunna

1. **Rapportera position och status:**
    a. Cykeln meddelar regelbundet sin position och indikerar om den är i rörelse eller stillastående, inklusive aktuell hastighet.
2. **Stänga av/stoppa cykeln:**
    a. Kunden har möjlighet att stänga av eller stoppa en cykel, vilket förhindrar ytterligare körning.
3. **Starta och köra cykeln:**
    a. När en kund hyr cykeln kan denne starta den och använda den för transport under hyrestiden.
4. **Lämna tillbaka cykeln:**
    a. Kunden kan lämna tillbaka cykeln, avsluta hyrestiden och släppa kontrollen över den
5. **Laddnings varning:**
    a. Cykeln utfärdar en varning när batterinivån är låg och den behöver laddas.
6. **Spara reselogg:**
    a. Cykeln sparar en logg över varje resa, inklusive start- och slutdestination med tid samt kundens information.
7. **Underhållsläge och laddningsstatus**:
    a. När cykeln tas in för underhåll eller laddning markerar den sig som i underhållsläge. Vid laddning på en laddstation är cykeln otillgänglig för uthyrning och en röd lampa indikerar dess otillgänglighet.

Starta elsparkcykel

Nedan är ett sekvensdiagram som beskriver vad som händer när en person startar elsparkcykeln och vad elsparkcykelns programvara gör för att ansluta till systemet.

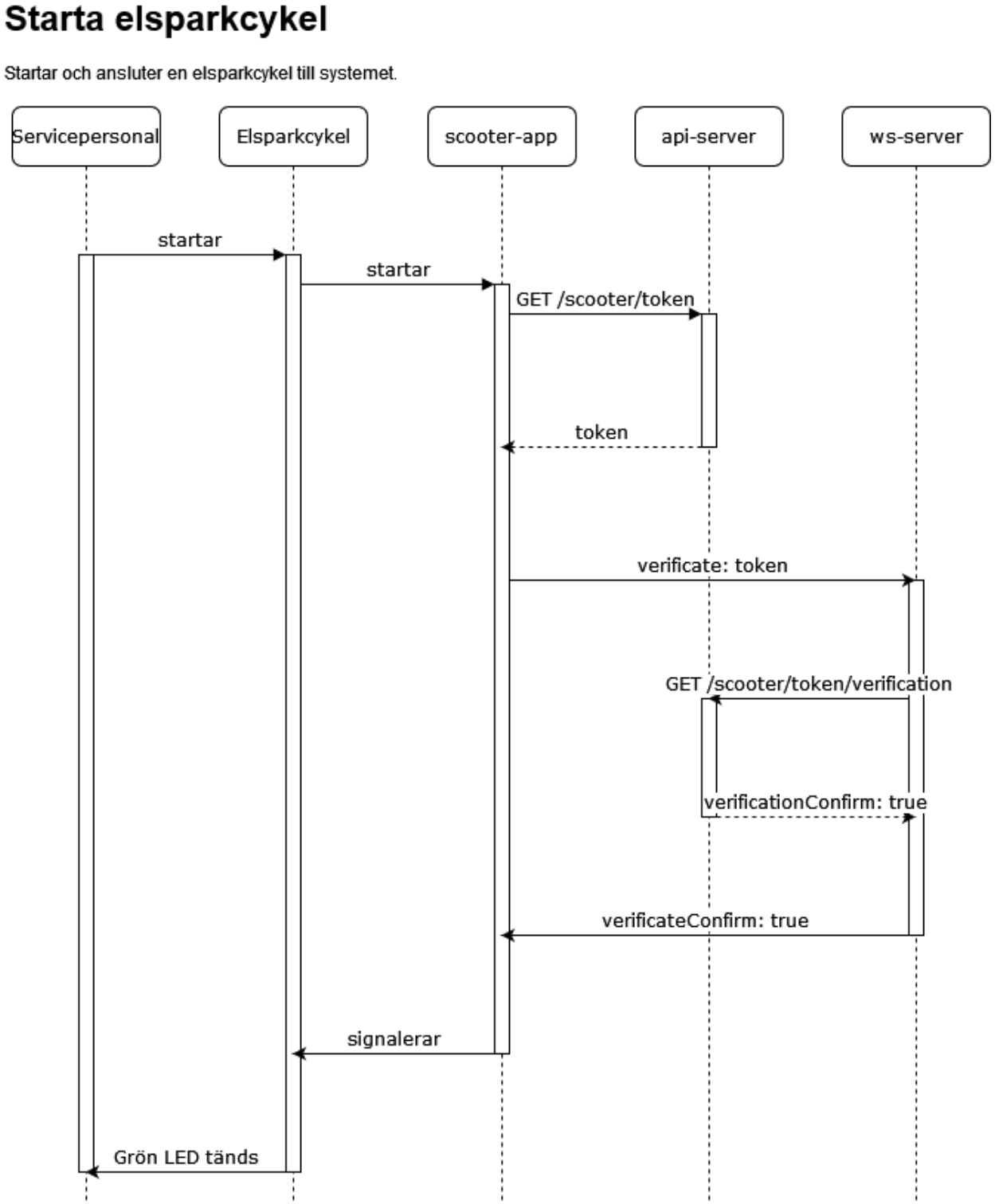

Diagram 4. Översiktligt sekvensdiagram för applikation.

Referenser

[1] Sapra, P. (2019, Mars 26). *Kruchten's 4 + 1 views of Software Design - Software Design Knowledge Categorization Framework*. [Online]. Tillgänglig: https://medium.com/the-mighty-programmer/kruchtens-views-of-software-design-e9088398c592

[2] Krutchten, P. ``Architectural Blueprints - The '4+1' View Model of Software Architecture", *IEEE Software,* vol. 12, issue 6, s. 42-50, Nov. 1995. Tillgänglig: https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

[3] Mark Richards. (2015, Aug. 15). *Software Architecture Patterns.* [Online]. Tillgänglig: https://www.oreilly.com/content/software-architecture-patterns/