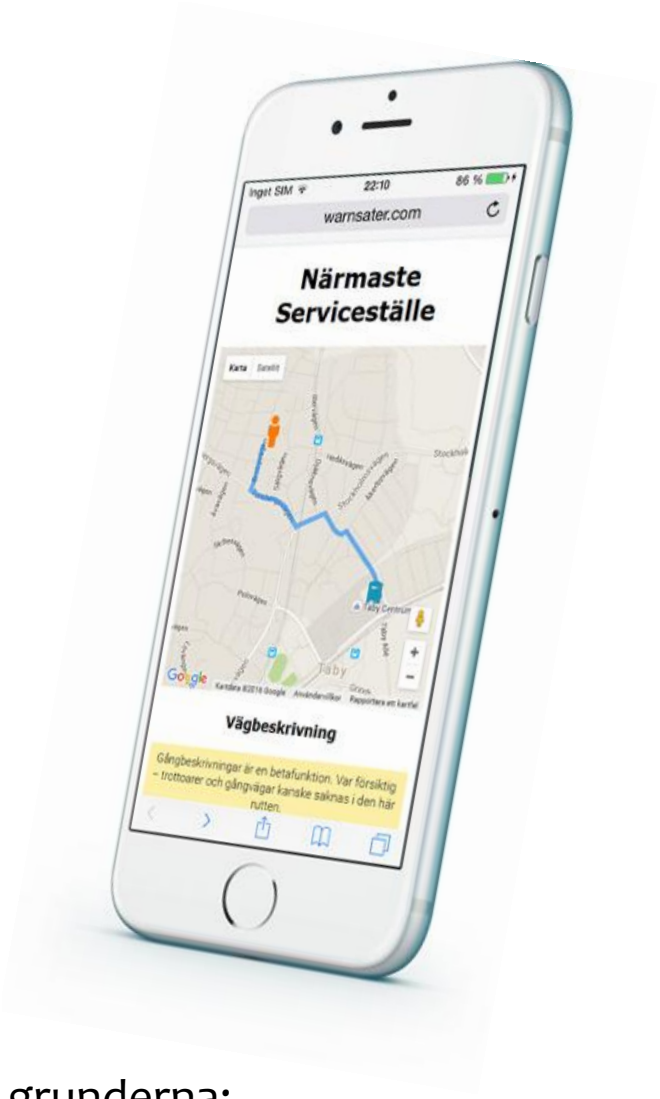
Lär dig programmera en app med PostNords öppna API

av Stefan Wärnsäter



Lär dig grunderna:

- ✓ HTML5, Javascript och CSS
- ✓ Använda PostNords öppna API
- ✓ Använda Google Maps API för att visa kartor och rutter

Inga förkunskaper i programmering krävs!

Lär dig programmera en app med PostNords öppna API

Av Stefan Wärnsäter

Introduktion

I den här övningen som tar ca en timme (eller ett par) att genomföra kommer du att skapa en enkel app, eller snarare en webbapp som bland annat använder PostNords öppna API för att hitta närmaste Serviceställe. Du kommer att använda flera av de vanligaste moderna verktygen som idag används av utvecklare på webbyråer och företag runt om i världen.

> ***Målet med övningen är att skapa en webbapp som på en karta visar var du befinner dig, var PostNords närmaste Servicesställe finns och en vägbeskrivning hur du tar dig dit.***

Det här kommer du att lära dig

Den här övningen består av tio tydliga steg där du lär dig olika moment i programmeringen av din webbapp.

Steg 1 Här sätts grunderna för programmeringsarbetet och du väljer verktyg för din utveckling.
Steg 2 Lär dig programmera en stomme för en webbapp i HTML
Steg 3 Komplettera HTML-koden för din webbapp så att du får med alla nödvändiga delar
Steg 4 Lär dig grunderna i Javascript-programmering och programmera en funktion som tar reda på din geografiska position
Steg 5 Lär dig använda Google Maps API och visa en karta i din webbapp
Steg 6 Lär dig använda PostNords öppna API och hur du tar reda på var närmaste Serviceställe ligger
Steg 7 Knyt ihop säcken och lär dig hur du med Google Maps API kan visa en rutt på kartan och presentera en vägbeskrivning
Steg 8 Överkurs: Pimpa appen – gör den snyggare och responsiv!
Steg 9 Överkurs 2: Mobilanpassa mera
Steg 10 Överkurs 3: Publicera din app

Till din hjälp har du också källkod till varje steg som du kan använda som referens eller ”facit”. Det gör det lättare om du kör fast någonstans längs vägen.

Kommentar

För att hålla formatet på den här övningen på en rimlig nivå så innefattar de ingående övningarna endast grundläggande felhantering! En mer fullständig felhantering kräver ytterligare fördjupning i programmering i Javascript vilket ligger utanför målet med övningen.

Steg 1 – skaffa ett programmeringsverktyg

Programmering innebär att skriva text enligt en strikt struktur som bestäms av det programmeringsspråk man använder. För att bygga en webb-app eller en webbsida så behöver man normalt använda tre olika programmeringsspråk:

- **HTML** som är ett språk som beskriver hur en webbsida ska se ut.
- **CSS** är ett språk som används för att ändra utseendet på delar av innehållet på en webbsida, t.ex. byta färg på en bit text.
- **Javascript** är ett språk för att bygga program som kan ge en webbsida nya funktioner. (Kan jämföras med macro som kan användas i Excel för att t.ex. göra avancerade beräkningar.)

När man skriver programkod så behöver man en editor (textredigerare). Det funkar inte med Word som lägger in mycket annat än själva texten i dokumenten, t.ex. information om teckensnitt, rubriker, o annat som behövs för att styra utseendet på ett dokument.

I den här övningen så väljer vi en textredigerare som riktiga hard-core programmerare brukar välja; **Notepad** (Anteckningar). Den ger ingen som helst hjälp till programmeraren, utan man måste tänka ut och skriva allt, precis allt, precis rätt!

Vanliga utvecklingsmiljöer

Vill man göra det lättare för sig kan man välja en utvecklingsmiljö (program) som ger stöd och hjälp att skriva rätt. ”Eclipse” är exempel på en utvecklingsmiljö som används av många som jobbar med Java, ”Visual Studio” är en annan miljö som används av utvecklare för .NET (Microsoft) och Xcode är vanligt om du utvecklar på Mac.

Förutsättningar

För att den här övningen ska fungera för dig finns det fyra grundförutsättningar.

- Du har tillgång till en dator med operativsystemet Windows.
- Du måste kunna hitta och starta Utforskaren.
- Du måste ha gjort en inställning i Utforskaren så att filnamnens ändelse (efter sista punkten) alltid visas.
- Du måste kunna hitta och starta Anteckningar (Notepad).
- Webbläsare:
    - Chrome fungerar inte för den här övningen. Säkerheten i Chrome medger inte anrop till APIer geolocation när en webbsida laddas från en lokal fil (på t.ex. en PC)
    - Firefox fungerar
    - Internet Explorer fungerar, men inte på en PostNord-dator

Om du inte är hemma på detta, titta då igenom den här guiden: **Använda en dator med Windows för enkel utveckling**.

Steg 2 – Hello World!

Dags att börja koda!

Det första man brukar göra i en ny (programmerings-) miljö och med nya verktyg är att göra enklast möjliga applikation, för att se att just det enklaste fungerar.

Du ska börja med att skapa ”stommen” till webbappen som är en vanlig webbsida skriven i HTML-kod. Du gör det genom att i Windows starta ”Utforskaren” och öppna mappen Dokument. Välj ”Skapa ny mapp” och kalla den ”Min app”.

Dubbelklicka på mappen ”Min app” så att den öppnas. Högerklicka i den nya mappen och välj ”Nytt…” och sedan ”Textdokument” i menyn som visas. Kalla dokumentet ”servicestalle.html” (ändelsen ”.txt” ska tas bort!).

Högerklicka på den nya filen och välj ”Öppna med…” och sedan ”Anteckningar”.

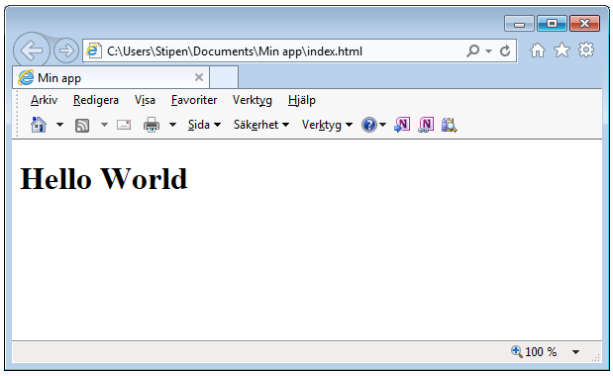
Nu ska du lägga till följande text i filen:

```
<!DOCTYPE  html>

<html lang="sv">
<head>
    <meta charset="utf-8">
    <title>Min app</title>
</head>
<body>
    <h1>Hello World</h1>
</body>
</html>
```

Sedan sparar du (Arkiv/Spara) filen och avslutar Anteckningar (Arkiv/Avsluta).

Grattis! Nu har du redan skapat en webbsida eller om man så vill en webbapp!

Du kan testa den genom att dubbelklicka på ”servicestalle.html”. Den ska då öppnas i din webbläsare och bör se ut ungefär så här:



Kort förklaring av HTML-koden
Innehållet markeras med en ”tagg” som skrivs ut före och efter innehållet, för exempelvis en huvudrubrik används taggen <h1>. 'h' står för heading och '1' för att det är en huvudrubrik, för underrubriker används <h2>, <h3> osv. "Pilarna" (vinkelparanteser) före och efter <taggen> finns där för att markera att det är just en tagg.

De taggar som markerat något innehåll måste "stängas", det innebär att man anger när taggen ska avslutas. För att markera ut en huvudrubrik ser en komplett tagg för en huvudrubrik ut såhär; <h1>Hello World</h1>. Notera att den sista taggen har ett snedstreck /, det anger att det är taggens avslut.

Det finns taggar för olika typer av innehåll och standarder styr vilka taggar som finns och hur de ska användas. Förklaringar till de taggar som används:

- `<!DOCTYPE html>`säger till webbläsaren att det är ett HTML-dokument (webbsida)
- `<html lang="sv">` omger hela webbsidan och anger med 'lang' att det är på svenska (sv)
- `<head>`omger dokumentets egenskaper

- `<meta charset="utf-8">` som ligger inom `<head>` anger att teckenuppsättning UTF-8 ska användas, i den finns Å Ä Ö med. (Notera att denna inte markerar något innehåll, och då inte stängs)
- `<title>` anger dokumentets titel, syns bland annat i webbläsarens sidhuvud eller i sökresultat från sökmotorer
- `<body>` omger dokumentets "kropp" där allt innehåll finns
- `<h1>` omger huvudtiteln som anges inom `<body>`

Steg 3 – HTML-kod för din app

Nu ska HTML-koden fixas till för din app.

Lägg till underrubriker och platshållare

Börja med att öppna filen igen med Anteckningar. Högerklicka på filen och välj ”Öppna med…” och Anteckningar.

Ändra titeln på appen genom att ändra innehållet i <title>-taggen enligt följande:

```
<title>Närmaste Serviceställe</title>
```

Ändra huvudrubriken enligt följande:

```
<h1>Närmaste Serviceställe</h1>
```

Nu ska du lägga till en underrubrik ”Karta” och en platshållare där kartbilden från Google-maps ska läggas. Lägg till följande taggar efter huvudrubriken:

```
<h3>Karta</h3>
<div id="karta"></div>
```

Nu ska du lägga till ytterligare en underrubrik ”Vägbeskrivning” och en platshållare där vägbeskrivningen från Google-maps ska läggas.

```
<h3>Vägbeskrivning</h3>
<div id="beskrivning"></div>
```

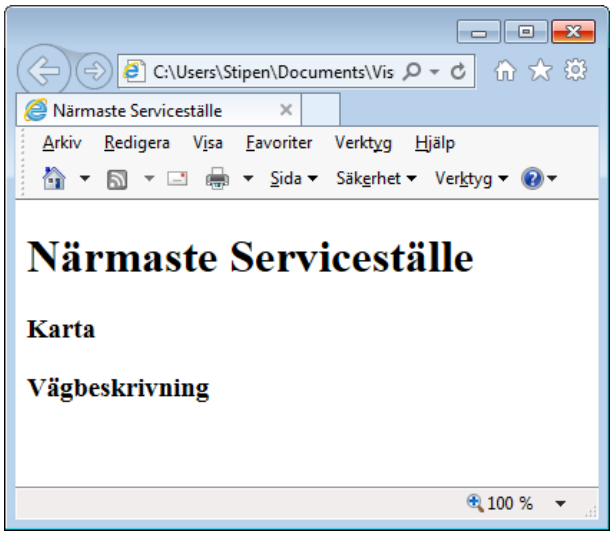
Nu är du nästan klar med HTML-koden och den bör se ut så här:

```
<!DOCTYPE  html>

<html lang="sv">
  <head>
    <meta charset="utf-8">
    <title>Närmaste Serviceställe</title>
  </head>
  <body>
    <h1>Närmaste Serviceställe</h1>
    <h3>Karta</h3>
    <div id="karta"></div>
    <h3>Vägbeskrivning</h3>
    <div id="beskrivning"></div>
  </body>
</html>
```

Testa

Medan man utvecklar kod är det viktigt att hela tiden testa de förändringar som görs så att man är säker på att resultatet blir som förväntat.

Spara dina ändringar i **Anteckningar** och välj sedan igen att öppna filen *”servicestalle.html”* i webbläsaren genom att dubbelklicka på den (eller växla till webbläsaren med Alt-Tab och tryck funktionstangent F5 så att sidan laddas om). Du bör då se ungefär följande:



Koppla sidan till en stilmall

För att förändra utseendet på olika delar av en webbsida används något som kallas Cascading Style Sheets eller CSS. Styling är något man brukar vänta med till dess man är klar med sidans innehåll och funktionalitet. Men nu är det läge att koppla webbappen till en stilmall som du sedan kan återkomma till för att justera utseendet på appen.

Lägg till följande inom `<head>`-taggen efter `</title>`-taggen:

```
<link href="servicestalle.css" rel="stylesheet" />
```

Skapa sedan filen *”servcestalle.css”* och öppna den i Anteckningar.

> I Utforskaren, högerklicka i mappen med din fil *”servicestalle.html”* och välj ”Nytt…” och sedan ”Textdokument” i menyn som visas. Kalla dokumentet *”servicestalle.css”* (ändelsen ”.txt” ska tas bort!). Högerklicka på den nya filen och välj ”Öppna med…” och sedan ”Anteckningar”.

Lägg sedan till följande i den filen:

```
/* servicestalle.css – Stilmall för webbappen servicestalle.html */
```

> ***Förklaring till koden:***
> I CSS-koden deklarerar man hur olika element på webbsidan ska se ut.
> Kommentarer i CSS-koden inleds med `/*` och avslutas med `*/` . Nu lägger du bara in en kommentar som förklarar vad stilmallen används till!

Spara *”servcestalle.css”* och avsluta Anteckningar.

Koppla sidan till programkod

Nu återstår att koppla sidan till den programkod som behövs för att den ska få den funktionalitet som önskas. Sidan ska ju på en karta visa var du befinner dig, var närmaste Serviceställe finns samt ge en vägbeskrivning. Platshållare för karta och vägbeskrivning är redan fixade, no ska koden ordnas som fyller dem med innehåll!

För att lyckas med det här så behöver du dels använda kod som Google har utvecklat för att hantera kartor på webbsidor och dels behöver du skriva lite kod själv som knyter ihop det hela!

Nu ska du skapa en referens i ”servicestalle.html” till den kod som du ska skriva. Det gör du genom att lägga in följande omedelbart mellan `<div id="beskrivning"></div>` och `</body>`:

```
<script src="servicestalle.js"></script>
```

`<script>`-taggen används för att inkludera programkod till en webbapp. Koden kan antingen skrivas in i samma fil som HTML-koden innanför taggen: `<script> programkod </script>`. Med det anses inte som ”god programmeringssed”. Man strävar efter att hålla isär olika delar av programmet som gör olika saker. Då blir det lättare att underhålla applikationen. Därför används parametern `src` för att peka ut den fil som innehåller programkoden.

I nästa avsnitt ska du ta dig an ”riktig programmering” och skapa innehållet i filen *”servicestalle.js”* !

Steg 4 – Javascript programmering

Det du nu ska programmera ska göras i programspråket Javascript. Det är ett språk som används för att komplettera webbsidor och webbappar med funktionalitet som kan köras direkt i webbläsaren.

Börja med att skapa filen *”servcestalle.js”* och öppna den i Anteckningar.

> I Utforskaren, högerklicka i mappen med din fil *”servicestalle.html”* och välj ”Nytt…” och sedan ”Textdokument” i menyn som visas. Kalla dokumentet *”servicestalle.js”* (ändelsen ”.txt” ska tas bort!). Högerklicka på den nya filen och välj ”Öppna med…” och sedan ”Anteckningar”.

Ta reda på dina koordinater

Det första som programmet ska göra är att ta reda på de geografiska koordinaterna där du befinner dig, eller snarare – där webbläsaren befinner sig som kör programkoden.

Nu kommer du att använda en **funktion** som finns inbyggd i alla moderna webbläsare som heter getCurrentPosition.

När man använder den här typen av funktioner så behöver man hitta information om dem så att man kan läsa på hur den ska användas. Webbläsarnas inbyggda funktioner finns väl beskrivna på många ställen på Internet. Genom att googla på ”getCurrentPosition” hittar man bland annat den här sidan som ger nödvändig information: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation/getCurrentPosition

Funktionen ska anropas med två **parametrar** som i sin tur är funktioner som du ska skriva. Den första parametern är en funktion som anropas av getCurrentPosition om den lyckade fastställa positionen och den andra parametern är en funktion som anropas om det av någon anledning misslyckas.

Ett anrop av getCurrentPosition kan se ut så här:

```
navigator.geolocation.getCurrentPosition(positioneringOK, positioneringFel);
```

> ***Förklaring:***
> **navigator** talar om att det är webbläsaren som ska tillhandahålla funktionen.
> **geolocation** anger namnet på det API i webbläsaren som tillhandahåller funktionen.
> **positioneringOK** anger namnet på den funktion som ska anropas om det gick bra.
> **positioneringFel** anger namnet på den funktion som ska anropas om något blev fel.

Deklarera en funktion som hanterar fallen när positionen kan fastställas

I Javascript skrivs – eller som det kallas på programmerarspråk ”deklareras” – en funktion så här:

```
function funktionensNamn(parameternamn1, parameternamn2, …)
{
    // Här skrivs koden som funktionen ska utföra
    // Dubbelslash ’//’ används för kommentarer
};
```

I Javascript används ”krullparenteser” { och } för att hålla samman flera programrader – eller som det heter på programmerarspråk, programsatser eller på engelska ”statements”. Semikolon används för att avsluta en programsats.

En geografisk position på jordklotet bestäms av två koordinater, latitud och longitud. När getCurrentPosition lyckas så anropar den funktion som man anger som första parameter i anropet. Den skickar då med en parameter som anger den uppmätta positionens latitud och longitud.

Det du ska göra nu är att skriva den funktionen som tar emot det uppmätta resultatet och sparar undan det så att du kan använda det senare när du ska visa positionen på en karta!

Skriv in följande kod i filen ”servicestalle.js” som du har öppnat i Anteckningar:

```
//Variabler för att spara uppmätt position
var minLatiud = 0;
var minLongitud = 0;

function positioneringOK(position)
{
    //Positionering har lyckats! Spara latitud och longitud i variabler
    minLatitud = position.coords.latitude;
    minLongitud = position.coords.longitude;

    //Enbart för test: Meddela latitud och longitud på skärmen
    alert("Latitud: " + minLatitud + " Longitud: " + minLongitud);
};
```

> ***Förklaring:***
>
> **minLatitud** och **minLongitud** är variabler som du skapar för att spara värden.
> **positioneringOK** är den funktion som du deklarerar som anropas av getCurrentPosition om allt går bra.
> **position** är en parameter som getCurrentPosition skickar med i anropet till positioneringOK.
> **position** innehåller de uppmätta koordinaterna som kopieras till variablerna du deklarerade.
> **alert** är en funktion som pausar programmet och visar ett textmeddelande på skärmen. Meddelandet som ska visas skickar man med som parameter till alert -> alert(”meddelande”);

Som du ser innehåller koden rikligt med kommentarer. Ta för vana att kommentera rikligt! Det underlättar när du senare ska underhålla koden, rätta fel eller lägga till nya funktioner.

Deklarera en funktion som hanterar när något går fel

En av de svåraste uppgifterna när man programmerar är att förutse och hantera allt som kan gå fel. Och tyvärr är det många saker som kan gå fel.

Du programmerar nu en webbsida/webbapp som kan köras av i princip vilken webbläsare som helst i vilket device som helst; t.ex. Safari i en iPad, Google Chrome i en PC eller Internet Explorer i en gammal Nokia telefon. Om något går fel så bör programmet försöka lösa problemet själv eller ge ett så ”intelligent” besked som möjligt till användaren om vad som gick fel och vad användaren kan göra. Du som programmerare är ansvarig för att programmet inte kan krascha och ge obegripliga meddelanden till användaren!

Enligt beskrivningen kan getCurrentPosition råka ut för tre felsituationer:

1. Användaren har inte tillåtit applikationen att ta reda på aktuell position. Det här är en integritetsfunktion för alla program som vill ta reda på din position. Användaren måste tillåta det!
2. Ett internt fel (i det aktuella devicet) gjorde att positionen inte kunde fastställas.
3. Tidsgränsen för att ta reda på positionen överskreds (timeout)

Vid felsituation 2 och 3 kan användaren uppmanas om att försöka igen lite senare. Möjligen var felet tillfälligt.

Skriv in följande kod i filen ”servicestalle.js” för att deklarera funktionen som hanterar eventuella fel när den aktuella positionen ska fastställas:

```
function positioneringFel(fel)
{
    if (fel.code === 1)
    { // felkod 1 betyder att användaren inte tillät programmet att ta reda på aktuell postion
        alert("Du måste tillåta applikationen att läsa din aktuella positon om den ska fungera!");
    };
    if (fel.code === 2)
    { // felkod 2 betyder att för tillfället var omöjligt att läsa den aktuella positionen
        alert("det var för tillfället omöjligt att läsa din aktuella position. Försök igen senare.");
    };
    if (fel.code === 3)
    { // felkod 3 betyder att det tog för lång tid (timeout) att ta reda på aktuell position
        alert("Det tog för lång tid att läsa din position. Försök igen senare.");
    };
};
```

> ***Förklaring till koden:***
>
> Funktionen tolkar parametern fel och ger relevanta felmeddelanden beroende på felkod.
> **positioneringFel** är den funktion som du deklarerar som anropas av getCurrentPosition om något går fel.
> **fel** är den parameter som getCurrentPosition skickar med i anropet till positioneringFel.
> **fel** har attributet **code** som innehåller den felkod som genererades.
> **If (…)** är en Javascript programsats som testar det villkor som står inom parentes. Om testet är SANT så körs kodavsnittet som står inom ”krullparenteser” efter. Om testet är FALSKT så hoppas kodavsnittet över.
> **===** ska läsas som ”är lika med”.

Knyta ihop säcken

Nu har du fixat grundförutsättningarna för att hantera resultatet av ett anrop till getCurrentPosition. Du har variabler för att spara latitud och longitud och du har funktioner som kan hantera om anropet går bra eller om det blir fel. Nu återstår bara själva anropet!

Nu måste själva anropet göras med viss försiktighet. Du kan inte förutsätta att alla webbläsare har stöd för APIet **geolocation**. Alla moderna webbläsare ska ha det, men ju äldre webbläsaren är ökar sannolikheten för att APIet inte stöds. Så det du behöver göra innan anropet av getCurrentPosition är att testa om webbläsaren stöder detta och hantera ifall den inte gör det.

Att testa om en **geolocation** finns tillgängligt är ganska enkelt med följande **if**-sats:

```
if ("geolocation" in navigator)
{
    // Ja, geolocation är tillgänglig
    // Anropa funktionen getCurrentPostion
}
else
{
```

```
    // Nej, geolocation ÄR INTE tillgänglig
    // Visa felmeddelande
};
```

Nu ska du i Anteckningar komplettera koden med en ny funktion **minPosition** förpacka **if**-satsen ovan och komplettera med anrop till **getCurrentPosition** samt att visa felmeddelande med ett anrop till **alert**.

```
function minPosition()
{
    //Kolla om webbläsaren kan läsa aktuell geografisk position
    if ("geolocation" in navigator)
    {
        // Ja, geolocation är tillgänglig
        // Anropa funktionen getCurrentPostion med parametrar
        //   positioneringOK som hanterar om anropet går bra och
        //   positioneringFel som hanterar om det blev något fel
        navigator.geolocation.getCurrentPosition(positioneringOK, positioneringFel);
    }
    else
    {
        // Nej, geolocation ÄR INTE tillgänglig
        // Visa felmeddelande
        alert("Din webbläsare har inte förmåga att läsa din position.");
    };
};
```

Nästan klar! Nu har du deklarerat variabler och funktioner som är redo att användas. Det som återstår är ett anrop till den sista funktionen som du deklarerade, **minPosition**. Det gör du genom att lägga till följande rader sist i javascript-filen:

```
//Ta reda på positionen genom att anropa minPosition()
minPosition();
```

Spara nu filen **”servicestalle.js”** i Anteckningar. Den bör i sin helhet se ut så här med de steg du gått igenom ovan:

```
//Variabler för att spara aktuell position
var minLatitud = 0;
var minLongitud = 0;

function positioneringOK(position)
{
    //Positionering har lyckats! Spara latitud och longitud i minPosition
    minLatitud = position.coords.latitude;
    minLongitud = position.coords.longitude;

    //Flagga att aktuell position är klar
    positionOK = true;

    //Enbart för test: Meddela latitud och longitud på skärmen
    alert("Latitud: " + minLatitud + " Longitud: " + minLongitud);
};

function positioneringFel(fel)
{
    if (fel.code === 1)
    { // felkod 1 betyder att användaren inte tillät programmet att ta reda på aktuell postion
        alert("Du måste tillåta applikationen att läsa din aktuella positon om den ska fungera!");
    }
    if (fel.code === 2)
    { // felkod 2 betyder att för tillfället var omöjligt att läsa den aktuella positionen
        alert("det var för tillfället omöjligt att läsa din aktuella position. Försök igen senare.");
    }
    if (fel.code === 3)
```

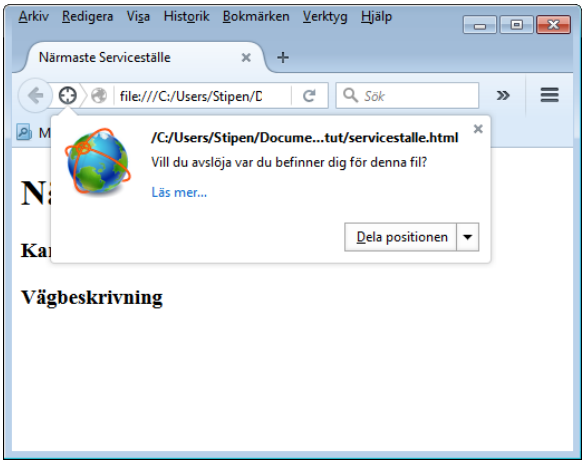
```
    { // felkod 3 betyder att det tog för lång tid (timeout) att ta reda på aktuell position
        alert("Det tog för lång tid att läsa din position. Försök igen senare.");
    }
};

function minPosition()
{
    //Kolla om webbläsaren kan läsa aktuell geografisk position
    if ("geolocation" in navigator)
    {
        // Ja, geolocation är tillgänglig
        // Anropa funktionen getCurrentPostion med parametrar
        //   positioneringOK som hanterar om anropet går bra och
        //   positioneringFel som hanterar om det blev något fel
        navigator.geolocation.getCurrentPosition(positioneringOK, positioneringFel);
    }
    else
    {
        // Nej, geolocation ÄR INTE tillgänglig
        // Visa felmeddelande
        alert("Din webbläsare har inte förmåga att läsa din position.");
    };
};

//Ta reda på positionen genom att anropa minPosition()
minPosition();
```

Testa

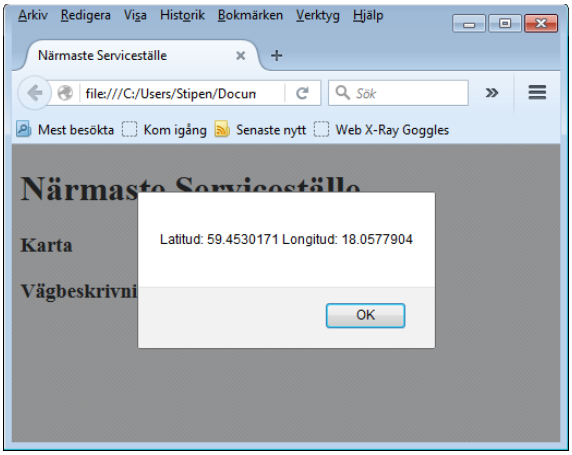
Nu är det dags att testa igen! Det här är kul ☺!

Du gör det genom att dubbelklicka på filen ”servicestalle.html” så att den öppnas i webbläsaren.

Om allt går som förväntas så ska webbsidan först visa en ”popup” där får tillåta applikationen att läsa din geografiska position.



Och sedan, när du gjort det, ska den visa en popup med ”Latitud: xxx Longitud: yyy”.



So far so good! Nu är det dags att visa positionen på en karta!

Steg 5 – Visa en karta

Nu behöver du koppla in stödet för kartor från Google. Googles instruktion för detta hittar du på https://developers.google.com/maps/documentation/javascript/tutorial#Loading_the_Maps_API Där kan man bland annat läsa att den HTML-tagg där kartan ska visas måste ha en definierad storlek för att kartan ska visas korrekt. I HTML-filen servicestalle.html har du skrivit in taggen `<div id="karta"></div>` där kartan ska visas. Nu ska du lägga till en stilmall till webbsidan där storleken på den taggen ska definieras.

Koppla sidan till en stilmall

Sidan som du skapat är ju ännu så länge inte så vacker.

För att förändra utseendet på olika delar av en webbsida används något som kallas Cascading Style Sheets eller CSS. Styling är något man brukar vänta med till dess man är klar med sidans innehåll och funktionalitet. Men nu måste du definiera storleken på ”kart-taggen”.

Börja med att öppna filen *”servicestalle.html”* och lägg till följande inom `<head>`-taggen efter `</title>`-taggen:

```
<link href="servicestalle.css" rel="stylesheet" />
```

Spara HTML-filen och avsluta Anteckningar.

Skapa sedan filen *”servcestalle.css”* och öppna den i Anteckningar.

> I Utforskaren, högerklicka på en tom del i mappen med din fil *”servicestalle.html”* och välj ”Nytt…” och sedan ”Textdokument” i menyn som visas. Kalla dokumentet *”servicestalle.css”* (ändelsen ”.txt” ska tas bort!). Högerklicka på den nya filen och välj ”Öppna med…” och sedan ”Anteckningar”.

Lägg sedan till följande i den filen:

```
/* servicestalle.css - Stilmall för webbappen servicestalle.html */

#karta
{
    width: 400px;
    height: 400px;
}
```

> ***Förklaring till koden:***
> I CSS-koden deklarerar man hur olika element på webbsidan ska se ut.
> Det gör man genom att först tala om vilket eller vilka element som ska få ett utseende och sedan, inom ”krullparenteser” listar man de egenskaper som ska få ett utseende definierat.
> I det här fallet ska elementet som har attributet **id=”karta”** få en bestämd bredd och höjd.
> I CSS identifieras en tagg med ett bestämt id med ett **#**-tecken följt av identiteten, i det här fallet **#karta**.
> Här sätter du storleken på kartan så att den blir 400 x 400 pixel, genom att ange `width: 400px;` och `height: 400px;`

Rita kartan

Nu behöver du koppla in stödet för kartor från Google. Googles instruktion för detta hittar du på https://developers.google.com/maps/documentation/javascript/tutorial#Loading_the_Maps_API där de visar följande exempel:

```
<script async defer
      src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&callback=initMap">
</script>
```

För att använda Google API så behöver man registrera sig som utvecklare och få en så kallad API-nyckel (***YOUR_API_KEY*** i exemplet ovan). API-nyckeln är förknippad med begränsningar av hur mycket man kan anropa APIet. Det är gratis för upp till 25 000 anrop, sedan måste man börja betala.

För den här övningen kan du använda en redan existerande API-nyckel som inte kostar något.

Öppna filen *"servicestalle.html"* i **Anteckningar** igen och börja med att lägga till en koppling till Google Maps Javascript API genom att lägga till följande rad före avslutnings-taggen `</body>` :

```
<script
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDeTQyrXUcc6UrYFvmY0pz2MWSVMCz9X5E&callback=googl
eMapsLaddat"></script>
```

Här är API-nyckel inlagd och i slutet av kommandot som laddar Google APIet ser du parametern **callback=googleMapsLaddade**. Den talar om för Google Maps APIet att när den har laddats färdigt så ska APIet anropa funktionen **googleMapsLaddat**.

I och med att Google Maps API laddas över Internet och att det är relativt mycket och komplicerad kod som ska laddas och förberedas så kan det ta lite tid. Då är det viktigt att ditt program inte försöker anropa Google Maps API innan det är laddat. Den här typen av utmaningar är ganska vanliga när man skapar webbapplikationer eftersom att dels är uppkoppling och hastighet över Internet okänd och dels är kapaciteten i det device som använder applikationen okänd.

Det du nu behöver göra är att skapa en funktion med namnet googleMapsLaddat som tar emot anropet från Google Maps API när det är laddat. Funktionen ska då visa en karta med din position markerad i centrum av kartan.

Först några förberedelser

Nu behöver du "kommentera bort" raden vi använde för att testa att **getCurrentPosition** fungerade. Det var **alert**-satsen i funktionen **positioneringOK**. Nu vet du att den fungerar!

Sätt en "dubbelslash" först på den raden så blir det en kommentar. Så här:

```
// alert("Latitud: " + minLatitud + " Longitud: " + minLongitud);
```

Sedan ska du också lägga till variabeln **karta** i början av filen efter deklarationen av **minLongitud**, skriv in följande:

```
//Variabel för att spara referens till den karta som ritas (behövs för Google Maps API)
var karta;
```

Nu dags att skapa kartan

Grunden för att visa och använda kartor Googles kartor fås genom den del av Google Maps API som kallas just **Map**-funktionen (eller klassen). Google Maps API innehåller flera olika delar som är specialiserade på olika saker, t.ex. **Directions** för att ta fram vägbeskrivningar, **Elevation** för att beräkna höjdkurvor, **StreetView** för att visa gatubilder, mm.

Nu ska du som sagt använda Map-funktionen och om du vill läsa mer om detaljerna kring den och förstå vad den kan göra så finns mycket att läsa om den, med många kodexempel, här https://developers.google.com/maps/documentation/javascript/tutorial och här https://developers.google.com/maps/documentation/javascript/examples/ och här https://developers.google.com/maps/documentation/javascript/3.exp/reference

För att rita din första karta behöver du göra några inställningar av hur kartan ska visas och tala om var på webbsidan den ska visas. Det kan du göra genom att lägga till följande kod i slutet av ”servicestalle.js”:

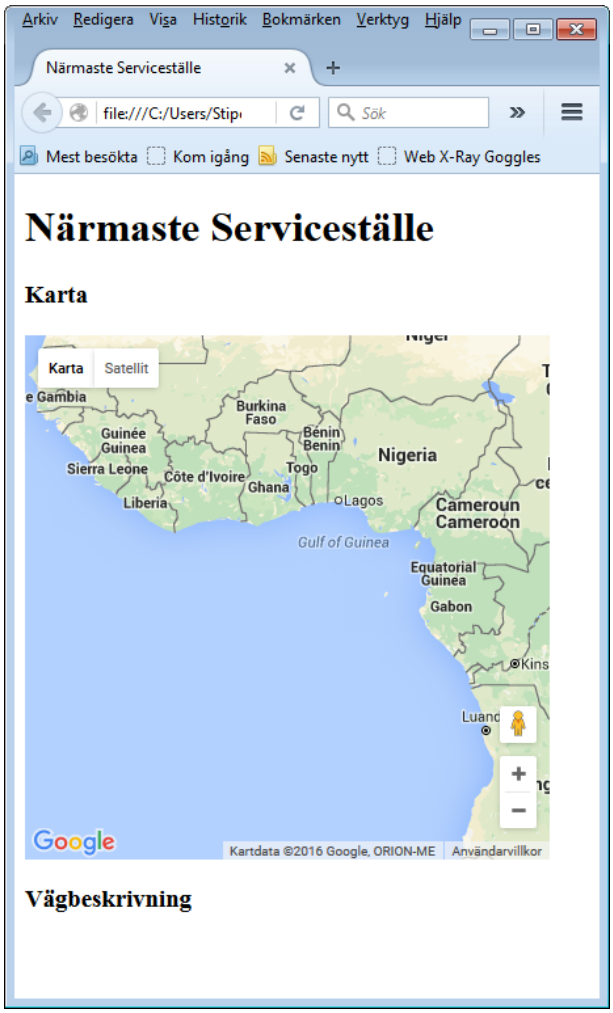
```
function googleMapsLaddat()
{
    // Här ska vi använda variabeln karta som deklarerats i början av filen (global variabel)
    // för att lagra en referens till den karta som Google Maps API ritar

    // Variabel för inställningar som bestämmer hur kartan visas
    var kartOptioner = { zoom: 4, center: { lat: minLatiud, lng: minLongitud } };

    // Variabel som pekar ut det element på webbsidan som är platshållare för kartan
    var kartPlats = document.getElementById("karta");

    // Ladda kartan på sin plats med kartOptioner
    karta = new google.maps.Map(kartPlats, kartOptioner);
};
```

> ***Förklaring till koden:***
>
> Funktionen fungerar som ”callback-funktion” för laddningen av Google Maps API och anropas när laddningen är klar.
> Variabeln **karta** skapas för att hålla en referens till den karta som skapas.
> Variabeln **kartOptioner** skapas för att ”ställa in” hur kartan ska visas:
> **zoom** anger hur inzoomad kartan ska vara
> **center** anger via **lat** och **lng**, latitud och longitud för centrum av kartbilden
> Här sätter du värdena till minLatitud och minLongitud som du redan har fixat sedan tidigare med anropet till **getCurrentPosition**.
> Variabeln kartPlats används för att peka ut ett element (en tagg) på HTML-sidan där kartan ska visas. För att hitta den ”platsen” används funktionen **document.getElementById(”karta”)** som är kopplad till aktuell webbsida (document) och som söker fram den tagg som har attributet id=”karta”.
> Till sist så anropas Google Map APIets funktion **Map** (**google.maps.Map**) med **kartPlats** och **kartOptioner** som parametrar. Då förväntas APIet rita en karta på platsen där du satte taggen `<div id="karta"></div>` (efter underrubriken ”Karta”) som är centrerad kring den lästa positionen och med zoomnivå 4.

Testa

Spara filen *”servicestalle.js”* och dubbelklicka på *”servicestalle.html”* så att den öppnas i webbläsaren.

Efter att du godkänt att din geografiska position läsas bör du se ungefär följande:



Ooops! Hur kunde det bli så här? Du angav ju att kartan skulle centreras till **minLatitud** och **minLongitud** som du hämtat via **getCurrentPosition**!

Det som händer här är ett timing-problem eller ”race condition”!

Kartan ritas innan getCurrentPosition hunnit räkna fram latitud och longitud. Kartans centrum sätts då till den position där dessa värden är 0, en punkt utmed ekvatorn (latitud = 0) som har en lodlinje som skär genom Engelska Greewich (longitud = 0) vilket råkar vara en punkt i havet 611 km söder om Ghana och 1078 km väster om Gabon.

Hur kan man fixa det här då? Kartan kan inte ritas innan ”min position” är klar.

Det du behöver göra nu är att definiera en ”semafor” eller en flagga, som hissas när **getCurrentPosition** har lyckats. Då kan funktionen **googleMapsLaddat** som ritar kartan kolla flaggan och om positionen inte är klar avvakta med att rita kartan.

Fixa semaforen positionOK

Lägg till en variabel i början av ”servicestalle.js” som du sätter till värdet false enligt följande:

```
//Variabel som används som semafor för att flagga när aktuell position är klar
var positionOK = false;
```

Och sedan, i funktionen **positioneringOK** efter att du satt **minLongitud** och **minLatitud** till den aktuella positionen så sätter du variabeln **positionOK** till true – du ”hissar flaggan” – enligt följande:

```
    //Flagga att aktuell position är klar
    positionOK = true;
```

Sådär nu finns det en flagga att kolla om det finns en aktuell position klar att använda.

Nu behöver du komplettera funktionen **googleMapsLaddat** så att den

1. Testar **positionOK**
2. Om den är `true` -> gå vidare och rita karta
3. Om den är `false` - > vänta en liten stund och anropa **googleMapsLaddat** igen

För att göra detta behöver du justera koden i **googleMapsLaddat** enligt följande:

```
function googleMapsLaddat()
{
    if (positionOK)
    {
        // Här ska vi använda variabeln karta som deklareats i början av filen (global variabel)
        // för att lagra en referens till den karta som Google Maps API ritar

        // Variabel för inställningar som bestämmer hur kartan visas
        var kartOptioner = { zoom: 4, center: { lat: minLatiud, lng: minLongitud } };

        // Variabel som pekar ut det element på webbsidan som är platshållare för kartan
        var kartPlats = document.getElementById("karta");

        // Ladda kartan på sin plats med kartOptioner
        karta = new google.maps.Map(kartPlats, kartOptioner);
    }
    else
    {
        // Anropa den här funktionen igen efter 500 millisekunder
        setTimeout(googleMapsLaddat, 500);
    };
};
```

> ***Förklaring till koden:***
>
> Med satsen if (positionOK) testas om positionOK är true eller false.
> Om den är `true` så körs koden i blocket mellan ”krullparenterserna” direkt efter.
> Om den är `false` så körs koden mellan ”krullparenteserna” efter `else`.
> Funktionen setTimeout är en standardfunktion i webbläsaren som anropar funktionen som anges i första parametern efter en fördröjning på det antal millisekunder som anges i den andra parametern.

Testa igen

Spara filen ”servicestalle.js” och dubbelklicka på den så att den öppnas i webbläsaren.

Efter att du godkänt att din geografiska position läsas bör du se ungefär följande:

Det blev ju lite bättre!

Nu ska du putsa lite på det här.

Dels ska du zooma in lite i kartan. Det gör du genom att ändra värdet för **zoom** för variabeln **kartOptioner**. Testa att ändra till något annat än 4, spara Javascript filen och öppna HTML-filen igen i webbläsaren. Prova dig fram till lämpligt värde på zoomen!

Sätt ut en markör på kartan där du befinner dig

Kartan är nu centrerad till platsen du befinner dig. Men det vore ju fiffigt om det sitter en markör där för att ge lite bättre exakthet!

Det finns en funktion för det här i Google Maps API som heter **Marker**. Den ska du använda genom att skriva in följande i funktionen **googleMapsLaddat** efter att du laddat in kartan:

```
// Variabel för inställningar av markör
var here = { position: { lat: minLatitud, lng: minLongitud }, map: karta, title: 'Här är jag!'};

// Sätt en markör på min plats
var marker = new google.maps.Marker(here);
```

> ***Förklaring till koden:***
>
> Först skapas variabeln here för att beskriva hur markören ska positioneras och se ut.
>
> **position** anger koordinaterna för var markören ska placeras. Här används positionen från **getCurrentPosition**.

map anger på vilken karta markören ska sättas (man kan ha flera kartor på en sida). Här används variabeln **karta** som pekar på din karta.
title anges för att visa en text när man håller muspekaren över markören.
Till sist skapas en ny markör genom anropet till **google.maps.Marker(here)**. Resultatet visas på kartan och en referens till markören sparas i variabeln **marker**.

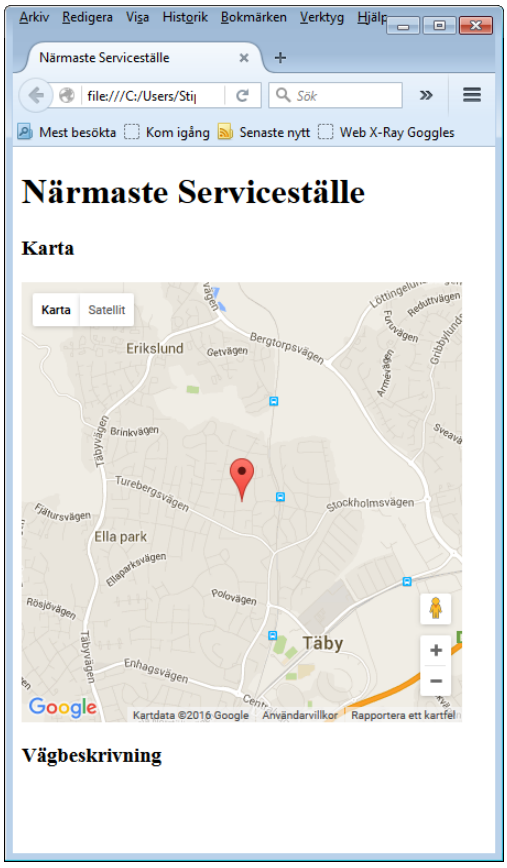
Hela funktionen **googleMapsLaddat** ska då se ut ungefär så här:

```
function googleMapsLaddat()
{
    if (positionOK)
    {
        // Här ska vi använda variabeln karta som deklareats i början av filen (global variabel)
        // för att lagra en referens till den karta som Google Maps API ritar

        // Variabel för inställningar som bestämmer hur kartan visas
        var kartOptioner = { zoom: 13, center: { lat: minLatitud, lng: minLongitud } };

        // Variabel som pekar ut det element på webbsidan som är platshållare för kartan
        var kartPlats = document.getElementById("karta");

        // Ladda kartan på sin plats med kartOptioner
        karta = new google.maps.Map(kartPlats, kartOptioner);

        // Variabel för inställningar av markör
        var here = { position: { lat: minLatitud, lng: minLongitud }, map: karta, title: 'Här är jag!'
};
        // Sätt en markör på min plats
        var marker = new google.maps.Marker(here);
    }
    else
    {
        // Anropa den här funktionen igen efter 500 millisekunder
        setTimeout(googleMapsLaddat, 500);
    };
};
```

Testa igen

Spara filen *”servicestalle.js”* och öppna *”servicestalle.html”* i webbläsaren.

Efter att du godkänt att din geografiska position läsas bör du se ungefär följande, och om du för muspekaren över markören bör du se texten ”Här är jag!” nära markören.



OK, nu har du kommit långt!

Du har tagit reda på din position, du har ritat en karta över närområdet och du har markerat din position på kartan. Nu återstår att hitta närmaste Serviceställe, rita ut den på kartan och ta fram en vägbeskrivning!

Steg 6 – PostNords Öppna API

För att hitta närmaste Serviceställe så finns ett Servicepoint API som beskrivs på PostNord Developer Portal och som har en metod (funktion) som kommer att vara användbar för dig: **findNearestByCoordinates**.

Funktionen behöver fem parametrar enligt följande:

apikey	En API-nyckel som man kan få genom att registrera sig som utvecklare på https://developer.postnord.com. För den här övningen behövs inte det!
countryCode	Anger vilket land som avses. Du använder ”SE” = Sverige.
northing	Latituden för den punkt du befinner dig på
easting	Longituden för den punkt du befinner dig på
numberOfServicePoints	Antal Serviceställen som man vill ha information om. Du ska bara ta reda på det närmaste så du använder värdet 1.
locale	Det språk som man vill ha resultatet på. Du använder ”se” = svenska!

PostNords APIer använder protokollet REST för anrop. Det gör det väldigt enkelt att använda genom att hela anropet, med parametrar och allt, ”bakas ihop” till en URL (en Internet-adress) som sedan anropas.

Du ska nu skriva en ny funktion i ”servicestalle.js” som anropar den här funktionen i PostNords API och returnerar information om geografisk position och vad Servicestället heter.

Förberedelse

I Javascript är det av säkerhetsmässiga skäl tekniskt knepigt att anropa en funktion som inte kommer från den server som man hämtat webbsidan ifrån. För att komma runt det tekniska hindret ska du ta hjälp av ett Javscript-ramverk som heter JQuery. JQuery innehåller många bra-att-ha-funktioner som på många sätt underlättar programmeringen i Javascript. JQuery är mycket vanligt och används till ca 70% på de 10 000 största siterna på Internet. PostNord är en av dessa.

För att inkludera JQuery ska du först lägga till följande rader i HTML-filen *”servicestalle.html”* före raden där du inkluderar din egen javascriptfil, *”servicesstalle.js”*:

```
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
```

Sedan behöver du också göra en mindre justering I “servicestalle.js” för att säkerställa att koden är I synk med laddningen av JQurey.

Först ska du lägga till följande efter variablerna som deklareras i början av filen men före deklarationen av funktionen **positioneringOK**:

```
$(document).ready(function ()
{
```

Sedan ska du lägga till följande längre ner i filen, före deklarationen av funktionen **googleMapsLaddat**:

```
});
```

Det kan tyckas ”petigt” men det är avgörande för att koden ska vara laddad, synkroniserad och användbar i alla delar.

Deklarera funktionen hamtaServicestalle

En egenhet hos de flesta programmeringsspråken är att de har sin grund i Engelska och inte tillåter användning av andra tecken än de engelska. Detta gäller även Javascript så den funktion du nu ska skapa skulle på svenska kunna kallas hämtaServiceställe, men begränsningarna i Javascript gör att du behöver byta ut ”ä” mot ”a” eller ”ae” – här föreslås ”a”.

Lägg till följande kod sist i filen *”servicestalle.js”* :

```
function hamtaServicestalle()
{
    // Bygg ihop en URL till PostNords API med rätt parametrar och värden
    // Grund-URL
    var postnordURL =
    "https://api2.postnord.com/rest/businesslocation/v1/servicepoint/findNearestByCoordinates.json";

    // Lägg till API-nyckel
    postnordURL = postnordURL + "?apikey=cb660527bd04cf4b56648e776e670313";

    // Lägg till landskod
    postnordURL += "&countryCode=SE";

    // Lägg till latitud för aktuell position
    postnordURL += "&northing=" + minLatiud;

    // Lägg till longitud för aktuell position
    postnordURL += "&easting=" + minLongitud;

    // Lägg till att bara ett serviceställe ska hämtas
    postnordURL += "&numberOfServicePoints=1"

    // Lägg till att språket är svenska
    postnordURL += "&locale=se"

    // Använd JQuery funktionen ajax för att genomföra anropet av PostNords API
    // Om det går bra ska funktionen svarFranPostNord anropas
    // Om det blir något fel ska ett felmeddelande visas
    $.ajax(
    {
        url: postnordURL,
        dataType: 'jsonp',
        success: svarFranPostNord,
    })
        .fail(function (jqXHR, textStatus, errorThrown)
        {
            alert("Något gick fel i kommunikationen med PostNords API. Vänligen försök igen senare!");
        });
};
```

Förklaring till koden:

Först skapas variabeln postnordURL som används för att bygga ihop Internet-adressen för anropet till PostNords API. Därefter adderas de olika parametrarna med sina värden. Notera att **minLatitud** och **minLongitud** andvänds som värden för parametrarna **northing** och **easting**.

Till sist används JQuery-funktionen **ajax** för att genoföra anropet av APIet. Till anropet används den nyligen ihopbyggda url:en **postnordURL**.

dataType: `'jsonp'` anger att det är ett anrop till en annan server än den som webbsidan hämtades från och att JQuery ska hantera denna komplikation.

success pekar på en funktion som ska anropas om allt går bra, i detta fall funktionen **svarFranPostNord**, som du ska skriva i nästa moment!

fail innhåller en funktion som anropas om något går fel. I så fall kommer ett felmeddelande att visas enligt **alert**.

För att testa det här så måste funktionen som hanteras svaret från PostNords API fixas till.

Hantera svaret från PostNords API

När anropet till PostNords API har gått bra så skickar funktionen **ajax** svaret som ett dataobjekt till den funktion som angivits efter **success**. Dataobjektet skickas som en parameter till funktionen.

Här samverkar på ett fiffigt sätt PostNords API med Javascript på så sätt att REST-protokollet som används skickar svaret man får i anropet i JSON-format. Och JSON-formatet kan direkt ”omfamnas” och hanteras som ett Javascript objekt.

Det här underlättar mycket för dig som programmerare som då slipper konvertera eller försöka komma på några finurliga sätt att komma åt de delar av datat som du är intresserad av. Du kan adressera det direkt! Det kan låta kryptiskt, men det är enkelt för både den som kan Javascript och den som är nybörjare!

Det data som du behöver ur svaret från PostNord är koordinaterna för det närmaste servicestället och dess namn.

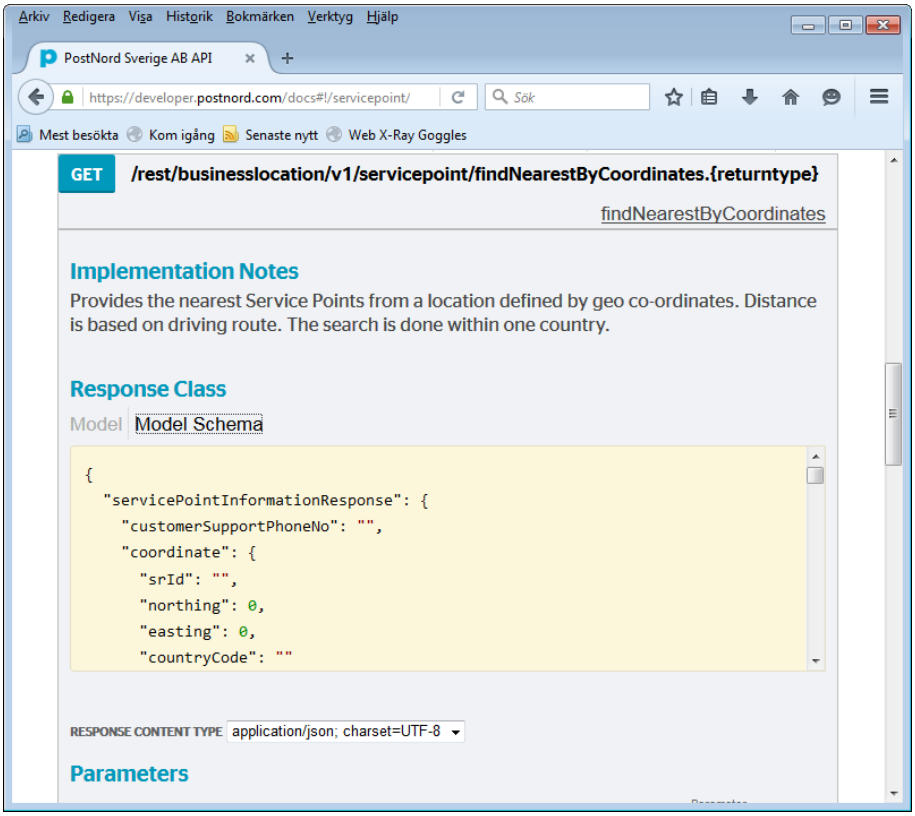
För att förstå hur du ska adressera de delarna behöver du titta på PostNords utvecklarportal för att se hur data i svaret är strukturerat.

Gå till https://developer.postnord.com/ i din webbläsare och klicka på **API DOCS**.

Strukturen på data som returneras från PostNord

Scrolla ner till **servicepoint:** ock klicka sedan på raden som inleds av GET och som innehåller **findNearestByCoordinates**.

Då öppnas informationen om den funktionen/metoden och du ser ungefär följande:



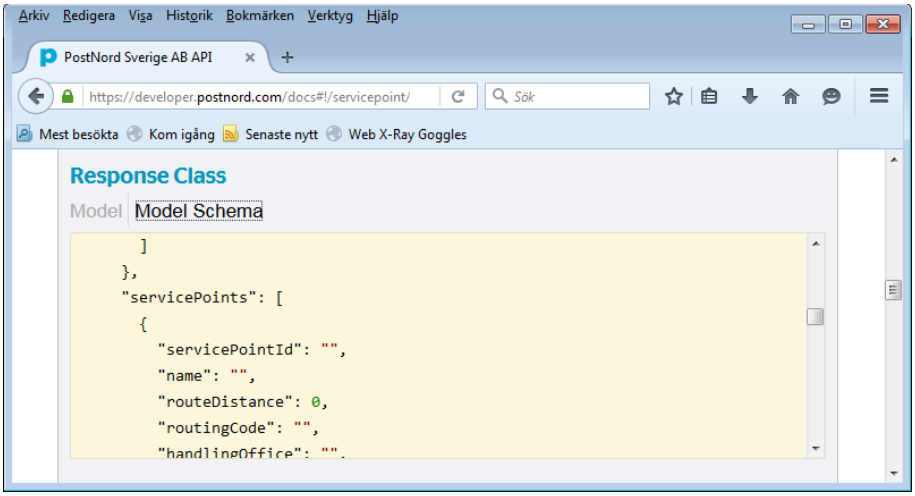
Det intressanta nu är det som visas i den gula rutan; ”Response Class”, ”Model Schema”. Den visar strukturen på det data som ett anrop till funktionen returnerar.

Då kan du se att där finns **northing** och **easting** som du känner igen!

Men det är **inte** latitud och longitud för serviceställets geografiska position. Det är de värden som du angav som din position när du anropade funktionen – så det vet du redan.

Notera det översta namnet här **servicePointInformationResponse** som är namnet på den struktur som omfamnar all den data som returneras.

För att hitta koordinaterna till servicestället så behöver du scrolla ner lite.



Här hittar du namnet **servicePoints**. Så det börjar brännas. Notera att det är plural, servicePoint**s**! APIet kan returnera en räcka med serviceställen i en så kallad **array**. Efter ”servicePoints”: står det en hakparentes **[**. Den indikerar början av en array.

När du anropar APIet så frågar du bara efter ett serviceställe, det närmaste. Och i svaret du kommer att få så kommer det på första platsen i arrayen.

Serviceställets namn

I gula rutan ovan ser du namnet på ett värde som du ska ta hand om, **name**. Det är namnet på servicestället, t.ex. ”Hemköp Täby Centrum”.

I en sådan här struktur så kan man i Javascript adressera alla värden i strukturen genom att ange de olika namnen, från det översta namnet i strukturen ner till namnet på det värde man söker, med punkter mellan namnen på de olika nivåerna. Så, för att adressera värdet för **name** i det här fallet anges den här adressen: **servicePointInformationResponse.servicePoints[0].name**.

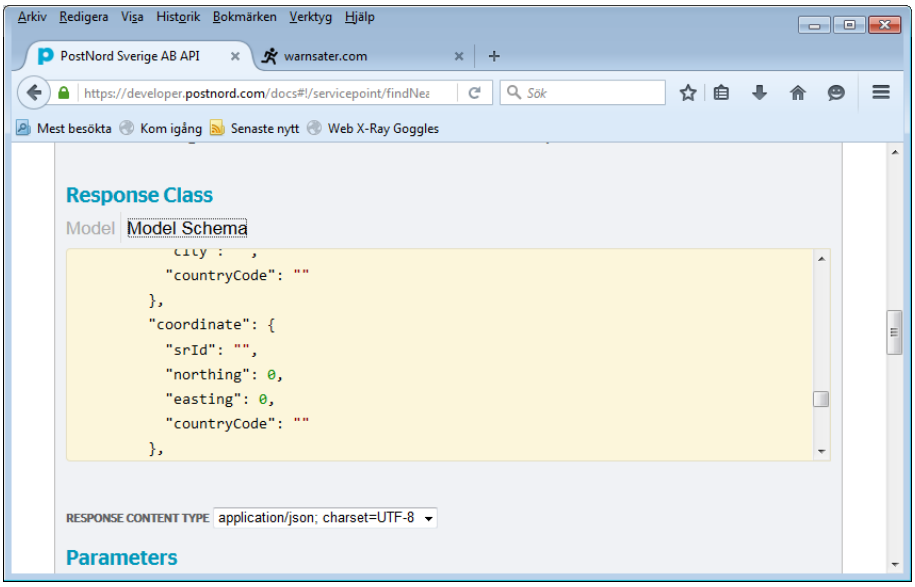
Om man ska läsa den adressen på ”svenska” så blir det ungefär så här:

> ”ge mig värdet på **name** som finns i första positionen i arrayen **servicePoints** som finns i strukturen **servicePointInformationResponse**”.

servicePoints[0] anger första positionen i arrayen. Det här en underlighet i många programmeringsspråk, att man börjar räkna från noll (0). Så är det med arrayer i Javascript. Den första positionen i en array är position noll, andra positionen är position 1, osv.

Serviceställets koordinater

För att rita ut servicestället på kartan så behöver du också koordinaterna. Scrolla ner en bit till i den gula rutan så ser du



Här har du koordinaterna till servicestället. I linje med hur **name** adresserades ovan så adresseras värdet för latituden (**northing**) så här:

servicePointInformationResponse.servicePoints[0].coordinate.northing

Nu vet du hur du ska komma åt serviceställets namn och koordinater. Dags att börja koda!

Funktionen svarFranPostNord

Nu behöver du deklarera tre nya variabler som ska användas för att spara undan serviceställets namn och position. De ska också skrivas in i början av ”servicestalle.js” efter deklarationen av variabeln **karta** med före `$(document).ready(function ()`

Skriv in följande:

```
//Variabler för att spara Serviceställets position
var servicestalletsLatitud = 0;
var servicestalletsLongitud = 0;

//Variabel för att spara Serviceställets namn
var servicestalletsNamn = "Okänt serviceställe";
```

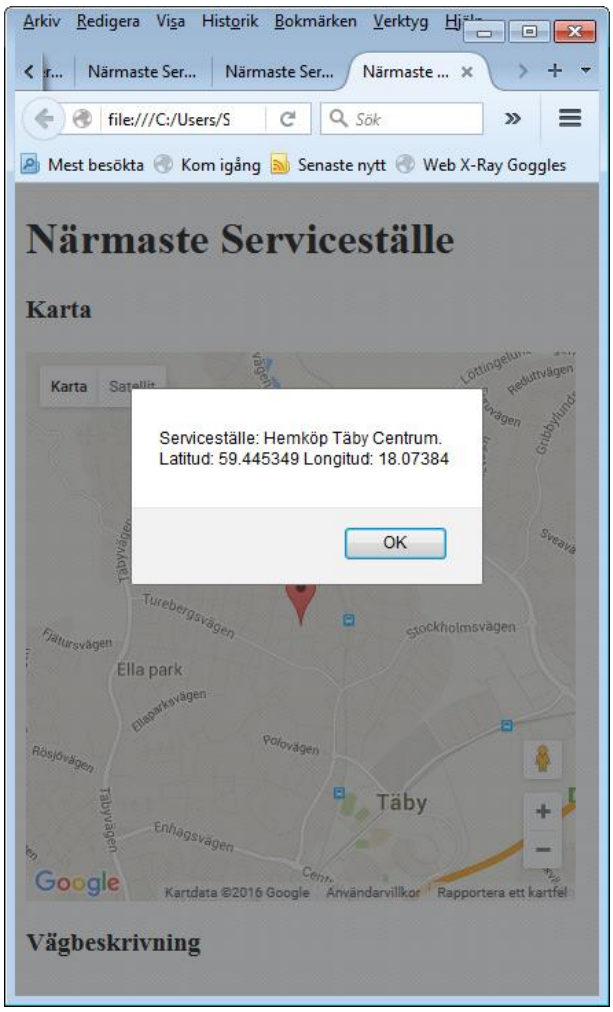
Då har du variablerna klara. Skriv in följande kod i slutet av filen *”servicestalle.js”* som deklarerar funktionen **svarFranPostNord**:

```
function svarFranPostNord(svar)
{
    // Hämta serviceställets namn och koordinater från svaret
    servicestalletsNamn = svar.servicePointInformationResponse.servicePoints[0].name;
    servicestalletsLatitud = svar.servicePointInformationResponse.servicePoints[0].coordinate.northing;
    servicestalletsLongitud = svar.servicePointInformationResponse.servicePoints[0].coordinate.easting;

    //Endast för test - visa serviceställets namn och position
    alert("Serviceställe: " + servicestalletsNamn +
        ". Latitud: " + servicestalletsLatitud + " Longitud: " + servicestalletsLongitud);
};
```

> ***Förklaring till koden:***
> Här sparar du undan serviceställets namn och koordinater från svaret från PostNords API. Här inleds ”adressen” till de olika värdena med **svar.** följt av adresseringen som redovisades ovan.
> För att testa så anropas alert för att visa serviceställets namn och koordinater.

Testa

Ok, spara nu *”servicestalle.js”* och öppna *”servicestalle.html”* i webbläsaren. Då bör du se ungefär följande:



Placera en markör på serviceställets position

Nu, innan rutten ritas ut, ska du placera en markör som visar servicestället på kartan, och sätter titeln på markören till namnet på servicestället – så att det visas om man håller markören över!

För att göra detta så skapar du en ny funktion **markeraServiceställe** som gör just det som sedan kan anropas av **svarFranPostNord**.

Lägg till följande kod sist i filen *”servicestalle.js”* :

```
function markeraServicestalle()
{
    // Variabel för inställningar av markör
    var there = {
        position: { lat: servicestalletsLatitud, lng: servicestalletsLongitud },
        map: karta,
        title: servicestalletsNamn
```

```
    };
    // Sätt en markör på serviceställets plats
    var marker = new google.maps.Marker(there);
};
```

Det är exakt samma förfarande här som när du placerade markören på startpositionen, du bytte bara koordinaterna till serviceställets koordinater och satte titeln till namnet på serviceället.

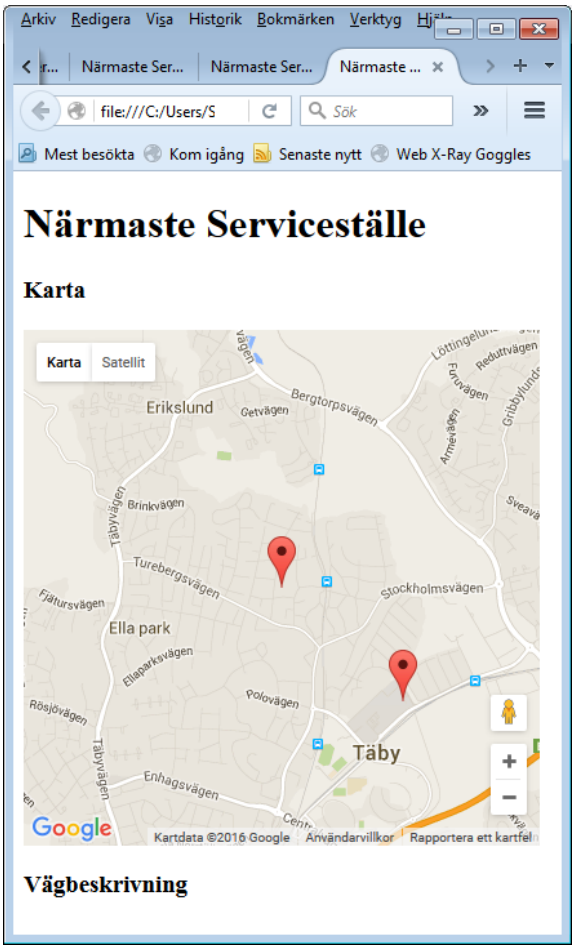
Nu behöver du bara ”kommentera bort” anropet till **alert** i slutet av **svarFranPostNord** och lägga till ett anrop till den nya funktionen **markeraServicestalle**.

Efter det ska **svarFranPostNord** se ut ungefär så här:

```
function svarFranPostNord(svar)
{
    // Hämta serviceställets namn och koordinater från svaret
    servicestalletsNamn = svar.servicePointInformationResponse.servicePoints[0].name;
    servicestalletsLatitud = svar.servicePointInformationResponse.servicePoints[0].coordinate.northing;
    servicestalletsLongitud = svar.servicePointInformationResponse.servicePoints[0].coordinate.easting;

    //Endast för test - visa serviceställets namn och position
    //alert("Serviceställe: " + servicestalletsNamn +
    //    ". Latitud: " + servicestalletsLatitud + " Longitud: " + servicestalletsLongitud);

    // Anropa markeraServicestalle för att rita ut en markör på kartan där serviceället är
    markeraServicestalle();
};
```

Testa

Spara nu *”servicestalle.js”* igen och öppna *”servicestalle.html”* i webbläsaren:



…och om du håller muspekaren över markören för servicestället så bör du se namnet på servicestället!

Nu är det bara en liten bit kvar. Att rita ut rutten från ”min position” till servicestället samt att visa en vägbeskrivning!

Steg 7 – Rutt och vägbeskrivning

För att beräkna rutter och visa vägbeskrivningar behöver du använda en del av Google Maps API som kallas **Directions API**. Om du använder kartorna på https://www.google.se/maps/ så är det **Directions API**et som används när du tar fram vägbeskrivningar och olika resealternativ

Om du vill djupdyka i **Directions API**et och dess olika möjligheter så kan du läsa om det här https://developers.google.com/maps/documentation/directions/ och här https://developers.google.com/maps/documentation/javascript/directions

Där finns ”eoner” av dokumentation och massor med kodexempel.

För att beräkna rutten och visa vägbeskrivning i det här fallet behöver du använda två tjänster i **Directions API**et

- **DirectionsService** som är den tjänst som beräknar själva rutten mellan de två punkterna
- **DirectionRenderer** som är den tjänst som kan visa den beräknade rutten på kartan och ge en verbal vägbeskrivning

Med de här två tjänsterna så kan du ett slag både rita ut rutten på kartan och få fram den verbala vägbeskrivningen. Skriv nu in följande kod i slutet av filen *”servicestalle.js”*:

```
function ritaRutt()
{
    // Initiera Goggle Maps API tjänster för att beräkna och visa rutt
    var directionsService = new google.maps.DirectionsService;
    var directionsDisplay = new google.maps.DirectionsRenderer;

    // Tala om för "ruttritaren" vilken karta den ska rita på
    directionsDisplay.setMap(karta);

    // Tala om var på webbsidan texten för vägbeskrivning ska placeras
    directionsDisplay.setPanel(document.getElementById('beskrivning'));

    // Tala om att "ruttritaren" inte ska sätta ut några markörer
    // (det är ju redan fixat!)
    directionsDisplay.setOptions({ suppressMarkers: true });

    // Anropa ruttberäknaren mes startpunkt (origin), slutpunkt (destination) och färdsätt (promenad)
    directionsService.route(
        {
            origin: { lat: minLatitud, lng: minLongitud },
            destination: { lat: servicestalletsLatitud, lng: servicestalletsLongitud },
            travelMode: google.maps.TravelMode.WALKING
        },
        function (response, status)
        {
            if (status === google.maps.DirectionsStatus.OK)
            {
                // Om ruttberäkningen gick bra - rita rutten och visa vägbeskrivningen!
                directionsDisplay.setDirections(response);
            }
            else
            {
                // Om det inte gick bra - Visa felmeddelande!
                alert("Det gick inte att bestämma färdväg. Försök senare!");
            }
        });
};
```

> ***Förklaring till koden:***
> Du skapar funktionen **ritaRutt**().
> Google Maps API tjänsterna DirectionsService och DirectionsRenderer initieras och referenser till dem sparas i de två variablerna directionsService respektive drectionsDisplay.
> Med **directionDisplay** funktionerna **setMap**, **setPanel** och **setOptions** ges intruktioner till directionDisplay hur karta och vägbeskrivning ska visas.
> Sedan anropas **directionService.Route** som gör beräkningen av rutten från **origin** till **destination** enligt färdsättet **travelMode**. Efter att rutten beräknats körs funktionen son angivits och om det gick bra att beräkna rutten så anropas **directionsDisplay.setDirections** som ritar ut rutten på kartan och visar instruktionerna!

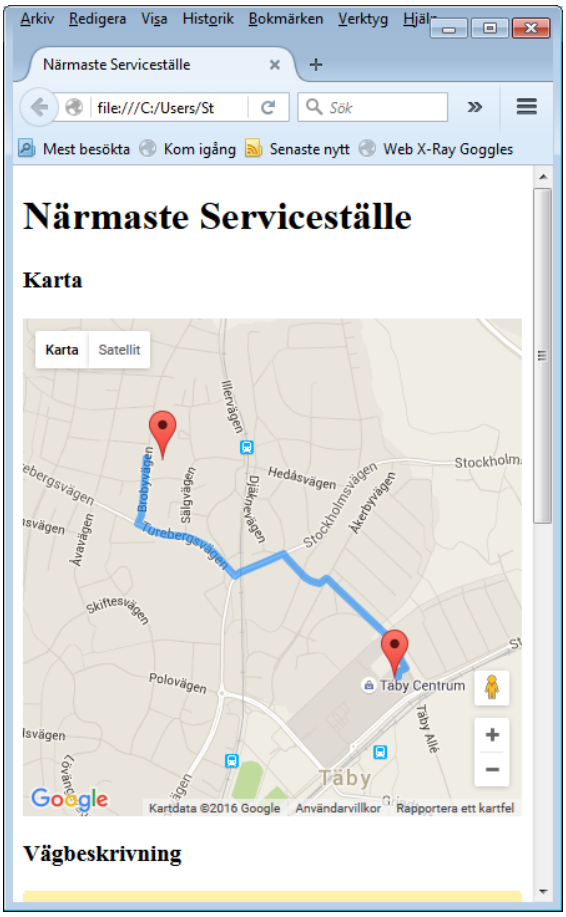
Nu har du deklarerat funktionen **ritaRutt** som kan visa rutten på kartan och trolla fram en vägbeskrivning. Nu återstår bara en liten men viktig detalj: Att se till att ditt program anropar **ritaRutt**.

Det gör du genom att lägga till anropet i slutet av funktionen **svarFranPostNord**, efter anropet till **markeraServicestalle** lägga till anropet till **ritaRutt**:

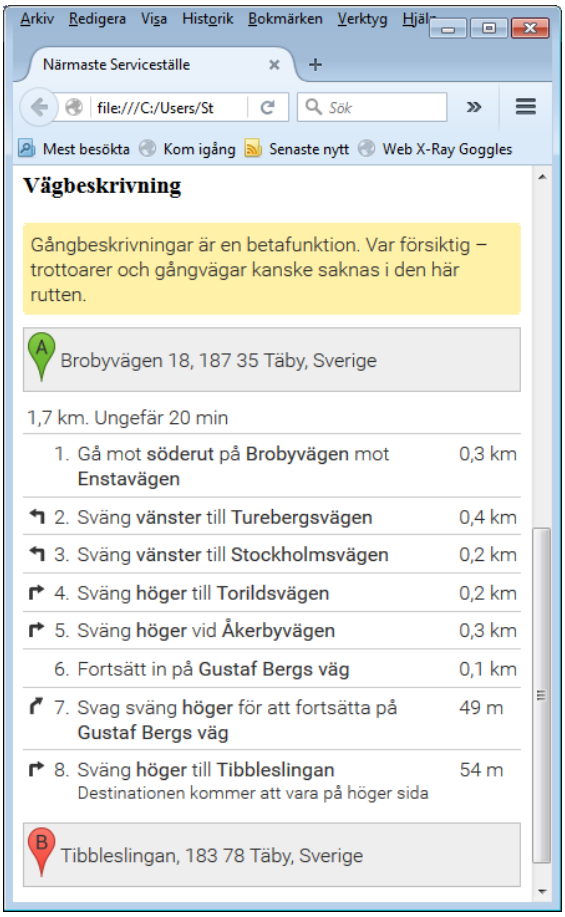
```
    // Anropa ritaRutt för att visa rutten på kartan och visa vägbeskrivning
    ritaRutt();
```

Spara filen *”servicestalle.js”* och öppna filen *”servicestalle.html”* i webbläsaren så bör du se ungefär följande:

Karta



Vägbeskrivning (nerscrollat)



Mission completed

Så där ja, där satt den!

Nu har du skapat webbappen. Låt oss då kolla kraven igen, som sattes i början:

> ***Målet med övningen är att skapa en webbapp som på en karta visar var du befinner dig, var PostNords närmaste Servicesställe finns och en vägbeskrivning hur du tar dig dit.***

- Skapa webbapp: Check √
- Visa karta karta: Check √
- ..som visar var du befinner dig: Check √
- visar var PostNords närmaste Serviceställe finns: Check √
- visar vägbeskrivning: Dubbel-check √√ (visar både rutt på karta och verbal beskrivning)

Alla krav är uppfyllda och du kan nöja dig här.

Om du tycker att appen kan snyggas till lite, fortsätt då med nästa överkursuppgift!

Steg 8 – Pimpa appen!

Den här uppgiften är en självstudieuppgift.

Titta på exempelkoden i mappen Steg-8.

CSS-, HTML- och Javascript-koden har kompletterats på några ställen för att få appen att se lite snyggare ut och så att den blir responsiv.

Med responsiv avses att appen kan ändra storlek automatiskt om storleken på skärmen ändras. Till exempel om en användare kör appen från en mobiltelefon och vrider skärmen så ändras bredd- och höjd-förhållandet. Att få till att kartan ”beter sig responsivt” är den tekniskt mest trickiga delen i hela den här övningen. Det mesta av de ”vanliga” HTML-delarna kan man få responsiva genom lite justeringar i CSS-koden. Men eftersom kartan kontrolleras av Google Maps API så krävs det lite Javascript för att reagera på storleksförändringar på skärm/fönster och sedan justera kartans storlek.

Ändringarna som är gjorda är rikligt kommenterade så du kan titta i koden och läsa dig till de gjorda förändringarna. Sedan kan du också titta i de här referenserna för att lära dig mer om HTML, CSS och Javascript:

Generisk tutorial till HTML, CSS & Javascript mm

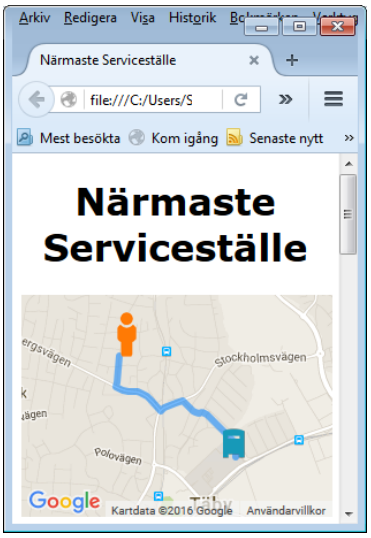
http://www.w3schools.com/html/default.asp

Allt om JQuery

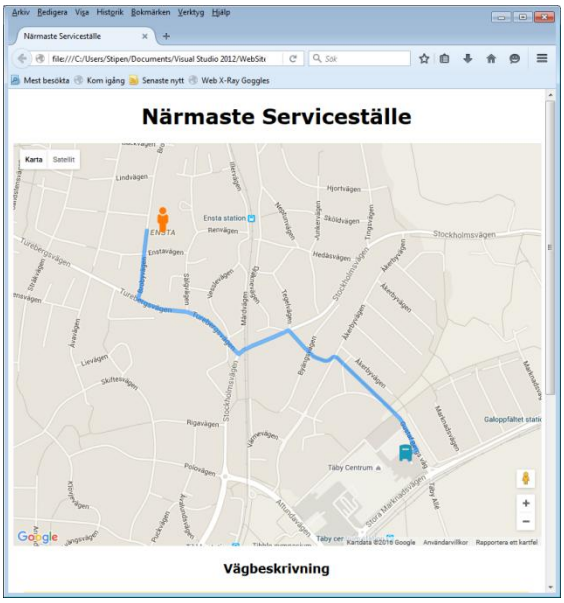
https://jquery.com/

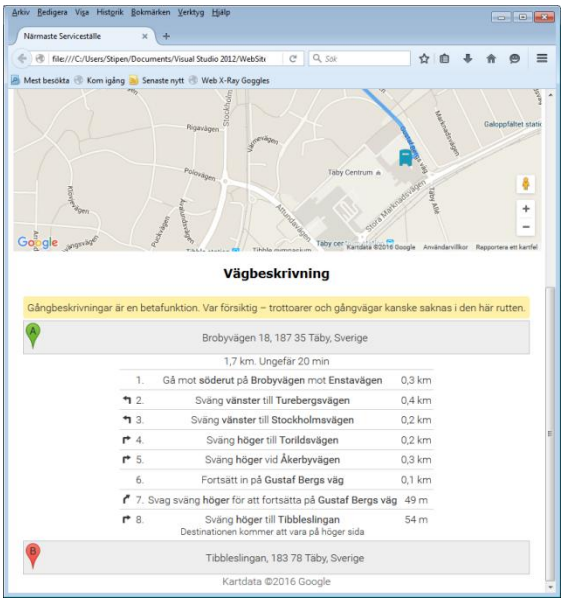
Pimpat exempel

Minimerad:



Bredare:





Steg 9 – Anpassa till mobila device

För att göra appen möjlig att installera lokalt på t.ex. en iPhone så krävs ytterligare en del kompletteringar av koden.

Ikoner

Bland annat så behöver du skapa ett antal ikoner (bilder) som symboliserar appen när den är installerad. För att passa alla tänkbara device så bör man skapa ikoner i en mängd olika storlekar. Är man ett programvaruhus som producerar spel-appar så är detta en uppgift för en grafisk designer som skräddarsyr ikonerna för varje storlek.

Men som hobbyprogrammerare kan man vara lat! I den här övningen har du använt en bild som symboliserar ett Serviceställe. Det är en bild på 45x45 pixlar och med hjälp av den så kan man på en webbsida (förstås!) generera alla nödvändiga storlekar av ikonen.

Webbsidan hittar du här: http://www.favicomatic.com/

Där kan man ladda upp bilden man vill ha duplicerad i olika storlekar och så får man tillbaka en ZIP-fil med ikonerna plus en fil med kod som man kan kopiera in HTML-kod.

Referenser till ikonerna ska läggas in i HTML-filen inom **<head>**-taggen. Under mappen Steg-9 hittar du HTML-koden med de inkopierade ikon-referenserna.

Startup-bild

För iPhone/iPad bör man också komplettera med en s.k. startup-bild. Det är en bild som visas (en kort stund) när appen installerats på devicet och när man startar den, om den tar lite tid att ladda.

Bilden ska vara 320x480 pixlar och porträtt-orienterad. För den här övningen finns det en bild som skapats i Photoshop i det här formatet som du hittar i **Steg-9**-mappen med namnet *startup.png*.

Mobilanpassing

För att ytterligare anpassa appen till mobilen och göra den installerbar så måste appen ”berätta” för devicet att den är anpassad för det. Det gör med följande kod i *”servicestalle.html”* som placeras inom **<head>**-taggen:

```
<!-- Tala om för en iPhone/iPad att appen kan köras i helskärmsläge -->
    <meta name="apple-mobile-web-app-capable" content="yes">

    <!-- Tala om för en iPhone/iPad/Android att skalning/zoom ska vara 1.0 (ingenskalning/zoom) -->
    <!-- Eftersom appen är enkel vill vi också unvika att användaren zoomar - zoom i kartan undantagen
         lite redundant, men ibland behöver man vara övertydlig när många olika device sta stödjas! -->
    <meta name = "viewport"
        content = "initial-scale=1.0, minimum-scale=1.0, maximum-scale=1.0, user-scalable=no,
                   width=device-width">
```

Till sist kan du också stänga av den inbyggda funktionen i iPhone/iPad som gör att när du håller fingret en stund på skärmen kan du få upp en fråga om du vill kopiera. Det kan ibland vara störande.

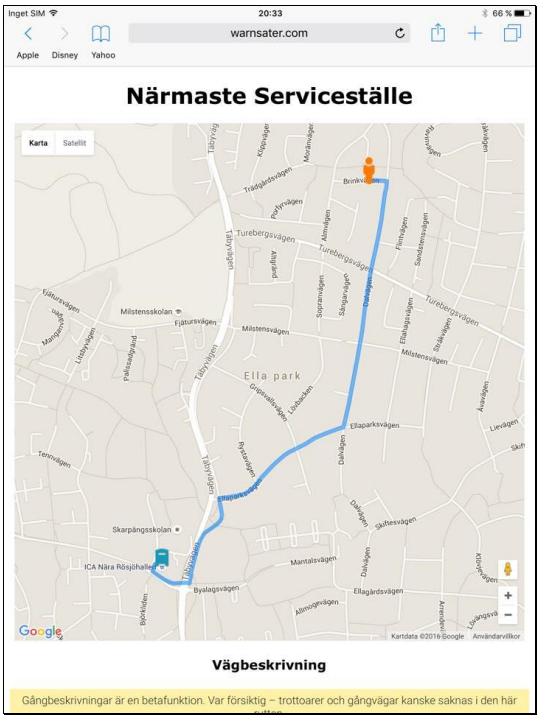
Den funktionen stängs av med en instruktion i CSS-filen *”servicestalle.css”* så här:

```
*
{
    /* Unvik att den "irriterande" kopiera-funktionen aktiveras om

        man råkar hålla fingret länge på en iPhone/iPad-skärm */
    /* '*' som inleder den här inställnigen avser alla element på sidan (allt innehåll)  */
    -webkit-user-select: none;
}
```

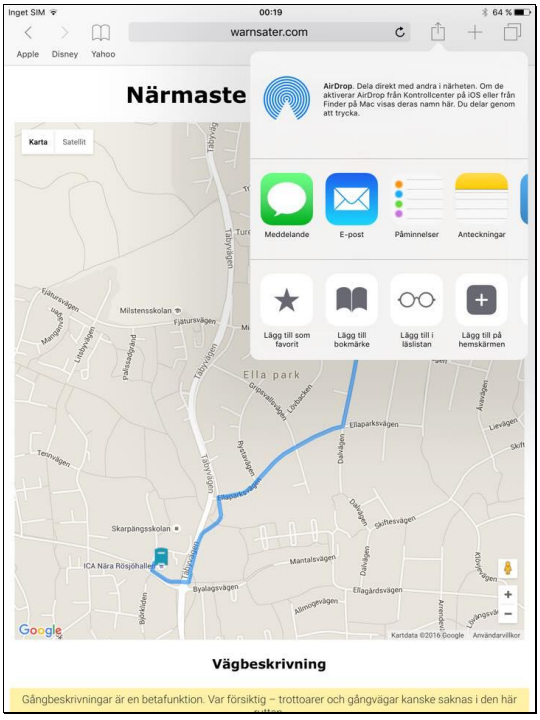
Du kan läsa mer om allt du kan göra för att anpassa din webbapp till iPhone m.fl. på Apples utvecklarsajt som du hittar här:
https://developer.apple.com/library/ios/documentation/AppleApplications/Reference/SafariWebContent/ConfiguringWebApplications/ConfiguringWebApplications.html

Bildexempel

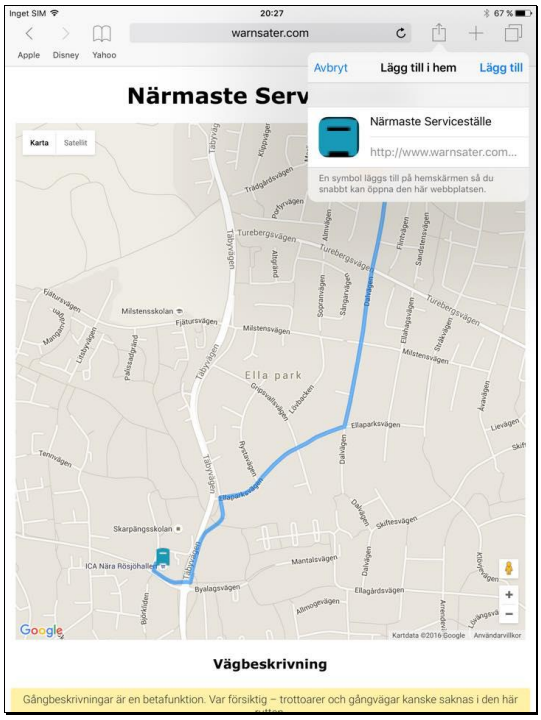
Så här kan appen se ut när du kör den från en iPad i Safari, innan den är installerad:



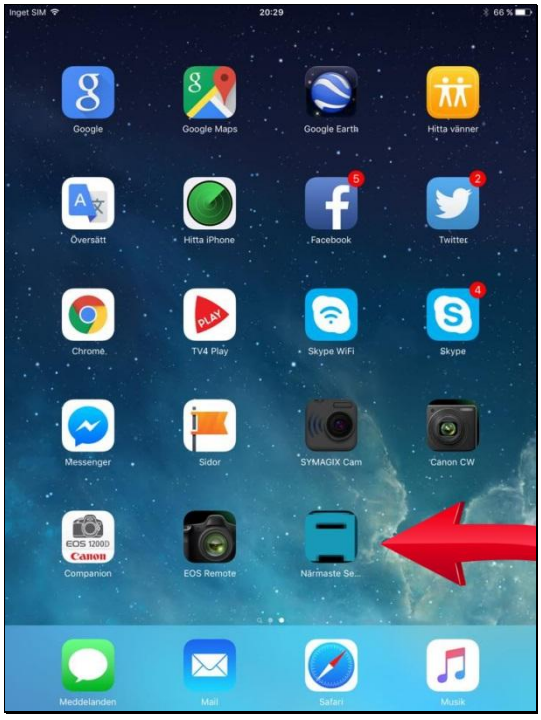
Om du klickar på den lilla rutan uppe till höger om webbadressen



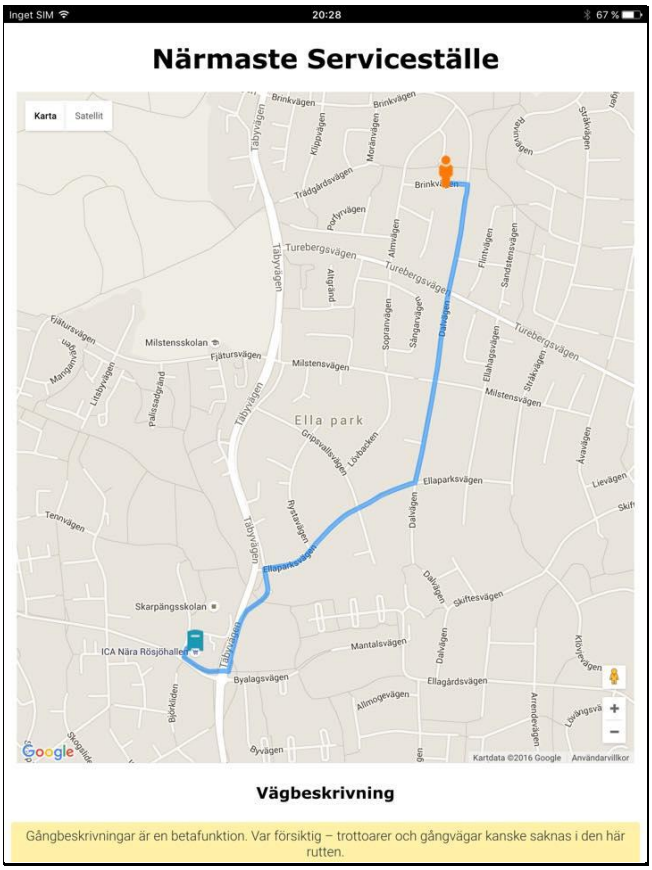
…så kan du välja ”Lägg till på hemskärmen”.



På hemskärmen kommer då appen upp med sin ikon så att du kan starta den direkt därifrån:



När du sedan startat appen från hemskärmen så ser den ut så här:



Skillnaden nu mot tidigare (innan den installerades) är att appen tar upp hela fönstret och Safaris adressfält och ikoner överst på skärmen är borta. Man får intrycket av att det är en app snarare än en webbsida!

Steg 10 – Publicera din app

Nu lider övningen mot sitt slut och då är det lite tråkigt att konstatera att det här steget har blivit svårare på senare tid.

För att göra den här typen av appar tillgängliga på Internet krävs en server som tillåter tig att spara samtliga filer som du skapat (som i mappen Steg-9) så att vem som helst kan komma åt dem via en URL (en Internet-adress) på Internet.

Tidigare hade de flesta bredbandsleverantörer ”egen hemsida” som ingick i det paketet. Men nu är det nog endast Glocalnet som fortfarande erbjuder det.

Tidigare kunde man också med lite trixande i Dropbox ordna det, men det är nu stängt för ”gratisanvändare”.

Det finns en mängd siter där ute som erbjuder ”gratis hemsida” men det är inget som jag testat och kan rekommendera. Det brukar vara förknippat med begränsningar och framförallt reklam på dina sidor som du inte själv kan kontrollera.

Github

Det gratisalternativ som jag hittat som fungerar och som är seriöst är på Github, https://github.com/

Github är en omfattande tjänst som används av många företag runt om i världen som ”projektplats” för programutveckling i allmänhet och för utveckling av Open Source i synnerhet. Tjänsten är mycket kompetent och omfattande, och även gratis om det man gör är öppet och kan delas med alla.

I tjänsten ingår också att skapa webbsidor där det är tänkt att man kan publicera information om sina projekt, men de går också att använda för att publicera resultaten av projekten om det är just webbsidor med HTML, Javascript och CSS!

Om du vill lära dig hur man gör detta så får jag hänvisa till Githubs startsida https://github.com/ och det du kan hitta om detta genom att Googla!

Koden för den här övningen finns publicerad och körbar från Github och du kan testa den här: http://stwe001.github.io/servicestalle.html

Happy coding!