Akutplattan – Teknisk dokumentation

Redaktörer:
Lisa Ax, Susanna Dahlgren, Oskar Joelsson, Kim Larsson,
Oscar Magnusson, Daniel Månsson, Olle Renius
Datum: 2017-05-30
Version 1.0

Dokumenthistorik

Version	**Datum**	**Utförda förändringar**	**Utförda av**	**Granskad**
1.0	2017-05-30	Version 1.0	KL, OM, DM	OM, KL

Innehåll

1 Inledning

I den här tekniska dokumentationen beskrivs utformandet av webbapplikationen Akutplattan och dess funktioner. Dokumentet tar upp hur de olika delarna i systemet kommunicerar med varandra samt hur anatomin ser ut som helhet. Vissa implementationsdetaljer gås även genom.

1.1 Bakgrund

Akutplattan är en webbapplikation som kan användas som hjälpmedel vid hjärt-lungräddningssituationer (HLR-situationer). Akutplattan togs fram av en grupp studenter från Linköpings universitet i samband med ett kandidatprojekt och där beställningen av produkten kom från Region Östergötland - Test och Innovation.

1.2 Syfte och mål

Den tekniska dokumentationen behandlar implementationen av applikationen Akutplattan och har som syfte att ge en beskrivning av dess uppbyggnad och funktionalitet. Målet är att underlätta rekonstruktion vid framtida tillfällen och även underhåll av den befintliga produkten.

1.3 Sekretess

Inga personuppgifter som matas in av användaren, vid exempelvis doseringsberäkningen för barn, sparas utan används enbart vi det tillfället. Historiken som visas efter varje avslutat flöde rensas av vårdpersonalen när efterarbetet genomförts för att säkerställa att ingen utomstående kommer information om händelserna i det utspelade HLR-scenario.

2 Teori

Här tas relevant teori upp som används för att förklara olika delar av produktens funktionalitet och implementation.

2.1 Angular

Webbapplikationen är uppbyggd i ramverket Angular. Angular är ett ramverk som underlättar utvecklingen av större webbapplikationer. Ramverket fokuserar på underhåll, struktur och flexibilitet i applikationen, med ett fokus på stöd för både mobila enheter och skrivbordsmiljöer. I Angular sker

arbetet med hjälp av modulära enheter som kallas komponenter. Dessa kan nästlas i varandra och på så sätt skapa hierarkier med utbytbara delar, vilket bidrar till en flexibel arbetsmetodik och återanvändbar kod. Med detta följer också tilläggsverktyg som underlättar exempelvis testning av kod.

I detta dokument antas det att läsaren åtminstone är bekant vid grundläggande koncept inom Angular. För mer info om Angular, se den officiella hemsidan *https://angular.io/*.

3 Hårdvara

Akutplattan är utformad först och främst för att storleksmässigt passa en iPad med en displaystorlek på 9,7 tum, mätt diagonalt, men går att köra på i princip vilken enhet som helst. Den enda skillnaden på andra skärmstorlekar är att upplägget av komponenterna inte kommer överensstämma med det upplägg som används till iPad.

Den indata som Akutplattan tar från användaren består enbart av pekskärmstryckningar eller musklick, om man kör applikationen via en dator utan pekskärm. Denna typ av indata används för att navigera igenom de olika sidor som applikationen är uppbyggd av samt att interagera med komponenter som till exempel knappsatser och checklistor.

4 Produkten

Detta kapitel går igenom systemets uppbyggnad och struktur.

4.1 Arkitekturdesign

Webbapplikationens mjukvaruarkitektur är baserad på en design där det finns en huvudvy som är den sida som presenteras för användaren och flera delvyer som ersätter det primära innehållet i denna huvudvy. Det skulle kunna liknas lite vid en modulbaserad design, där varje delvy utgör en modul som innehåller flera komponenter. Delvyerna i sig är egentligen också komponenter, men som inte har några direkta beroenden till andra delvyer (se kapitel 4.1.1 för undantag). En komponent i sig är helt enkelt den struktur som Angular väljer att representera objekt i ramverket på, dessa tillämpar internt en slags MVC-arkitektur för att särskilja och strukturera koden.

4.1.1 Kommunikation mellan delvyer

För att kommunicera mellan delvyer så används så kallade ”services” i Angular. Dessa kan i ramverket skapas för att innehålla såväl variabler som funktioner, men den viktiga egenskapen att de kan injiceras i andra komponenter. När en service injiceras i två eller flera olika komponenter som inte nödvändigtvis är beroende av varandra så skapas en gemensam instans av denna service. Den gemensamma instansen kan sedan användas för att utbyta data mellan komponenterna, oberoende av om båda komponenterna redan är skapade. Detta innebär alltså att en komponent som förstörs kan föra över data till en ny komponent som skapas vid ett senare skede. Medan detta är ett smidigt sätt att överföra tillstånd och information mellan komponenter som inte är sammankopplade så skapar det också ett beroende i koden, något som är viktigt att tänka på vid vidareutveckling. I webbapplikationen så används dessa services sporadiskt för att överföra information mellan olika delvyer, men också till viss del för att låta komponenter inom delvyer kommunicera med varandra, samt för att möjliggöra loggningsfunktionaliteten (se kapitel 4.2.5).

4.2 Implementation

I detta kapitel beskrivs implementationsdetaljer om ett antal viktiga aspekter hos systemet. Här redogörs på en hög nivå hur komponenter och klasser är relaterade till varandra och vilka filer som är relevanta att undersöka för att få en djupare förståelse för systemet.

4.2.1 App

App (eller appmodule) är den rot-komponent som applikationen kör. Dels innehåller den alla nödvändiga importer för att applikationen ska köras samt all ”routing” som sker i applikationen. Komponenten kallar alltid på en ”header”-komponent som visas överst på varje sida av applikationen. App-komponenten kallar också på ett s.k. ”router-outlet”-element. ”Router-outlet”-elementet hämtar den delvy vars URL-adress används just nu. En lista med alla URL:er finns att hitta i routing-klassen.

- App-klass: ”src/app/app.component.ts”
- Routing-klass: ”src/app/app.routes.ts”
- Header-komponent: ”src/components/header/header.component.*”

4.2.2 HLR

Ett HLR-flöde utgörs av ett antal olika Angular-komponenter: ”hlr.component”, ”hlrflow.component”, ”checklist.component” och ”timer.component”. Det finns också en komponent ”step.component” som beskrivs i detalj i avsnitt 4.2.4. Dessa komponenter utgör en hierarki. Som rot ligger HLR-komponenten, som i sin tur innehåller övriga komponenter.

I HLR-komponenten finns ett antal funktioner för allmän hantering av ett HLR-flöde, exempelvis hantering av ett avslutat flöde. Den mest intressanta logiken finns i HLR-flöde-komponenten, som utgör applikationens huvudfunktion. Här skapas och hanteras de steg som användaren navigerar mellan under en HLR-situation.

Relevanta sökvägar:

- HLR-komponent: ”src/components/hlr/hlr.component.*
- HLR-flöde-komponent: ”src/components/hlr/hlrflow/hlrflow.component.*”
- Checklista-komponent: ”src/components/hlr/checklist/checklist.component.*”
- Timer-komponent: ”src/components/hlr/timer/timer.component.*”

4.2.3 Barn-HLR

Barn-HLR fungerar ungefär som det vanliga HLR-flödet men innehåller några extra steg där emellan. De relevanta komponenterna är ”barnhlr.component”, ”barnhlrsettings.component” och ”keypad.component”. Barnhlr.component är ett förval för den logik som sker i barnhlrsettings.component. Logiken för keypaden finns i keypad.component. Efter att användarens preferenser är satta så går man vidare till ett HLR-flöde (se 4.2.2).

- Barnhlr-komponent: ”src/components/barnhlr/barnhlr.component.*”
- Barnhlrsettings-komponent: ”src/components/barnhlrsettings/barnhlrsettings.component.*”
- Keypad-komponent: ”src/components/hlr/keypad/keypad.component.*”

4.2.4 Step

Ett så kallat steg (step) är benämningen som valdes för ett steg i HLR-flödet. Ett steg representeras av två delar i koden: en Angular-komponent (hlrstep.component) och en TypeScript-klass (Step). Step-klassen är i grund och botten endast en behållare av data för en Step-komponent, medan det

är i Step-komponenten som alla funktioner och operationer finns. Variabler och funktioner finns beskrivna i kommentarer i koden, se källkodsfilerna för mer information.

Relevanta sökvägar:

- Step-klass: ”src/components/hlr/hlrflow/step.ts”
- Step-komponent: ”src/components/hlr/hlrflow/hlrstep/hlrstep.component.ts”

4.2.5 Logg

Loggen utgörs av en komponent ”log.component”. Systemet utnyttjar en service ”logging.service” för att registrera händelser i HLR-flödet till loggen.

Relevanta sökvägar:

- Logg-komponent: ”src/components/log/log.component.*”
- Logg-service: ”src/services/logging.service.ts”

4.2.6 Andningsstopp

Andningsstopp är den sista knappen i huvudmenyn och är en egen klass. I den klassen visas enbart en lokal bild på ett instruktionsblad för andningsstopp. Klassen går att hitta med sökvägen: ”HLRproject/src/components/respiratoryarrest/respiratoryarrest.component.ts”.

5 Driftsättning

Applikationens kod finns tillgänglig på Applikationens kod finns tillgänglig på GitHub (https://github.com/ollerenius/akutplattan). Där finns även instruktioner om hur systemet byggs och startas.

6 Tester

Denna produkt har gått igenom flertalet tester för att säkerställa att all funktionalitet är robust och tilltalande för användarna.

6.1 Användartester

För att säkerställa att alla funktioner som applikationen har är relevanta och dugliga för användning i praktiken har flertalet användartester genomförts. Dessa tester har genomförts tillsammans med utbildad vårdpersonal på Linköpings

universitetssjukhus med olika erfarenhet. Dessa tester skedde i form av iscensatta hjärt- lungräddningssituationer där vårdpersonalen använde sig av Akutplattan som hjälpmedel. Efter dessa tester hölls utvärderingssamtal för att analysera testerna och för att se om någon funktionalitet behövdes läggas till, justeras eller tas bort. Det är dessa tester som har legat till grund för de flesta av applikationens design och funktioner.

6.2 Uthållighetstester

För att säkerställa att Akutplattan håller den kvalité och robusthet som krävs för att tas i bruk i vården. Dessa tester bestod av stresstester för att se om applikationen klarade av att vara igång och köras under betydligt längre perioder än den egentligen är menad för. Tester som Akutplattan har klarat av har bland annat varit:

- Ha ett HLR-flöde igång under 3 timmar för att säkerställa att även de mest extrema tidsspann på en HLR-situation klaras av.
- Vara igång under 24 timmar med periodvis körda HLR-flöden med rimligt tidsspann för att säkerställa att ingen funktionalitet sviktar under längre perioder utan ”vila”.

7 Kodstandard

En kodstandard har tagits fram för att ge all kod i projektet en struktur som är genomgående för alla filer. Det beslutades att endast engelska kommer användas i koden - detta inkluderar namngivning (variabler, funktioner, etc), kommentarer i koden och commit-meddelanden. All kod ska vara väldokumenterad, commit-meddelandena ska vara så pass tydliga att alla i gruppen kan förstå vad som gjorts. Riktlinjer för TypeScript beskrivs av Microsoft.

7.1 Funktions- och variabelstruktur

Alla funktioner utför endast det som sägs att funktionen ska göra. Alla variabel- och funktionsnamn är skrivna i ”camelCase”, med liten bokstav i början. Detta används för att undvika mellanrum eller understreck. Koden är indenterad.

7.2 Kommentarer

Koden är väldokumenterad och det finns JavaDoc för varje funktion eller variabel som behöver en förklaring. Det finns också övriga kommentarer som förklarar delsteg i funktioner.