WebLogger

Ett system för förbättrad felsökningskommunikation mellan användare och utvecklare

Examensarbete inom Data- och Informationsteknik

ANDERS TRUONG
BOBBY PANG

Institutionen för Data- och Informationsteknik

CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2022
www.chalmers.se

Examensarbete 2022

WebLogger

Ett system för förbättrad felsökningskommunikation mellan användare och utvecklare

ANDERS TRUONG
BOBBY PANG

CHALMERS

Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola
Göteborgs Universitet
Göteborg, Sverige 2022

WebLogger
Ett system för förbättrad felsökningskommunikation mellan användare och utvecklare
ANDERS TRUONG
BOBBY PANG

© ANDERS TRUONG, BOBBY PANG 2022.

Handledare: Joachim von Hacht
Examinator: Lars Svensson

Examensarbete 2022
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet
SE-412 96 Gothenburg
Telephone +46 31 772 1000

WebLogger
Ett system för förbättrad felsökningskommunikation mellan användare och utvecklare
ANDERS TRUONG
BOBBY PANG
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
Göteborgs Universitet

Abstract

Today there are millions upon millions of users on the internet constantly scouring over the countless amounts of websites that exist. Daily usage of the internet has become such a natural part of life that oftentimes, people do not really think about the amount of effort that goes into creating and maintaining web applications. It is a common occurrence for errors to occur on websites whether it is issues with user interface or certain functionalities not functioning properly. The error reporting on these mistakes often results in feedback with lacking details due to the users having a difficult time recollecting every step that caused the error. This has led to the development of the WebLogger project. The purpose of the project is to improve error reporting between a user and a developer in conjunction with troubleshooting and improving web applications. The WebLogger project has resulted in the development of a programming library as well as a web application. The WebLogger-library is capable of easily being integrated into other software with minimal changes that logs and saves user events on web applications. The user events are saved locally and can be sent to developers who utilize the WebLogger-application in order to recreate the user events.

Förord

Rapporten är ett resultat av vårt examensarbete på högskoleingenjörsprogrammet Datateknik i Chalmers Tekniska Högskola i Göteborg. Arbetet har utförts av Anders Truong och Bobby Pang under vårterminen 2022. Examensarbetet föreslogs av företaget Diadrom som har visat mycket stöd under examensarbetets gång. Vi vill speciellt tacka Henrik Fagrell som har varit vår kontakt med Diadrom för alla möten och diskussioner kring de tekniska aspekterna av projektet. Även ett stort tack till vår handledare Joachim von Hacht i Chalmers som har hjälpt oerhört mycket med rapporten och haft mycket tålamod med oss.

Bobby Pang & Anders Truong, Göteborg, Juni 2022

Contents

1 Introduktion **1**
- 1.1 Bakgrund . . . 1
- 1.2 Syfte . . . 2
- 1.3 Mål . . . 2
- 1.4 Avgränsningar . . . 2

2 Metod **3**

3 Teknisk Bakgrund **4**
- 3.1 Webbapplikationer . . . 4
- 3.2 HTML och CSS . . . 4
- 3.3 JavaScript . . . 5
- 3.4 TypeScript . . . 5
- 3.5 JSON . . . 5
- 3.6 JavaScript API:er . . . 6
  - 3.6.1 Document Object Model (DOM) . . . 6
  - 3.6.2 Cookies . . . 6
  - 3.6.3 LocalStorage och sessionStorage API:erna . . . 6
- 3.7 NodeJS . . . 7
- 3.8 Angular . . . 7
- 3.9 ngx-capture . . . 7

4 Genomförande **8**
- 4.1 Iterationer . . . 8
- 4.2 Kravspecifikation . . . 8
  - 4.2.1 Mjukvaruavgränsning . . . 9
- 4.3 Systemdesign . . . 9
- 4.4 Utvecklingsmiljöer . . . 10
- 4.5 WebLogger-biblioteket . . . 10
  - 4.5.1 Loggning av händelser . . . 10
  - 4.5.2 Lagring av information . . . 13
- 4.6 WebLogger-Applikation . . . 15
- 4.7 Driftsättning av WebLogger-biblioteket . . . 17
- 4.8 Driftsättning av WebLogger-applikationen . . . 17

5 Resultat **18**
- 5.1 WebLogger-Biblioteket . . . 18

5.2 WebLogger-Applikationen 19
5.2.1 Pages 19
5.2.2 Details 19
5.3 Kvalitativ undersökning av kommunikation 21

6 Slutsats **22**
6.1 Diskussion 22

Källor **24**

1
Introduktion

1.1 Bakgrund

Idag utnyttjar över 60 procent av världens befolkning internet och i Sverige är siffran över 90 procent[1][2]. Applikationerna på internet förväntas fungera i en mängd olika miljöer, till exempel PC-datorer, mobiltelefoner och surfplattor. Ganska vanligt är att det uppstår problem t.ex. på grund av att användargränssnitt i olika skärmstorlekar inte fungerar som det ska eller att viss funktionalitet saknas eller uppför sig felaktigt.

Eftersom att surfa på internet är en daglig rutin för de flesta, uppmärksammar man normalt inte alla detaljer och handlingar som man själv utför. Detta leder ofta till att användare som stöter på problem i webbapplikationer inte själva minns vilka handlingar man har utfört som lett till situationen. Detta är en uppenbar svårighet då användare skall felanmäla problemen.

Denna typ av situation förekommer dagligen för de som arbetar inom teknisk support för olika webbapplikationer. Utifrån mängden människor som använder applikationerna genereras en mycket stor mängd av vaga felanmälningar hos den tekniska supporten [3].

Felanmälningarna som ges av användaren till supporten återkopplas senare till en utvecklare för korrigeringar. Återkopplingen är en av de viktigaste verktygen för utvecklare att förbättra applikationen men det kan bli besvärligt när återkopplingen som ges inte är tydlig eller tillräckligt detaljerad. För utvecklare är det också viktigt att återkopplingen som ges gör det möjligt att återskapa problemet. Att förbättra kommunikationen från slutanvändare till utvecklare är av mycket stor betydelse både för slutanvändaren och företaget. Felsökning och rättningar är dyra [4].

Idén för detta projektet föreslogs av företaget Diadrom som har haft egna erfarenheter om felrapporteringar. Ett exempel som stod ut för Diadrom var en händelse där en kund initialt inte kunde få teknisk hjälp av företaget på grund av felrapportering. Problemet löstes slutligen då det visade sig att vara ett problem med användargränssnitt där knappar var placerade utanför skärmen. Detta ledde till utvecklingen av projektet vid namnet WebLogger som siktar på att förbättra kommunikationen mellan slutanvändare och utvecklare.

1.2 Syfte

Projektets syfte är förbättringen av felrapportering mellan slutanvändare och utvecklare i samband med felsökning och rättning av webbapplikationer.

1.3 Mål

Målet med projektet är att utveckla mjukvara som kan kopplas ihop med existerande applikationer för att hjälpa webbutvecklare att samla in bättre och mer högkvalitativ återkoppling från användaren.

1.4 Avgränsningar

I praktiken kan det vara svårt att mäta hur mycket kommunikationen har förbättrats. Därför kommer i slutändan av projekt, en kvalitativ uppskattning med hjälp av informella tester att utföras. Dessa informella tester anser projektgruppen kan ge en vägledning om projektet bidrog till förbättringen av kommunikationen mellan utvecklare och användare. Projektet kommer fokusera på att hantera följande handlingar; musklick, tangentklick och tidpunkt som ett proof of concept.

2
Metod

Projektarbetet påbörjades med planering av projektet. I planeringen ingick möten med handledare från företaget Diadrom angående projektidéer och planering kring projektidéns genomförbarhet. Efter att projektidén valts påbörjades sökandet efter passande tekniska verktyg. Detta skulle också innebära en förbättring av projektgruppens förståelse kring de nya verktygen och även hur mjukvaran ska framställas för att uppnå projektets mål. Den resterande tiden efter planering och tekniska verktyg, delades upp i 4 iterationer som sträckte sig under 4 veckor. Under denna tid implementerades mjukvaran. Iteration 4 och avslutning hade fokus på rapportskrivande och förberedelse för kommande presentationer.

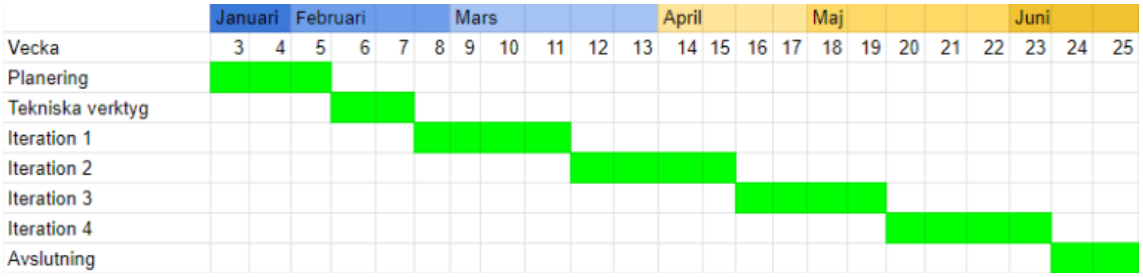

Figure 2.1: Schema över projektarbetet

3
Teknisk Bakgrund

3.1 Webbapplikationer

Webbapplikationer består av en klient-sida och server-sida som ofta refereras som front-end och back-end programming. Kommunikation mellan klient och server sker med hjälp av HTTP protokollet.

Klientsidan exekveras i en webbläsare och består av användargränssnittet och den del av applikationen som kan exekveras lokalt i klienten. Ett exempel på en operation som kan ske på klientsidan är när användaren klickar på en knapp som ändrar utseendet på användargränssnittet. Eftersom denna operationen normalt inte kräver kommunikation med servern kallas det för en klientsidig operation. Klientsidan lagrar även lokal data och kommunicerar med serversidan genom att exempelvis hämta data.

Serversidan består av en webbserver vars syfte är att hantera klientsidans begäran. Serverprogrammering avser implementation av kod som körs i servern. Exempel på serversidiga operationer är konfigureringen och anrop av databaser och generering av dynamiska webbsidor, det vill säga sidor som innehåller dynamisk genererat data oftast från en databas [5].

3.2 HTML och CSS

HTML, Hypertext Markup Language, är ett sidbeskrivningsspråk som möjliggör för webbläsare att bygga upp en intern representation av webbsidan, ett så kallat DOM-träd. (Document Object Model). Detta görs genom att skriva olika element i HTML-koden. Några vanliga element är: sektioner, paragrafer och länkar. Element kan även ha olika attribut. CSS står för Cascading Style Sheets och är ett presentationsspråk som utnyttjas i samband med HTML för att specificera hur ett element ser ut på en HTML sida[6].

```css
.time {
 font-size: 24px;
 color:white;
}
```

Figure 3.1: Kodexempel som visar hur CSS specifierar utseendet för HTML elementet time som representerar tiden som en användare har spenderat på en webbsida

3.3 JavaScript

JavaScript är ett programmeringsspråk som huvudsakligen används för applikationer som skall exekveras i en webbläsare. Applikationer laddas ner och körs som skript i webbläsaren. JavaScript brukar kallas ett objekt-baserat språk eftersom det inte finns några klasser. JavaScript är svagt typat och stödjer högre ordningens funktioner[7].

3.4 TypeScript

TypeScript är baserat på språket JavaScript. Den främsta skillnaden mellan TypeScript och JavaScript är att TypeScript är ett statiskt typat språk. I figur 3 och 4 finns ett exempel på variabeltilldelning i både JavaScript och TypeScript. TypeScript är även mer lämplig för objektorienterad programmering i jämförelse med JavaScript eftersom klasser existerar i TypeScript. Klasser underlättar objektorienterad programmering eftersom de tillför en mall för att skapa objekt[8].

```typescript
//TypeScript
let x: number = 1;  //x är deklarerat som en typ "number"
x = 'asdf'          //Ger ett felmeddelande eftersom x är av typ "number" och inte en String
```

Figure 3.2: Exempel på variabeltilldelning i TypeScript

```javascript
//JavaScript
var x = 1234    // Variabeln x har typen "number"
x = "asdf"      // Variabeln x har typen "String"
```

Figure 3.3: Exempel på variabeltilldelning i JavaScript

3.5 JSON

JSON står för Javascript Object Notation och är ett textbaserat format för att representera strukturerad data baserat på JavaScript-objektens syntax. JavaScript-objekt kan enkelt serialisera och deserialiseras till och från JSON[9].

```
'{"name":"ClickEvent", "X":100, "Y":100}'
```

Figure 3.4: Exempel på information som lagras i JSON format

3.6 JavaScript API:er

3.6.1 Document Object Model (DOM)

DOM API:et används för att manipulera det interna trädstrukturen i webbläsaren och även sköta händelsehantering. JavaScript kan via DOM API:et ändra HTML-sidans struktur och utseende. API:et exponerar ett antal objekt som klientapplikationen kan använda. Några exempel på objekt ur DOM-API:et:

Window
: Representerar webbläsarens fönster som visar webbsidans innehåll [10]

PointerEvent
: Objekt som representerar musklick i webbläsaren [11]

KeyboardEvent
: Object som representerar tangenttryck i webbläsaren [12]

3.6.2 Cookies

Cookies är textinformation som sparar data lokalt i webbapplikationen som exempelvis användarnamn och lösenord. Servern kan få tillgång till denna data vilket skapar ett potentiell säkerhetsproblem, exempelvis kan flera användare drabbas om innehållet hamnar i fel händer. Dessutom får användaren mindre kontroll över vad som lagras. Data som lagras i cookies kan ha olika lång varaktighet beroende på inställningar. Generellt kan datan som lagras i cookies fortsätta att existera även om fliken eller webbsidan stängs ner vilket gör det passande för att spara information som exempelvis inställningar och preferenser. Cookies kan endast lagra data av typen String [13].

3.6.3 LocalStorage och sessionStorage API:erna

LocalStorage och sessionStorage är alternativ till cookies för att spara data i en webbapplikation. Den främsta skillnaden mellan localStorage och session-Storage är varaktigheten av datan. Lagringen av data i sessionStorage sparas endast för en session vilket innebär att om fliken blir nedstängd, rensas datan. Lagringen av data i localStorage sparas tills antingen webbapplikationen eller användaren manuellt rensar data. Om en ny flik med identisk URL öppnas, skapas ett separat session-Storage eller localStorage. Båda typerna kan endast lagra data av typen String vilket gör det passande att utnyttja i samband med textbaserade format som exempelvis JSON. De har även kapaciteten att hålla en större mängd data jämfört med andra alternativ som cookies. Eftersom att data sparas lokalt och servern inte kan begära informationen är det därför ett säkrare alternativ till cookies.

3.7 NodeJS

NodeJS är en plattform som kan exekvera program skrivna i JavaScript och Typescript utanför webbläsaren [14].

3.8 Angular

Angular är ett ramverk som utnyttjar TypeScript för att skapa webbapplikationer. En webbsida representeras av en klass som heter AppComponent. AppComponent innehåller en HTML-fil, en CSS-fil och färdiga inbyggda generella skript och funktioner[15].

Figur 6 visar ett scenario när en klient besöker en webbsida. Klienten frågar servern om webbsidan. Beroende på webbsidans URL kommer ramverkets routing module att kopplas till en av webbapplikationens AppComponent. AppComponent innehåller webbsidans data som kan skickas till klienten.

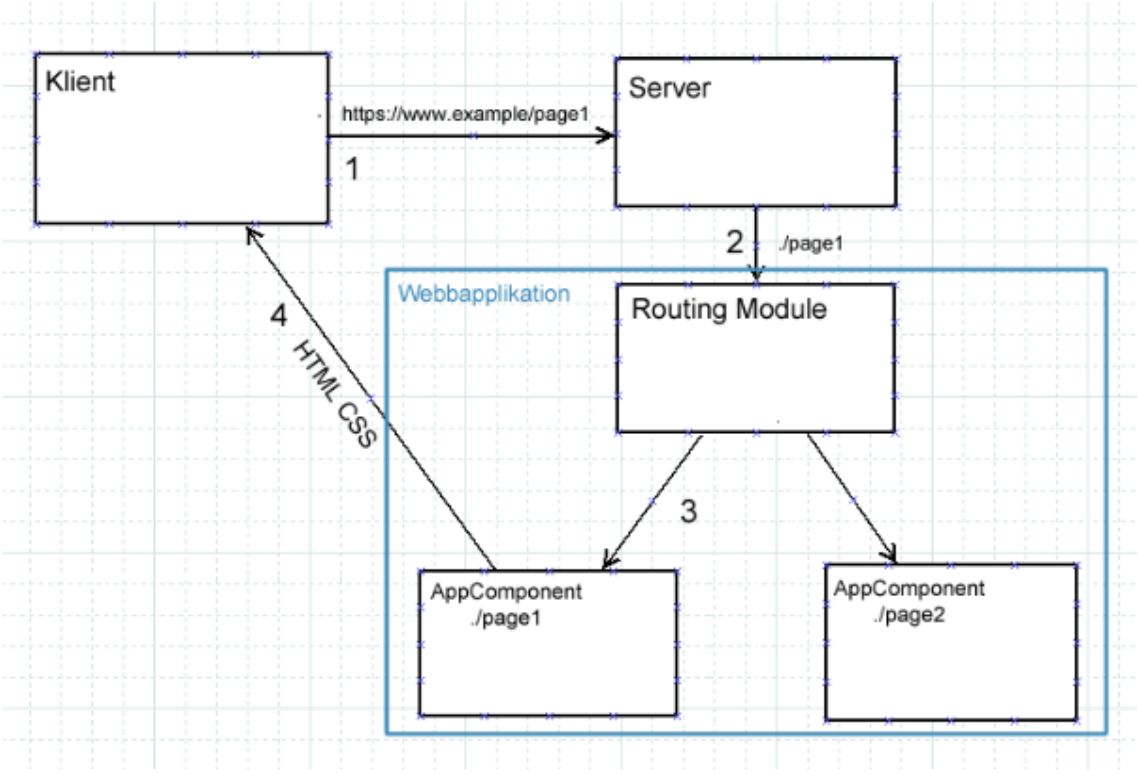

Figure 3.5: Exempel på en klient som besöker en hemsida.

3.9 ngx-capture

ngx-capture är ett Angular-bibliotek som kan utnyttjas för att ta skärmbilder. Med biblioteket kan man definiera ett zon av HTML-sidan som sedan returneras som en sträng innehållande en base64-kodad png-bild [16].

4
Genomförande

4.1 Iterationer

Arbetet av projektet delades främst upp i 4 olika iterationer. För iteration 1 påbörjades utvecklingen av mjukvaran vilket innefattar utvecklingen av en hemsida, loggningen av händelser och lagring av data. Under iteration 2 vidareutvecklades webbapplikation för att återskapa händelser från den lagrade datan. Iteration 3 användes för att anpassa mjukvaran för att kunnas integreras in till andra existerande webbapplikationer med minimala förändringar. Under iteration 1-3 utfördes arbete i ett AGILE-arbetssätt vilket innebär att varje vecka är en så kallad sprint med olika mål och krav på mjukvaran som behöver uppnås. Mellan varje sprint pågick möten med företaget Diadrom kring krav på mjukvaran, rådgivning kring problem och möjliga förbättringar.

Iteration 4 och resterande tid användes för finslipning av mjukvaran som exempelvis optimeringen för skalbarhet och mestadels rapportskrivning. Varje vecka av iteration 4 innebar veckomöten med en handledare från Chalmers för en förbättring av rapportens struktur och andra möjliga förbättringar.

Målet för projektet är att utveckla en mjukvara med syfte att förbättra kommunikation mellan användare och utvecklare. Mjukvaran måste både samla in och presentera datan. Detta åstadkoms genom att projektet delas i två delar: Dels ett bibliotek som kan kopplas ihop med existerande mjukvara för att logga en användares aktivitet på en webbsida, och dels utvecklingen av en webbapplikation som kan presentera den lagrade datan och tillåter utvecklaren att återskapa händelsen. Projektet planeras att byggas som en prototyp med grundläggande funktionalitet.

4.2 Kravspecifikation

- Biblioteket ska vara möjligt att integrera i existerande applikationer med minimala förändringar.
- Biblioteket ska logga följande handlingar; tangenttryck, musklick och tidpunkt.
- Loggad data ska sparas lokalt.
- Loggad data ska kunnas skickas till en extern källa för utvecklare.
- Loggad data ska presenteras på ett tydligt sätt som tillåter utvecklaren att lätt gå igenom användarens steg för att återskapa problemet.

4.2.1 Mjukvaruavgränsning

- Biblioteket och applikationen ska vara en prototyp.
- Mjukvaran ska köras på klientsidan och kunna integreras i existerande mjukvara med minimala ändringar.

4.3 Systemdesign

Systemet består av två PC-datorer för respektive användare, personen som upptäcker fel, och utvecklaren, se figur 7. För båda datorerna gäller att valfritt operativsystem kan användas Windows, iOS eller Linux och valfri Webbläsare.

På användarens PC finns en webbläsaren som kör någon webbapplikation som eventuellt behöver felsökas. (AppComponent i fig 7). Integrerad med denna är WebLogger-biblioteket (Kallas för WL-bibliotek framöver). WL-biblioteket loggar händelser som tangenttryck, musklick, tidpunkt, bild på webbsidan efter varje händelse och sparar datan lokalt i webbläsarens sessionStorage. Bilden sparas i base64 string och resterande datan i JSON-format. Vid behov av felrapportering kan användaren skicka den lagrade datan som en fil till utvecklaren.

Utvecklarens PC kör en separat webbapplikation som kallas för WebLogger-applikationen (Kallas för WL-applikation framöver). Applikationen kan presentera det lagrade datan för utvecklaren och återskapa alla händelser som användaren har gjort. WL-Applikation är skriven i Angular med plattformen NodeJS. Initialt hade projektet mål om att spåra fler typer av data som exempelvis operativsystem och inställningar men eftersom att webbläsaren inte har tillgång till dessa typer av data kunde inte detta åstadkommas.

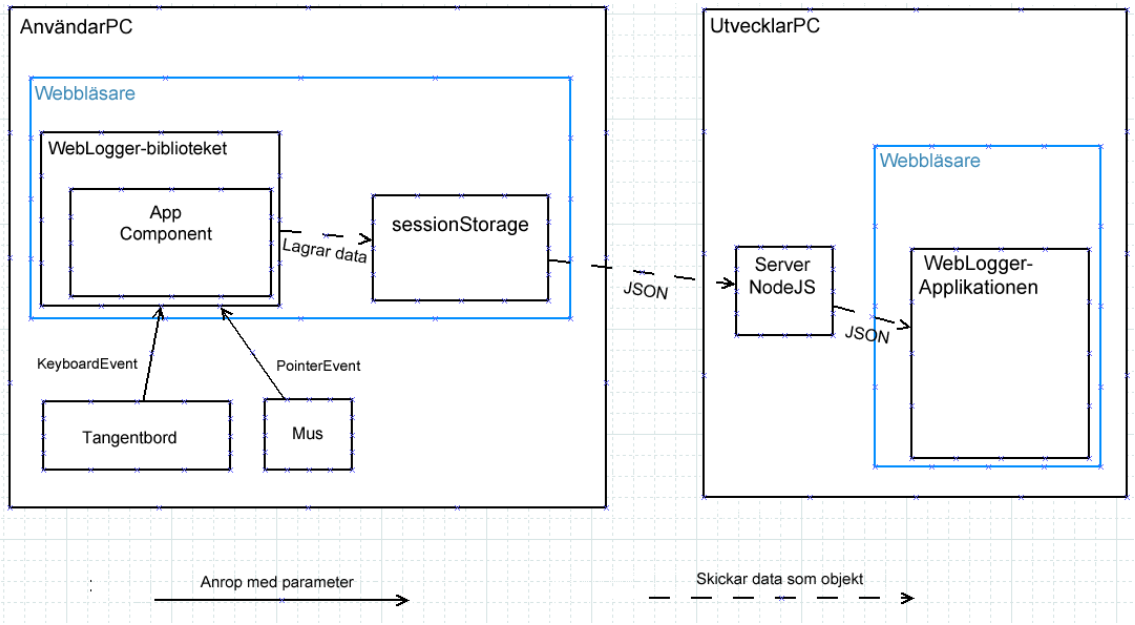

Figure 4.1: Översiktlig bild av WL-biblioteket och WL-applikationen. Händelser plockas upp av WL-Biblioteket och lagras i sessionStorage i JSON. Denna lagrade datan kan användas i WL-applikationen för att återskapa händelser.

4.4 Utvecklingsmiljöer

- Webstorm IDE; specialiserat för JavaScript som innehåller flera verktyg och funktionaliteter som underlättar programmering.
- Github; Utvecklingsplattform för att dela kod.
- Angular; Ramverk för webbapplikationer
- NodeJS; Platform för att exekvera program i TypeScript och JavaScript

4.5 WebLogger-biblioteket

4.5.1 Loggning av händelser

Den existerande webbapplikationen som skall felsökas ärver fyra funktioner från WL-biblioteket för att logga händelser och lagra data i sessionStorage.

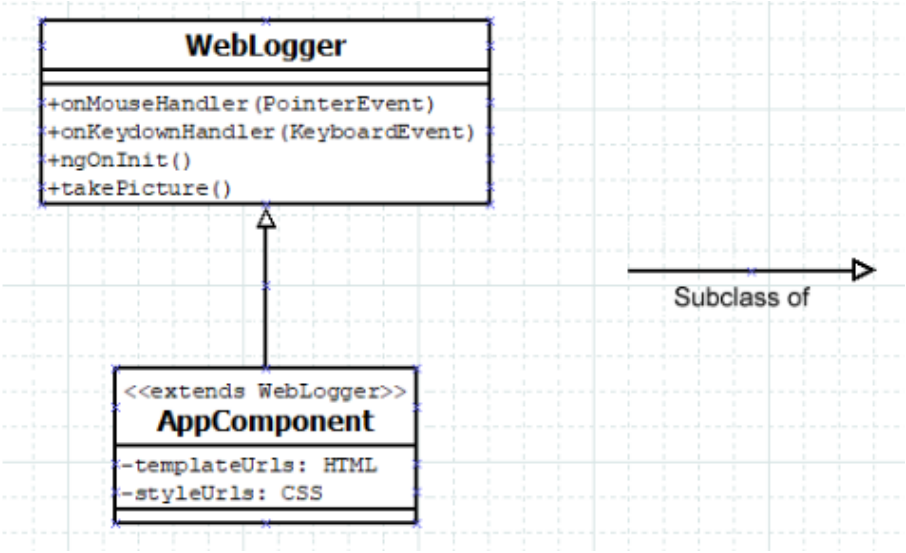

Figure 4.2: AppComponent ärver WL-Bibliotekets funktioner

Dessa fyra funktionerna benäms: onMouseHandler(), onKeydownHandler(), ngOnInIt() och takePicture(). ngOnInIt körs först när användaren besöker en webbapplikation och konfigurerar WL-biblioteket genom att deklarera alla variabler som skall finnas i sessionStorage.

I figur 9 visas en användares webbapplikation som omges av WL-biblioteket. WL-biblioteket fångar upp händelser från tangentbord och mus som sedan lagrar denna data till sessionStorage. Beroende på vilken händelse som sker körs antingen en av två funktioner i WL-biblioteket, onMouseHandler() eller onKeydownHandler(). DOM kommer sedan se till att händelser skickas vidare ner i DOM-trädet till den existerande webbapplikationen.

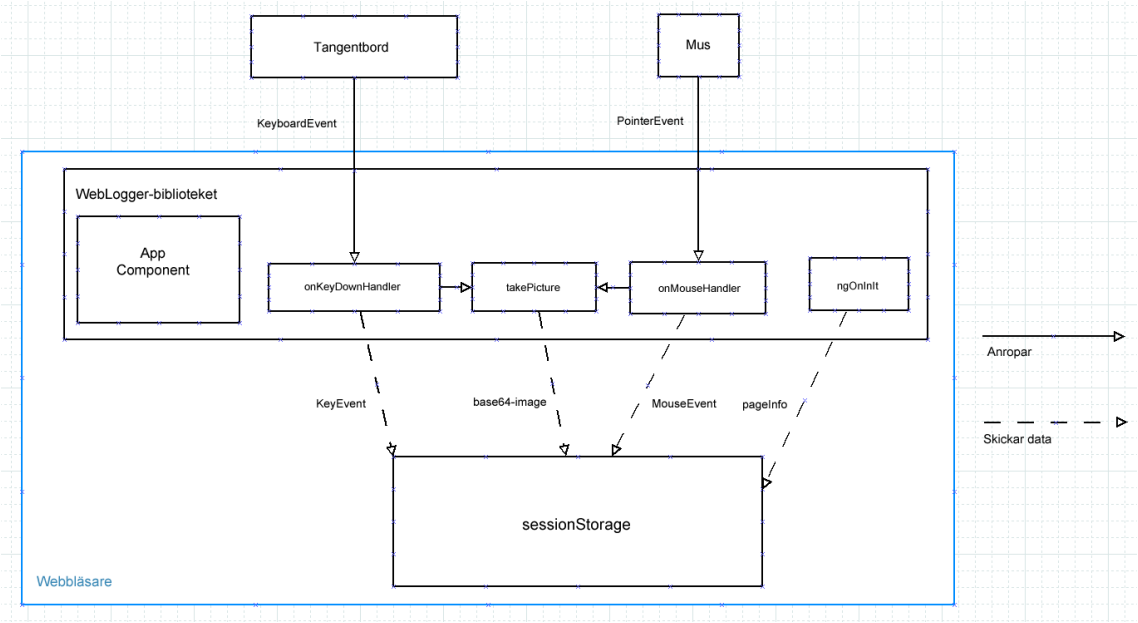

Figure 4.3: AppComponent är webbapplikationen som loggas och fångar upp handlingar från tangentbord och mus skickas

I händelse av ett musklick körs onMouseHandler(). I funktionen onMouseHandler skapas ett MouseEvent-objekt som lagrar nödvändig information från DOM-objektets PointerEvent. Detta gör det möjligt att senare återskapa musklicket. Informationen som lagras är koordinaterna av musklicket samt tidpunkten. Denna informationen lagras i en lista som kallas currentActions i sessionStorage.

```
onMouseHandler(event: MouseEvent) {
  // @ts-ignore
  let lastIn = parseInt(sessionStorage.getItem( key: 'lastIn'));
  // @ts-ignore
  let currentActions = JSON.parse(sessionStorage.getItem( key: 'currentActions'));
  let clickEvent = {
    type: 'ClickEvent',
    xPos : event.x,
    yPos : event.y,
    time : moment.utc( inp: Date.now() - lastIn).format( format: 'H:mm:ss')
  } as ClickEvent
  currentActions.push(clickEvent);
  sessionStorage.setItem('currentActions',JSON.stringify(currentActions));
  this.takePicture();
}
```

Figure 4.4: Kod för onMouseHandler

I fallet av ett tangenttryck körs funktionen onKeyDownHandler. Likt onMouseHandler lagras nödvändig information från DOM-objektets KeyEvent. Exempelvis

vilken tangent samt tidpunkt och lagrar det i currentActions. I slutet av både dessa funktioner körs även funktionen takePicture.

```
@HostListener('document:keydown', ['$event']) onKeydownHandler(event: KeyboardEvent) {
  // @ts-ignore
  let lastIn = parseInt(sessionStorage.getItem( key: 'lastIn'));
  // @ts-ignore
  let currentActions = JSON.parse(sessionStorage.getItem( key: 'currentActions'));
  let keyEvent = {
    type: 'KeyEvent',
    key : event.key,
    time : moment.utc( inp: Date.now() - lastIn).format( format: 'H:mm:ss'),
  } as KeyEvent
  currentActions.push(keyEvent);
  sessionStorage.setItem('currentActions',JSON.stringify(currentActions));
  this.takePicture();
}
```

Figure 4.5: Kod för onKeyDownHandler

takePicture tar en bild på klientens skärm i syfte för att observera hur webbsidan har ändrats efter handlingen. Detta ger utvecklaren en bättre visuell representation av informationen. Detta åstadkoms genom att utnyttja ngx-capture biblioteket i angular. Bilden lagras i form av Base64 image i sessionStorage.

```
takePicture() {
  this.captureService.getImage(document.body,  fullCapture: true)
    .pipe(
      tap( next: img => {
        // @ts-ignore
        let images = JSON.parse(sessionStorage.getItem( key: 'images'));
        images.push(img)
        sessionStorage.setItem('images' , JSON.stringify(images));
      })
    ).subscribe();
}
```

Figure 4.6: Kod för takePicture

4.5.2 Lagring av information

För att lagra informationen fanns en del alternativ mellan cookies, databaser, localStorage och sessionStorage. Informationen kunde inte lagras hos servern på grund av att klienten inte håller konstant kontakt med servern. Det gjorde att alternativet med databaser uteslöts. Dessutom ansågs det utifrån sekretess synpunkter säkrare att lagra datan lokalt. Att lagringen av informationen sker lokalt leder även

till att användaren endast skickar relevant information till utvecklarna. Eftersom sessionStorage lagrar information temporärt under en session vilket uppfyller WL-bibliotekets behov av att kortvarigt lagra information ansågs sessionStorage vara den mest lämpligaste tekniken för lagring av information.

SessionStorage lagrar data i en form av en nyckel-värde databas enligt tabellen nedan.

Nyckelord	Beskrivning
previousPage	URL till senaste länk
lastIn	Tidpunkt där användaren besökte hemsidan
pages	En lista av PageInfos
currentActions	En lista av alla KeyEvent och MouseEvents
images	En lista av alla images

Figure 4.7: Tabell av sessionStorage data

När en händelse uppstår, exempelvis användaren har klickat på en tangent eller med musen, läggs denna händelse in till currentAction samtidigt som den lagrar en bild av webbsidan till images. När användaren lämnar eller navigerar till en annan webbsida skapas pageInfo där datan från currentActions och images läggs in till objektet. PageInfos lagras sedan in till pages vilket är en nyckel i sessionStorage. Pages består av en lista av pageInfo som innehåller all data som behövs att återskapa alla händelser användaren har utfört i webbapplikationen. Denna listan kan skickas till en utvecklare för att granskas.

```
let pageInfo = {
    pageUrl: previousPage,
    timeSpent: lastIn - Date.now(),
    actions: currentActions,
    images: images,
} as PageInfo
```

Figure 4.8: Kodsnutt kring skapandet av objektet pageInfo

Data som loggas av WL-bibliotekets representeras av tre klasser PageInfo, MouseEvent och KeyEvent som i TypeScript kallas för interfaces. PageInfo innehåller data om länken som användaren befinner sig i. Några exempel på typer av information som PageInfo innehåller är URL länken, tidpunkt när klienten besökte webbsidan, bilder och en lista av Mouse- och KeyEvent som representerar alla händelse som användar-PCn har utfört.

```
export interface PageInfo {
  pageUrl : string;
  timeSpent : string;
  actions : [];
  images : [];
}
```

Figure 4.9: Kod för PageInfo

ClickEvent är en klass som innehåller all den nödvändiga data kring musklickningar. Det innehåller bland annat positioner i form av X och Y koordinater och vilken tidpunkt av musklicket.

```
export interface ClickEvent {
  type : string;
  xPos : number;
  yPos : number;
  time : string;
}
```

Figure 4.10: Kod för klassen ClickEvent

KeyEvent representerar knapptryckning på tangentbord. Likt MouseEvent innehåller KeyEvent tidpunkten av knapptryckningen och även information om vilken knapp som har tryckts. Detta inkluderar även kombinationer av olika knapptryckningar som exempelvis SHIFT, ALT eller CTRL.

```
export interface KeyEvent {
  type: string;
  key: string;
  time : string;
  ctrl : boolean;
  alt : boolean;
  shift : boolean;
}
```

Figure 4.11: Kod för klassen KeyEvent

4.6 WebLogger-Applikation

För att presentera den loggade datan används en separat webbapplikation, WL -Applikationen, utvecklad i Angular. Den lagrade datan skickas från användar-PC till utvecklar-PC som JSON. NodeJS används för att starta WL-Applikationen och skapar en lokal server som utvecklaren har åtkomst till. WL-Applikationen innehåller tre webbsidor benämnda AppComponent Index, AppComponent Pages och AppComponent Details, se figur 18.

På webbapplikationen kommer man först till startsidan Index. På Index-sidan visas en textruta som kräver den lagrade datan från WL-bibilioteket i JSON-format. Efter

den lagrade informationen har införts navigeras användaren till webbsidan Pages. Pages visas en lista över alla pageInfos lagrat av WL-biblioteket. Varje individuell pageInfo innehåller information om en specifik webbsida som Details använder för återskapa händelser.

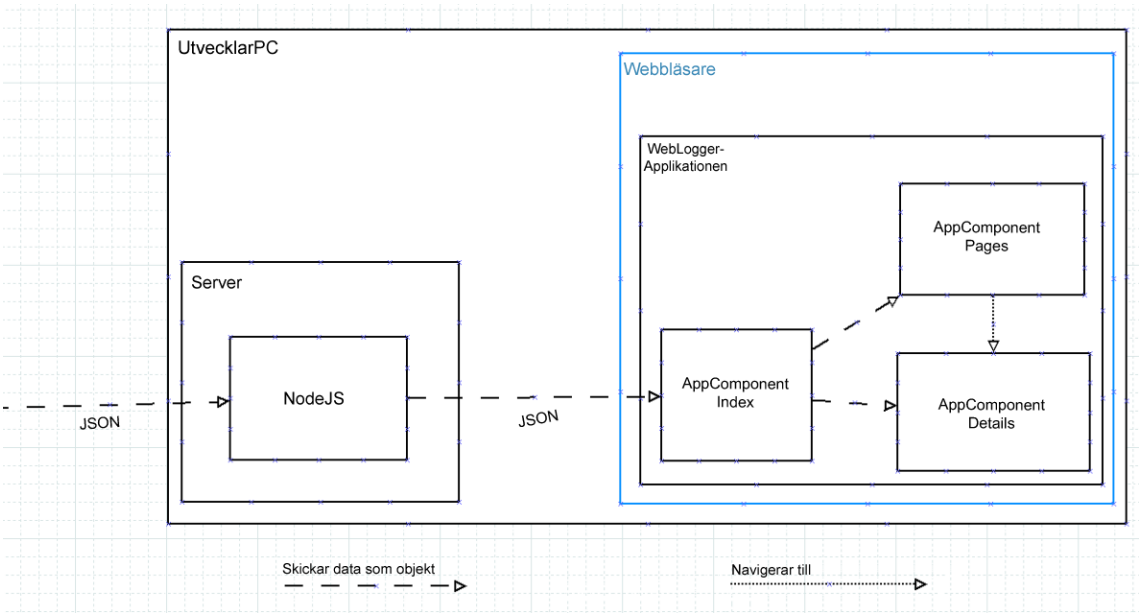

Figure 4.12: NodeJS startar en server med WL-applikationen som tar emot en JSON fil med lagrad data.

I Details återskapas användarens händelser genom att applikationen går igenom den lagrade datan och skapar HTML element specifik för varje händelse. Ett användargränssnitt skapas även som justerar vilka HTML element som visas vid en viss tidpunkt. På detta vis kan utvecklaren kontrollera förloppet av händelser. Användargränssnittet består av en tidslinje som innehåller varje händelse som användaren har utfört. Varje steg i tidslinjen visar tid som har passerat sen händelsen, vad för typ av händelse som har skett och bild på webbsidan efter händelsen. I figuren nedan visas exemplet av ett användargränssnittet på en användare som har klickat på tangenten a. JavaScript används för att lägga till ytterligare funktioner som tillåter utvecklaren att gå vidare till nästa eller föregående händelse. Utvecklaren kan även flytta till nästa eller föregående sida för att se händelser på en annan webbsida. HTML och CSS används för designen av användargränssnittet.

Figure 4.13: Användargränssnittet i WL-applikationens Details-sida

4.7 Driftsättning av WebLogger-biblioteket

För att integrera WL-biblioteket med en existerande webbapplikation behöver AppComponent från den webbapplikation ärva WL-bibliotekets funktionaliteter enligt figur 20.

```
export class AppComponent extends WebLogger {

  /* Funktioner på existerande app */

}
```

Figure 4.14: En webbapplikations AppComponent som ärver WL-bibliotekets funktionaliteter.

Därefter behöver HTML-sidan vara omgiven av en div som observerar med hjälp av en #screen tagg. Ett div är en tagg i HTML som används för att definera en sektion. Sektionen fungerar som en behållare för flera andra HTML element. Div sektionen behöver även exekvera en funktion från WL-bibliotekets som fångar upp alla DOM-APIns PointerEvents.

```
<div #screen (click)="onMouseHandler($event)">

  <!-- HTML-sidan på existerande app -->

</div>
```

Figure 4.15: En HTML-sida integrerad med WL-biblioteket.

4.8 Driftsättning av WebLogger-applikationen

För att starta WL-applikationen behöver applikationen laddas ned och köras via NodeJS kommandot ng serve.

5

Resultat

5.1 WebLogger-Biblioteket

WL-biblioteket är implementerat som ett antal funktioner som samlar in värden utifrån användarhändelser i sessionStorage

Key	Value
pages	[]
previousPage	http://localhost:4200/page1
currentActions	[{"type":"ClickEvent","xPos":134,"yPo...
lastIn	1653431897000
images	["...

Figure 5.1: Innehållet i sessionStorage efter att WL-biblioteket fångar en användarhändelse. I detta fall en mushändelse.


```
[{"pageUrl":"http://localhost:4200/page1","timeSpent":"0:00:08","actions":
[{"type":"ClickEvent","xPos":110,"yPos":126,"time":"0:00:01"},
{"type":"KeyEvent","key":"Shift","time":"0:00:01","ctrl":false,"alt":false,"shift":true},
{"type":"KeyEvent","key":"H","time":"0:00:01","ctrl":false,"alt":false,"shift":true},
{"type":"KeyEvent","key":"e","time":"0:00:02","ctrl":false,"alt":false,"shift":false},
{"type":"KeyEvent","key":"l","time":"0:00:02","ctrl":false,"alt":false,"shift":false},
{"type":"KeyEvent","key":"l","time":"0:00:03","ctrl":false,"alt":false,"shift":false},
{"type":"KeyEvent","key":"o","time":"0:00:03","ctrl":false,"alt":false,"shift":false},{"type":"Clic
{"type":"KeyEvent","key":"Shift","time":"0:00:05","ctrl":false,"alt":false,"shift":true},
{"type":"KeyEvent","key":"W","time":"0:00:05","ctrl":false,"alt":false,"shift":true},
{"type":"KeyEvent","key":"o","time":"0:00:05","ctrl":false,"alt":false,"shift":false},
{"type":"KeyEvent","key":"r","time":"0:00:05","ctrl":false,"alt":false,"shift":false},
{"type":"KeyEvent","key":"l","time":"0:00:05","ctrl":false,"alt":false,"shift":false},
{"type":"KeyEvent","key":"d","time":"0:00:06","ctrl":false,"alt":false,"shift":false}],"images":
["
SJIFWaYEisIBayxKZF8hkYpkMtfjeM4SpBBErGku1MogtOGBpAYsJxVJsBVERw2DIW//dt2/PPdzkDrk399zv+Zy1uiTJGfb38/
ECBAgQIECAAAECjYDSYxEIECBAgQIECAAAAQJKjx0gQIECAAAAQJKjx0gQIECBAgQIECAAAAQJKjx0gQIECBAgQIECAAAAQJ
0avHfuaZT5VTT1nTHO+GDJcvf6YsWvTken1vu+1FZc89f2FS/H0oAQIECBAgQIAAAgakkMKIzPbff/vNS/7G+fPkLyjbbTCv113v
v/upmTVm5//61ZcGCJ8v8+Vustzy2WWu57Cw4bXEaSf6JmoP3JUCAAAECBAgQIDBVBMZUemq4+g/vFSueKddc84Ky447TN3neWi
c+rz0TdMUVLygLF24xrH99r+uvf6b84R9uOexzPYEAAQIECBAgQIBAksCYS0/7j+7JusxqY0rPZB/7+s5Cjab0tJfvPfjgc+stT
nkGovOStE6e9jst3d+/GcnZi40pPZN97K1BzbBq1XOjvrytfX13js7vC9Xn1EK6007TBy4HbF9Xf3/WrGnN5XV33372+W2Xyk2t
0Vzq1nnZ1tZbl+ZmCLNnb9b84779Lsuf/MlWzWVZtfCceupT5bDDtmheu6FLujpjjbX09MKx1xzduYca2XAWnWes6us7v3dV//v
gQIAAgSkiMOLSU29c0PlY35mB9mzDvHmbN5dd1Ue98UD7Zfz2H9xtCWpLSLfXcHeHG03p6bVjr1lr7npp2oa+jzNc6Rnqe1WdZ8
```


Figure 5.2: Innehållet i sessionStorage i JSON format

5.2 WebLogger-Applikationen

5.2.1 Pages

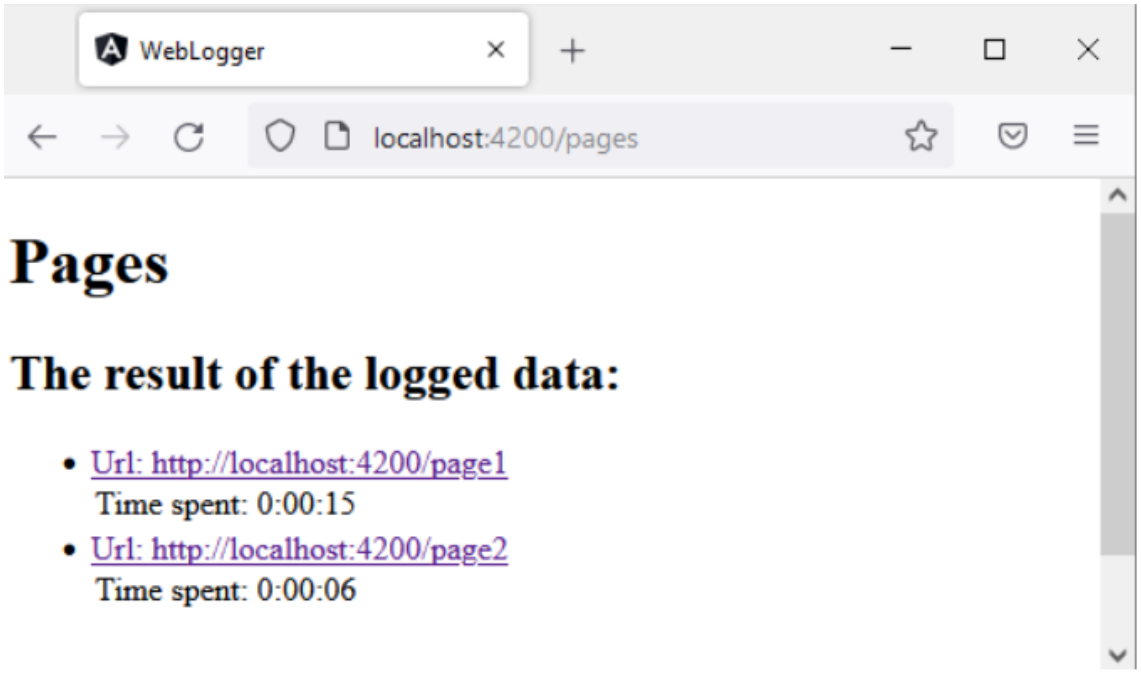

Figure 5.3: Exempelvy på Pages

Figur 25 visar webbsidan Pages i WL-Applikationen. Sidan innehåller information om vilka sidor användaren har besökt och även hur länge användaren har varit på webbsidan. Länkarna på Pages-sidan leder till Details sidan som innehåller fler detaljer om användarens händelser.

5.2.2 Details

På details-sidan presenteras resultatet av en enskild loggning. Utvecklaren kan mha en tidslinje spela upp webbsidan som användarens besökt och även med ett användargränssnitt stegvist återskapa användarens händelser. Bilden på webbsidan uppdateras efter tidslinjen och visar placeringen av musklick som har skett. Användargränssnittet visar även nästa tangenttryck som har skett med ytterligare funktionaliteter som att exempelvis gå vidare till nästa eller föregående händelse. För att se en användarens handlingar på en annan webbsida i webbapplikationen finns även funktionaliteten att gå vidare till föregående eller nästa webbsida.

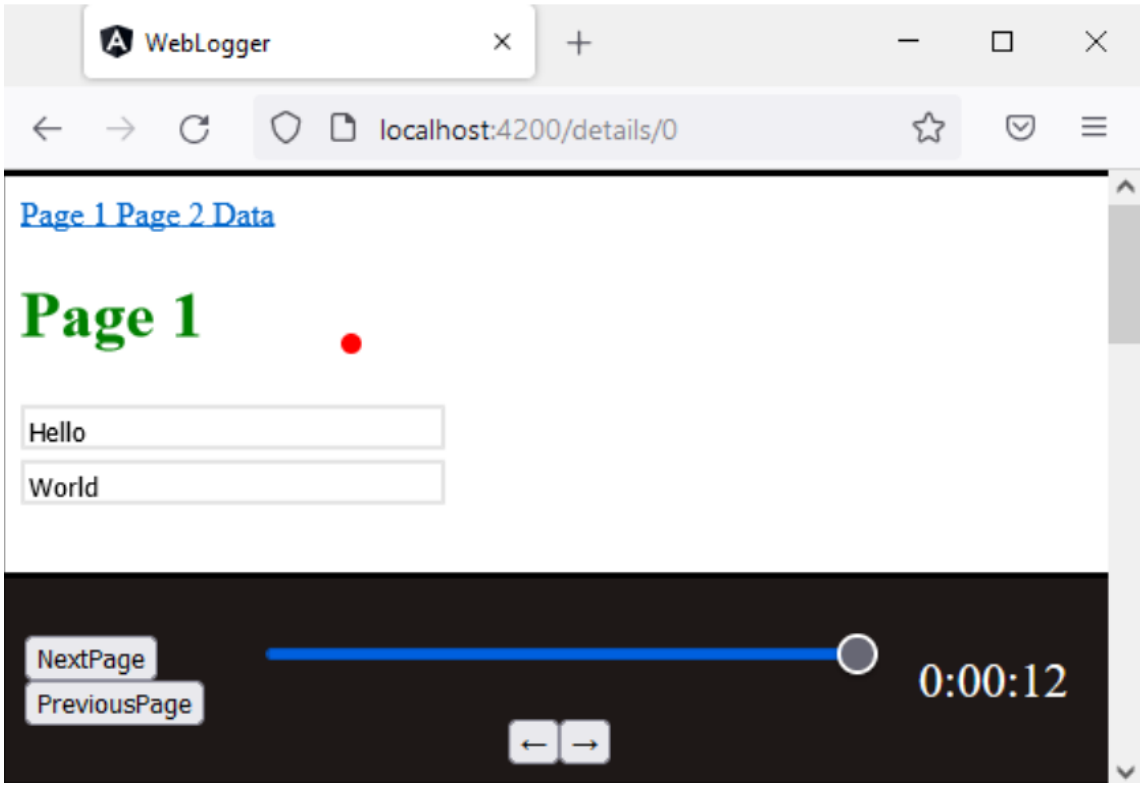

Figure 5.4: Återspelning på musklick i details

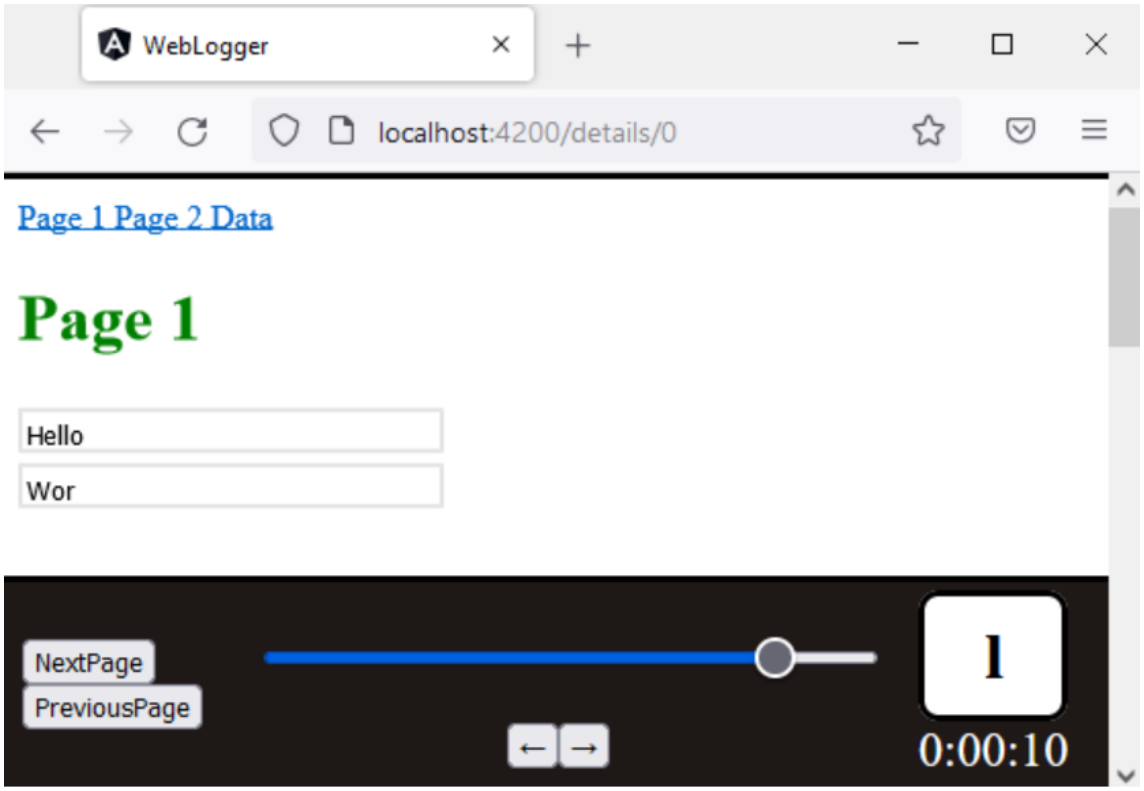

Figure 5.5: Återspelning på tangentklick i details

5.3 Kvalitativ undersökning av kommunikation

Eftersom att det är svårt att mäta hur mycket kommunikationen har förbättrats i praktiken så har en kvalitativ uppskattning av informella tester utförts. Detta utfördes genom att ta kontakt med företaget Diadrom som föreslog projektidéen och se om företagets mål på projektet hade uppnåtts. Företaget erbjöd även sina tankar och åsikter om projektet hade uppfyllt sitt syfte. Utvecklare från företaget ansåg lösningen som lovande och att lösningen skulle vara mest användbar i att hitta fel som var svåra att återskapa på utvecklarnas datorer. Diadrom uttryckte även att en industrialiserad version av lösningen skulle med stor säkerhet ge många fördelar för utvecklare av webbapplikationer.

6
Slutsats

Syftet med detta projektet har varit att förbättra felrapportering mellan användare och utvecklare i samband med felsökning och rättning av webbapplikationer. Utifrån målsättningen har mjukvara tagits fram. Mjukvaran resulterar i fler detaljer presenterat i ett format som tillåter utvecklaren att återskapa händelsen som hen finner lämpligt. Med detta och de informella undersökningar som har utförts anser projektgruppen att kommunikationen har förbättrats. Projektgruppen anser därför att projektet har uppfyllt sitt syfte.

6.1 Diskussion

Planeringen av projektet har inte gått felfritt med flera problem angående tidsuppskattningar. Problemen förekomm främst i början eftersom användandet av flera nya verktyg och språk tog mer tid att lära än planerat. Projektgruppen hade även planer att undersöka den etiska aspekten av projektet men på grund av tidsbrist valde projektgruppen att fokusera på den tekniska aspekten istället. Projektgruppen implementerade en del funktionaliteter i åtanke till den etiska aspekten för att skydda användarens integritet, exempelvis att lagra informationen lokalt hos användaren och även tillåta utvecklaren att exkludera webbsidor från att loggas, exempelvis inloggningssidor eller betalningssidor.

På grund av att hanteringen av HTTP-requests exekveras hos server-sidan har projektgruppen undvikit att implementera en funktionalitet som överför data mellan slutanvändare och utvecklare. Detta är för att det kan resultera i svårigheter med en av mjukvarans krav, vilket är att mjukvara ska vara enkel att integrera med minimala förändringar.

Projektgruppen har även av tidsbrist inte lyckats att implementera en funktionalitet som gör det möjligt för användaren att granska den loggade datan innan det skickas till en utvecklare. Detta beror på att granskning av den lagrade datan kräver WL-Applikationen vilket endast utvecklare har tillgång till. Detta kan lösas på server-sidan genom att införa WL-Applikationen som en extra sida på webbapplikationen men eftersom projektgruppen har främst arbetat inom klient-sidan har inte detta testats.

En möjlig förbättring till biblioteket är relaterat till loggning och lagring av bilder. För att logga bilder kunde en teknik som heter Rolling cache utnyttjas. Rolling cache

var ett alternativ för att förbättra WL-biblioteket ifall sessionStorage möjligen får slut på kapacitet. SessionStorage kan maximalt spara 5MB data vilket leder till en risk då bilder tar upp mycket minne. För att undvika detta kan man exempelvis logga den senaste minuten och skriva över den äldre informationen istället för att lagra hela användarens session.

Källor

[1] J. Johnson. (2022). "Worldwide digital population as of april 2022", [Online]. Available: https://www.statista.com/statistics/617136/digital-population-worldwide/ (visited on 06/05/2022).

[2] (2020). "6 procent av befolkningen riskerar ett digitalt utanförskap som sällan- eller ickeanvändare av internet", Internetstiftelsen, [Online]. Available: https://svenskarnaochinternet.se/rapporter/digitalt-utanforskap-2020/internetutveckling-och-uppkoppling/ar-2020-q1-anvander-96-procent-av-svenskarna-internet/ (visited on 06/05/2022).

[3] M. Patterson. (2022). "Why customers ask vague questions (and what to do about it)", [Online]. Available: https://www.helpscout.com/blog/vague-questions/ (visited on 06/21/2022).

[4] (2021). "The importance of feedback", actiTIME, [Online]. Available: https://www.actitime.com/project-management/importance-of-feedback (visited on 06/21/2022).

[5] (). "What do client side and server side mean? | client side vs. server side", Cloudflare, [Online]. Available: https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/ (visited on 05/30/2022).

[6] A. S. (2022). "What is html? hypertext markup language basics explained", [Online]. Available: https://www.hostinger.com/tutorials/what-is-html (visited on 05/30/2022).

[7] (2022). "What is javascript?", Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (visited on 05/30/2022).

[8] (). "What is typescript?", typescript, [Online]. Available: https://www.typescriptlang.org/ (visited on 05/30/2022).

[9] (2022). "Working with json", Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON (visited on 05/30/2022).

[10] (2022). "Window", Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window (visited on 06/05/2022).

[11] (2022). "Pointerevent", Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/PointerEvent (visited on 06/05/2022).

[12] (2022). “Keyboardevent”, Mozilla, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent (visited on 06/05/2022).

[13] (2018). “Http cookies”, GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/http-cookies/ (visited on 05/30/2022).

[14] (). “About node.js”, NodeJS, [Online]. Available: https://nodejs.org/en/about/ (visited on 05/30/2022).

[15] (2022). “Launching your app with a root module”, Angular, [Online]. Available: https://angular.io/guide/bootstrapping (visited on 06/05/2022).

[16] (2022). “Ngx-capture”, npm, [Online]. Available: https://www.npmjs.com/package/ngx-capture/ (visited on 06/05/2022).

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige
www.chalmers.se

CHALMERS