

Bachelorprosjekt 2018

OsloMet – storbyuniversitetet



Gruppe 19

Ali Arfan	s301599
John Husfloen Håvardstun	s305053
Kent Erlend Bratteng Knudsen	s300358
Shamil Magomadov	s305052



Institutt for Informasjonsteknologi

Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo
Besøksadresse: Holbergs plass, Oslo



PROSJEKT NR.

19

TILGJENGELIGHET

Offentlig

BACHELORPROSJEKT

Telefon: 22 45 32 00

HOVEDPROSJEKTETS TITTEL PosPay Monitor	DATO 23.05.2018
	ANTALL SIDER / BILAG 119 / 10
PROSJEKTDELTAKERE Ali Arfan s301599@oslomet.no John Husfloen Håvardstun s305953@oslomet.no Kent Erlend Bratteng Knudsen s300358@oslomet.no Shamil Magomadov s305052@oslomet.no	INTERN VEILEDER Torunn Gjester Torunn.Gjester@oslomet.no

OPPDRAGSGIVER PayEx Norge AS	KONTAKTPERSON Dani Alexander Berentzen dani.alexander.berentzen@payex.com
--	---

SAMMENDRAG Webapplikasjon for overvåkning av betalingstransaksjoner i sanntid.

3 STIKKORD
Webapplikasjon
Betalingsovervåkning
Java Spring

Forord

Denne rapporten dokumenterer arbeidet og produktet som ble utviklet i forbindelse med bacheloroppgaven ved OsloMet - storbyuniversitet våren 2018.

Oppdragsgiver i dette prosjektet er PayEx Norge AS, som vi kom i kontakt med gjennom direkte henvendelse. Av PayEx ble vi tilbudt oppgaven med å utvikle en webapplikasjon for overvåking over deres transaksjonssystemer i sanntid.

Applikasjonen som ble utviklet finnes på: <http://hp2.vlab.cs.hioa.no/>. Testbrukeren for innlogging er "user" og "password".

Vi ønsker å takke vår veileder ved universitetet, Torunn Gjester, for gode råd og tips gjennom denne prosessen og for å virke som en motivator for oss. En stor takk rettes til teamleder Dani Berentzen og systemutvikler Even Holthe hos PayEx for å tilrettelegge og gi oss muligheten til å utføre denne oppgaven og all bistand de har bidratt med underveis.

Innholdsfortegnelse

Innledning.....	6
Bakgrunn	6
Effektmål.....	8
Resultatmål.....	8
Arbeidsmetodikk.....	9
Teknologier	10
Produktdokumentasjon.....	14
Backend.....	14
Systemarkitektur for backend.....	16
Datamodell	17
Websocket-grensesnittet	19
Generering og bevaring av fiktive data	19
Fiktive data.....	24
Feilhåndtering og logging.....	26
Sikkerhet	27
Spring Security Filter Chain	27
Autentiseringsfilteret	28
Autorisasjonsfilter	30
Henting og lagring av brukere.....	31
Token	33
Frontend	35
Systemarkitektur for frontend	36
Innloggingssiden	37
Dashboard	38
Merchants	39
Visning av feilmeldinger	40
Rammeverk.....	41
Vue.js	41
Komponenter	41
Mounted.....	42
Computed	43
Watch	43
Vue Router	44
Vuex	46
Behandling av mottatt transaksjonsdata.....	48

D3	50
Bootstrap.....	51
Systemet i helhet	53
Konklusjon	55
Funksjonelle krav	55
Ikke-funksjonelle krav.....	56
Hva vi har lært.....	56
Vedlegg	57
Kilder	118

Innledning

Bakgrunn

PayEx Norge AS er et nordisk konsern med cirka fem hundre medarbeidere fordelt i Sverige, Norge, Danmark og Finland. Siden starten på Gotland i 1972 har selskapet jobbet kontinuerlig med utvikling og salg av betalingstjenester. De kan i dag tilby betalingsløsninger for internett, mobil, fysisk handel og administrative tjenester innen fakturering, regnskap og inkasso. I mai 2017 ble PayEx kjøpt opp av Swedbank og er nå et heleid datterselskap. I PayEx sin kundeportefølje finner vi blant annet 7-Eleven, Narvesen, NSB og Clas Ohlson. PayEx behandler i gjennomsnitt drøyt 1 million transaksjoner hvert døgn. Selskapets visjon er at alle før eller siden skal betale via PayEx.

Fysiske handelsløsninger innebærer levering av fysiske betalingsterminaler som blir brukt av kunden. Det blir levert ut og installert kortterminaler som kan brukes til å utføre betalinger. De er enten integrert med nåværende kasseløsning som brukes i butikken, eller frittstående mobile terminaler som kan brukes hvor som helst. Dette har teamet PosPay ansvaret for. De programmerer, leverer og setter opp de fysiske betalingsløsningene og tjenester rundt disse.

Hver gang det utføres en betaling med en kortterminal, sendes metadata med beskrivelse av transaksjonen til PayEx i henhold til *ISO 20022*. Dette er en åpen internasjonal standard som beskriver grunnprinsippene på hva en betalingstransaksjon skal inneholde. Deretter blir transaksjonen behandlet hos PayEx før den sendes videre til sine respektive innløser¹.

For å se antall betalingstransaksjoner som har blitt utført i løpet av dagen, benytter PayEx seg av en egenutviklet løsning de har døpt PosPay Monitor. Denne applikasjonen kan vise gjennomsnittlig antall transaksjoner per sekund, hvor lang tid siste transaksjon tok, hvor lang tid som har gått siden siste transaksjon ble utført og totalt antall utførte transaksjoner.

Ulempen med dagens løsning er at den mangler mekanismer for brukerautentisering og -autorisering, og kan derfor ikke brukes til å filtrere data basert på tilgang. Dette begrenser mulighetene PayEx har for å tilgjengeliggjøre applikasjonen for flere brukere. Av den grunn må man i dag være pålogget det interne nettverket til PayEx for å kunne bruke applikasjonen.

Et annet aspekt ved PosPay Monitoren i dag, er at den er skrevet i et eldre rammeverk. Ønsker man å tilpasse applikasjonen andre enheter og plattformer, innebærer det en nær total omskriving. Nåværende PosPay monitor er skrevet i Java 5 og er veldig dårlig dokumentert. Derfor ønsker PayEx at vi skal utvikle en web-basert løsning sammen med et API som også gjør det mulig å enkelt utvikle andre typer klienter, som apps, i fremtiden.

¹ Innløser er banker og andre betalingsprosesseringstjenester.

Den nye løsningen må kunne hente innhold fra flere kilder enn betalingsautomater, som for eksempel betalinger utført på internett. Løsningen må være fleksibel, slik at den trenger minimalt med eller ingen konfigurasjon for å tegne grafer av nye kunder. Applikasjonen skal gi brukeren informasjon om statusen på betalingstransaksjonen, samt detaljert informasjon om transaksjonene som utføres av spesifikke kunder oppdragsgiveren har. For mer informasjon om kravene rundt applikasjonen, se [Vedlegg C: Kravspesifikasjon](#)

Effektmål

PayEx har som mål å lage en ny versjon av sin betalingstransaksjonsmonitor som skal erstatte dagens løsning. Den skal brukes av support- og vaktpersonell og kan hjelpe til med å identifisere sentrale driftsproblemer enten hos PayEx eller hos tredjepart. Ved å vise tydelig hvor eventuelle problemer er hjelper det i feilsøkingsituasjoner og vil kunne bedre opptiden på systemene ved at feil identifiseres så tidlig som mulig.

Den nye monitoren vil på sikt også kunne automatisk oppdage avvik eller stopp i transaksjonstrafikken for også mindre kunder. Et slikt automatisk varslingsystem vil kunne redusere antall henvendelser til support og kundetilfredsheten vil øke ved at feil blir utbedret før kunden merker alvorlet.

Resultatmål

Oppdragsgiveren ønsker en fullstendig webapplikasjon som består av to hoveddeler:

- Backend
 - Autentisering av brukere.
 - Autorisering gjennom bruk av tokens.
 - Skal kunne kjøre flere instanser av backend.
 - Utsending av transaksjonsstatus med universelt format.
 - Oppdagelse av anomalier i transaksjonsbildet.
 - Utsending av feilmeldinger på mail ved avvik.
- Frontend
 - Innlogging og autentisering med rettighetsnivåer for forskjellige brukere.
 - Statusbilde med informasjon over transaksjonene.
 - Feilmeldinger fremkommer tydelig og er informative.
 - Applikasjonen skal kjøre på forskjellige enheter, blant annet mobiltelefon, nettbrett, og store skjermer.

Arbeidsmetodikk

Under planleggingsfasen ble vi enige med PayEx om å jobbe smidig (eng: agile) med Scrum som prosessmetodikk. Arbeidet skulle foregå i PayEx sine lokaler i Kongens Gate 6, tre til fire dager i uken med kjernetid kl. 09-17. Sprintene skulle planlegges på mandager, ha en lengde på to uker med presentasjon av resultatet på fredag i slutten av perioden. Jira² skulle brukes for prosjektstyring.

Fordi PayEx har mange utviklere er det lagt ned retningslinjer for hvordan kode skal produseres. Disse retningslinjene tilfalt også oss og bidro til at vi fikk en god innføring i hvordan det er å jobbe i et profesjonelt miljø. For å gi oss en litt mykere start, var Even Holthe i PayEx satt til å kvalitetssjekke koden og gi oss konstruktiv tilbakemelding på hva som kunne gjøres bedre.

Første fase av prosjektet ble derfor å konfigurere de integrerte utviklingsmiljøene samt klargjøre verktøyene rundt. Dette innebærte at WebStorm ble konfigurert til å ta i bruk ESLint mens IntelliJ IDEA ble satt opp med CheckStyle. Disse nevnte teknologiene sørget for at koden hele tiden oppfylte formateringsstandarden til PayEx. For versjonskontroll av koden benyttet vi oss av GitHub.

Slack ble brukt som kommunikasjonsverktøy mellom PayEx og oss, og for å få notifikasjoner fra GitHub og TeamCity. Sistnevnte er en server for kontinuerlig integrasjon, som automatisk kompilerer koden og kjører tester, slik at hele teamet er oppdatert på statusen til kodebasen. For utfyllende informasjon om arbeidsprosessen, se [Vedlegg B: Prosessdokumentasjon](#).

² Se [prosessdokumentasjon](#) for ytterligere informasjon om Jira

Teknologier

Frontend	
WebStorm	Foretrukket som integrert utviklingsmiljø (eng: IDE) fordi applikasjonen har veldig bred støtte for webteknologier og omfattende kodeassistanse. Dette letter på ferdigstillingen av kode og bidrar til å effektivisere arbeidet.
Node.js	JavaScript-miljø utenfor nettleseren. Benyttes for å bygge frontend og kjøring av server lokalt på utviklingsmaskinene. Lar hver utvikler kontinuerlig teste endringene sine i isolasjon.
Npm	Pakkesystem for Node.js. Brukes til å holde orden på tredjepartskomponenter, -rammeverk og -biblioteker man ønsker å bruke, samt å kjøre byggeprosesser og lignende.
Babel	JavaScript-kompilator. Lar oss benytte moderne EcmaScript ³ -versjoner ved at koden <i>transpileres</i> (oversettes) til JavaScript-kode som kan kjøres i nettleserne.
Webpack	Prosesserer alle ressurser i frontend og pakker disse sammen til HTML, CSS, JS og andre statiske ressurser for bruk i produksjon. Bruker plugins til å håndtere en mengde forskjellige ressurstyper.
HTML & CSS	Naturlig del av webapplikasjoner.
Bootstrap	CSS-rammeverk som tilbyr et solid grunndesign og mange hjelpere for å effektivisere utvikling av responsive sider.
Feather	Tilbyr et stort utvalg av ikoner i SVG-format.
JavaScript	Lar oss manipulere data i nettleseren. For en applikasjon som vår, hvor hele formålet er å vise data i sanntid, er JavaScript en nødvendighet - uten ville vi vært nødt til å sende en helt ny side fra backend med data for hver oppdatering.
D3	Fleksibelt JavaScript-bibliotek for å lage datavisualiseringer i

³ EcmaScript er en standard fra Ecma International; JavaScript er en implementasjon av EcmaScript i nettleserne.

	SVG (vektorgrafikk i XML-format).
Vue.js	JavaScript-rammeverk for å utvikle komponentbaserte og skalerbare webapplikasjoner. Gjør det enklere å separere ansvarsområder i koden og gjenbruke funksjonalitet
Vuex	Tilbyr sentralisert tilstandshåndtering for Vue. Lar forskjellige deler av applikasjonen kommunisere med hverandre uten å ha direkte kjennskap til hverandre, og fungerer som en opphavskilde (eng: <i>Single source of truth</i>) som komponenter deriverer sin tilstand fra.

For informasjon om oppsett av utviklingsmiljø for frontend, se [Vedlegg G: Oppsett av utviklingsmiljø for frontend](#).

Transport	
JSON	Et enkelt, men fleksibelt format for datautveksling. Foretrekkes fordi det er det naturlige formatet for JavaScript - ethvert gyldig JSON-objekt er et gyldig JavaScript-objekt (herav navnet JavaScript Object Notation).
JSON Web Token	En åpen standard som definerer en måte å utstede en verifiserbar access token på. Nødvendig for å gjøre autentiseringen stateless, som forenkler prosessen med å gjøre applikasjonen skalerbar.
Websocket	Brukes til direktestrømming av data mellom frontend og backend. Data utveksles i JSON.

Backend	
IntelliJ IDEA	Foretrukket integrert utviklingsmiljø for Java-applikasjoner. Har svært god kodeassistanse og -analyse ⁴ .
Java	Objektorientert programmeringsspråk. Foretrukket på grunn av sin plattformuavhengighet og sterke posisjon i Enterprise selskaper. Kan utvikle funksjonsrike applikasjoner på kort tid.
Spring Framework	Et funksjonsrikt og modulært rammeverk. Ofte brukt i utviklingen av store bedriftsapplikasjoner. Øker produktivitet ved at klassene blir mer modulære slik at de kan lettere gjenbrukes, utvides og erstattes.
Spring Boot	Tilbyr ferdigkonfigurert oppsett av Spring etter <i>convention over configuration</i> -prinsippet. Gjør det enklere å komme raskt i gang med Spring og tilbyr bl.a. TomCat webserver for å kjøre applikasjonen som en webtjeneste.
Problem	Et tredjeparts rammeverk tatt i bruk for å forenkle utsending av feilmeldinger og gi meldingene et standardisert format ⁵ .
Maven	Håndterer avhengigheter og byggeprosess i Java-prosjekter. Gjør det enkelt å sette opp prosjektet på nye maskiner og sørger for at alle har samme versjoner av avhengighetene.

For informasjon om oppsett av utviklingsmiljø for backend, se [Vedlegg F: Oppsett av utviklingsmiljø for backend](#).

For informasjon om oppsett av servermiljø, se [Vedlegg E: Oppsett av servermiljø](#).

⁴ Cheptsov, A. (2016). "Enjoying Java and Being More Productive with IntelliJ IDEA." Retrieved 21 May, 2018, from <https://blog.jetbrains.com/idea/2016/03/enjoying-java-and-being-more-productive-with-intellij-idea/>

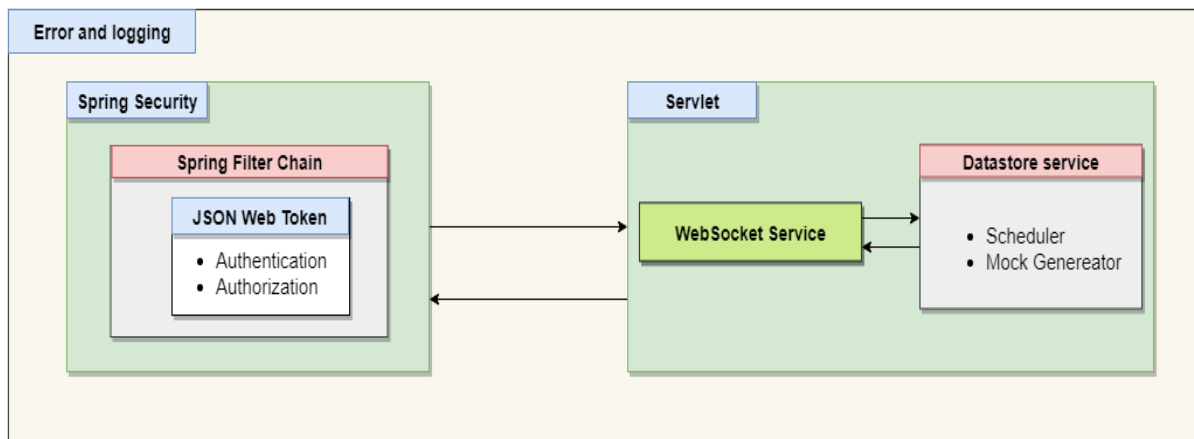
⁵ M. Nottingham, A., E. Wilde (2016). "Problem Details for HTTP APIs." *Request for Comments: 7807* Retrieved 22 May, 2018, from <https://tools.ietf.org/html/rfc7807>

Testing	
WebDriverIO	Node-bindinger til Selenium WebDriver, et automatiseringsrammeverk for testing av nettsider i faktiske nettlesere. Lar oss teste frontend fra et JavaScript-miljø, som er mest naturlig.
ChromeDriver	Implementasjon av WebDriver-grensesnittet for å styre Google Chrome.
Spring MVC Test	Brukt i backend for å skrive tester og generere mock-tjenester som for eksempel websocket, HTTP requests og så videre.

For utfyllende informasjon om testingen, se [Vedlegg A: Testdokumentasjon](#).

Produktdokumentasjon

Backend



FIGUR 1: OVERORDNET ILLUSTRASJON AV BACKEND

Det ble tidlig bestemt at frontend og backend skulle være uavhengige av hverandre. Backend vil da fungere som en ren ressursserver. Dette ville gi en enklere struktur å forholde seg til med tanke på senere utvikling av andre klienter, for eksempel apps.

Oppdragsgiver ønsket også mulighet for at brukerautentisering kunne skje mot en hvilken som helst instans hvis det skulle være nødvendig å skalere horisontalt – det vil si kjøre flere parallelle instanser av backend. Dette innebar at vi måtte finne en løsning der autentiseringen kunne desentraliseres fra én spesifikk instans. Samtidig måtte løsningen sørge for at ressursserveren kunne verifisere autorisasjonen til en bruker. Dette vil si at autentiseringen må være stateless.

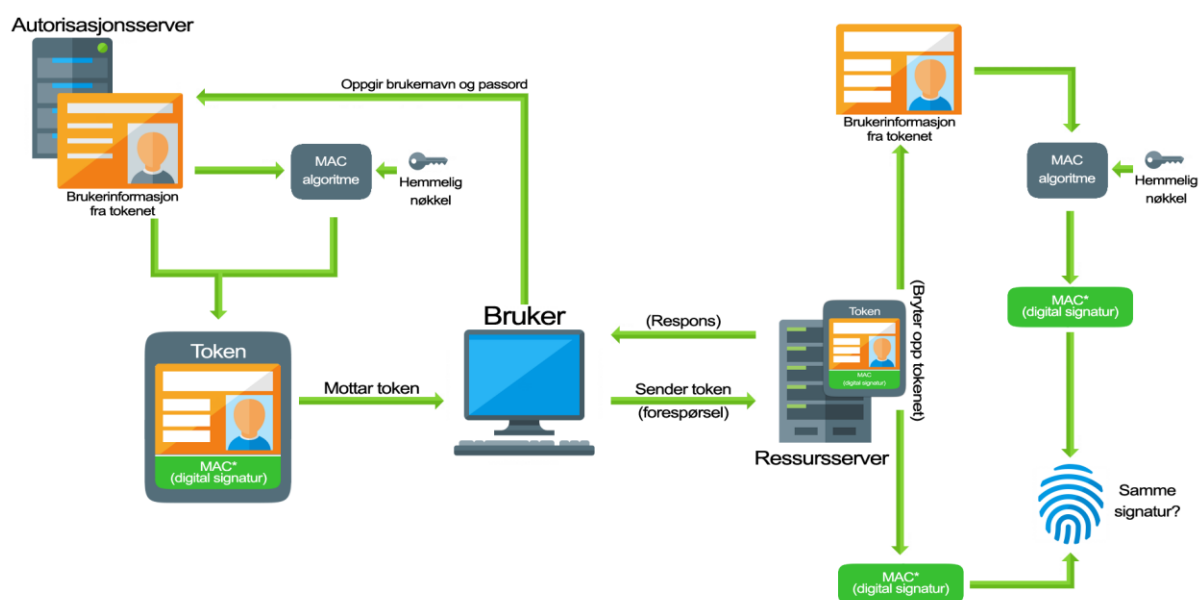
I motsetning vil stateful autentisering si at serveren lagrer informasjon om brukerens autorisasjonsnivå etter at brukeren har blitt autentisert, typisk kalles dette en sesjon. Når en bruker etterpå sender en forespørsel må det refereres til denne registreringen. En slik løsning blir fort kompleks når autentiseringen skal være desentralisert og ressursserveren skal kontrollere at en brukers autorisasjon er gyldig. Hvis instansen som mottar forespørselen ikke er den som autentiserte brukeren i utgangspunktet må den på et eller annet vis synkronisere mot de andre instansene, som i seg selv kompliserer skaleringen. Det er med denne grunn at backend ble utviklet med stateless autentisering.

Ønskene fra oppdragsgiver oppnådde vi ved å ta i bruk JSON Web Token (JWT). I stedet for at serveren lagrer informasjonen om brukerens autorisasjonsnivå (og andre nødvendige detaljer), lagres disse i et token som sendes til klienten. Klienten bruker dette tokenet til å identifisere seg ved hver forespørsel til serveren. For å beskytte informasjonen tokenet bærer blir det digitalt signert med en nøkkel kun kjent av PayEx.

Fordi tokenet har en digital signatur, kan ikke påstandene forfalskes. Dette skjer ved at serveren som utdeler tokenet beregner en Message Authentication Code (MAC) på informasjonen i tokenet og legger denne med tokenet. Skulle informasjonen i

tokenet bli endret, vil ikke den digitale signaturen samsvare med informasjonen, og tokenet blir da ansett som ugyldig.

Ressursserver kan derfor stole på informasjonen fordi integriteten er ivaretatt gjennom signaturen. Alt ressursserveren trenger å gjøre er å kontrollere signaturen og tokenet sin gyldighet. På den måten får man separat håndtering av innlogging fra ressursserveren. Dette resulterer i at man kan ha et system bestående av flere ressursservere for å takle store belastninger eller at man kan ha et system som starter flere instanser av ressursserveren ved behov. Serverne som håndterer innlogging kan også settes bort andre steder. Man får en løsning som er mer fleksibel og lar seg skalere bedre enn om man brukte stateful autentisering.



*MAC = Message Authentication Code

FIGUR 2: OVERORDNET ILLUSTRASJON AV DESENTRALISERT AUTENTISERING MED TOKEN

For å implementere JWT i backend måtte vi ha en forståelse av hvordan Spring Security Filter Chain var satt sammen. Dette fordi autentiserings- og autoriseringsprosessen ville bli avhengig av et token, en tjeneste som ikke var innebygget i Spring Security modulen. For å implementere dette manuelt, ble det nødvendig å få en forståelse for hvert filters ansvar før vi kunne legge inn støtte for JWT.

Den viktigste oppgaven til ressursserveren er å tilby brukeren statistikk over transaksjoner utført de siste 60 minuttene. Dette innebærer at transaksjonsbildet deles opp i intervaller som det genereres statistikk på. Når en bruker ønsker å få tilsendt statistisk data, kobler han til backend over en websocket og identifiserer seg med token. Backend registrerer brukeren som en lytter i en observasjonsklasse. Denne klassens funksjon er å sende et signal til alle lytterne etter at de statistiske beregningene er utført. Hoveddelen av beregningene utføres én gang i sekundet.

Signalet indikerer at ny data er tilgjengelig og utløser samtidig en hendelse på ressursserveren som sender data over websocket til alle tilkoblede brukere. Denne prosessen gjentas helt til tokenet utløper eller at tilkoblingen brytes av brukeren. Da vil ressursserveren automatisk fjerne brukeren som lytter.

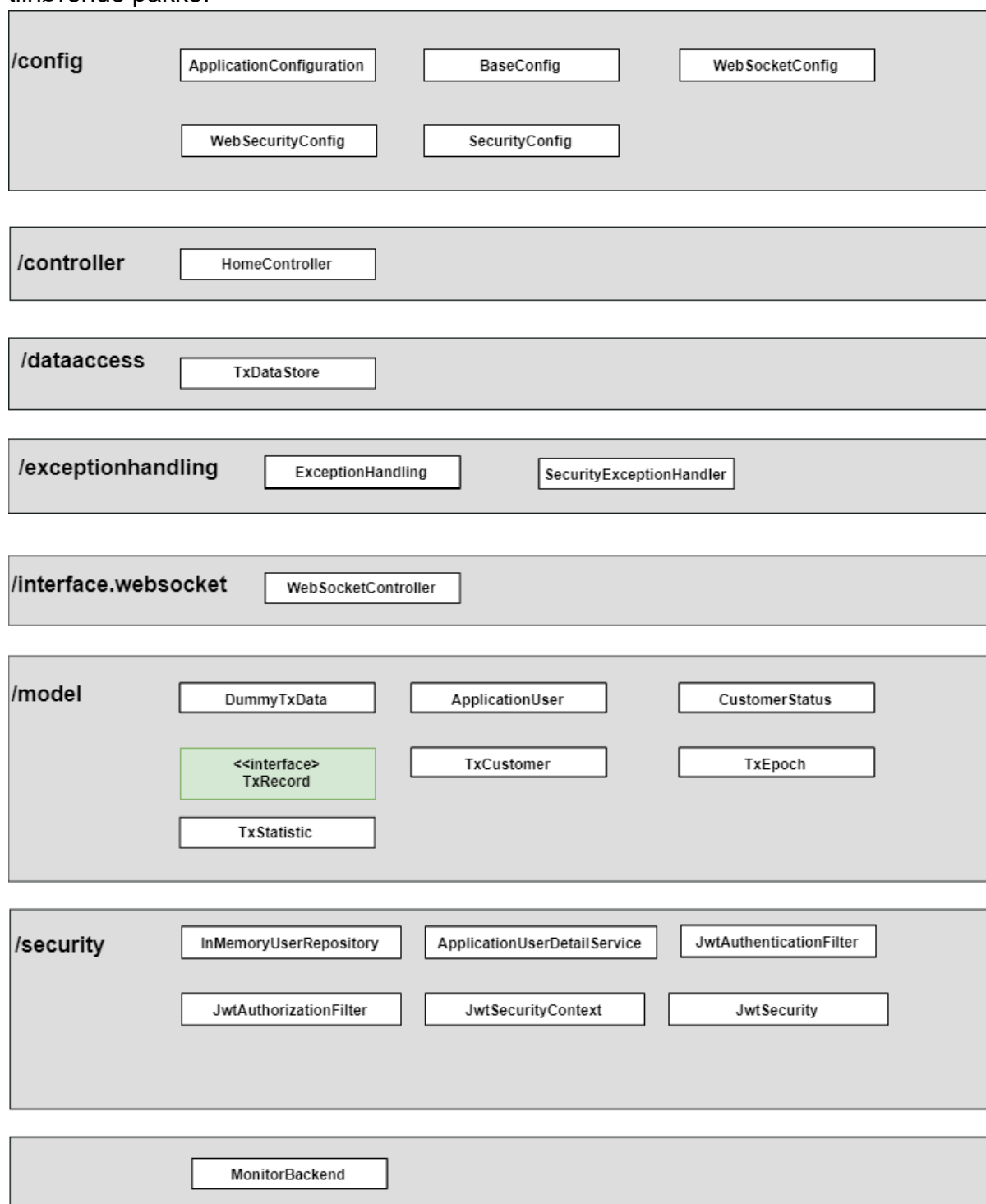
Systemarkitektur for backend

Backend er hovedsakelig delt inn seks deler.

Configuration	Inneholder konfigurasjonsdata for oppstart, utsending av data og sikkerhet.
WebSocket interface	Nettverksgrensesnitt for strømming av transaksjonsdata til klienten. Så lenge tokenet er gyldig vil grensesnittet sørge for at dataen blir korrekt overført.
Datastore and generator	Sorterer transaksjonsdata og utfører statistiske beregninger. Genererer fiktiv transaksjonsdata brukt under utviklingen.
Authentication	Kontrollerer identiteten til en bruker. Hvis bruker er gyldig skal det bli generert et JSON Web Token.
Authorization	Kontrollerer brukerens ressurstillgang basert på informasjonen i tokenet.
Error handling and logging	Sørger for at feil blir håndtert forsvarlig og at brukeren får tilbakemelding på hva som er feil. Alt av forespørsler, påloggingsforsøk og avvik blir logget.

Datamodell

Klassediagrammet er et overordnet diagram som viser alle klassene til backend og tilhørende pakke.



FIGUR 3: VISER KLASSENE OG TILHØRENDE PAKKE TIL BACKEND

Konfigurasjonen er delt inn i fem klasser.

Klassenavn	Konfigurerer
ApplicationConfiguration	Varighet av JSON Web Token, fornyelsesvinduet av tokenet, og oppsett av sikkerhetsnøkkelen blir satt her.
BaseConfig	Automatisk oversetting fra Java-objekter til JSON for overføring mellom backend og frontend.
SecurityConfig	Sikkerhet på tvers av domener, samt oppsett av kryptering for passord.
WebSecurityConfig	Spring Web Security, der det blir satt hvilke URL-er av API-et som er tilgjengelig og hvilke rettigheter som trengs for å besøke disse.
WebSocketConfig	Strømming av data, URL-en til strømmingen og kobler WebSocketController til håndteringen av dette.

Applikasjonen er konfigurert slik at tokenet har en gyldighet på 15 minutter, med et vindu for fornyelse satt til de siste 60 sekundene av tokenets gyldighet.

```
@NotNull
@DurationUnit(ChronoUnit.MINUTES)
private Duration tokenDuration = Duration.ofMinutes(15);

/**
 * The threshold for renewing tokens.
 */
@NotNull
@DurationUnit(ChronoUnit.SECONDS)
private Duration thresholdRenewalTime = Duration.ofSeconds(60);
```

FIGUR 4: UTDRAK FRA APPLICATIONCONFIGURATION HVOR VARIGHETS- OG FORNYELSESVINDUET TIL TOKENET BLIR DEFINERT

WebSocket-grensesnittet

Grensesnittet overfører nødvendig transaksjonsstatistikk til en klient. Dataen blir sendt gjennom en websocket-økten som er åpen så lenge tokenet er gyldig. Dataen blir sendt som et JSON-objekt og inneholder en liste over alle kundene og deres aktivitet, samt diverse overordnet statistikk. Generering og formatering av dataen som sendes foretas av TxDataStore.

```
@Override
public void handleMessage(WebSocketSession session, TextMessage message)
    throws IOException {

    this.session = session;

    try {
        JSONObject json = new JSONObject(message.getPayload());
        claims = jwtService.getClaims(json.get("token").toString());

        if (jwtService.isTokenStillValid(claims)) {
            /*
             * SENDING WHOLE EPOCH LIST OF ALL CUSTOMERS
             */
            session.sendMessage(new TextMessage(dataStore.getAllEpochsAsJson()));
            dataStore.addObserver(this);

            log.info(
                "WebSocket connection granted [ " + claims.getSubject() + " ] with client-IP " + session
                    .getRemoteAddress().getHostName()
                    + ". Token expires: " + claims.getExpiration().toString());
        }
    }
}
```

FIGUR 5: UTDRAK FRA WEBSOCKETCONTROLLER. HER SER VI TOKENET BLIR KONTROLLERT FØR KLIENSENS ØKT REGISTRERES SOM EN OBSERVATØR I DATASTORE

Generering og bevaring av fiktive data

Alle data-klassene applikasjonen jobber med ligger definert i model-pakken. Her finner vi modeller for brukere, transaksjonsdata, kundedata, med mer.

Klassen ApplicationUser inneholder hvilke roller brukeren kan ha. Rollen bestemmer hvilke rettigheter og ressurser brukeren har tilgang til.

```
/**
 * Lists of authority groups. More authorities can be added here.
 */
public enum Role implements GrantedAuthority {
    ROLE_USER("ROLE_USER"),
    ROLE_ADMIN("ROLE_ADMIN");

    private final String role;

    Role(String role) {
        this.role = role;
    }

    public String getAuthority() {
        return role;
    }
}
```

FIGUR 6: UTDRAK FRA APPLICATIONUSER. VISER AT EN BRUKER HAR EN ROLLE

Klassen DummyTxData inneholder de fiktive dataene som er brukt til å representere transaksjonsdataene. Det blir blant annet generert en fiktiv liste over kundene til PayEx og tilfeldige verdier på når en transaksjon inntreffer og tiden den tok å fullføre.

```
@Component
public class DummyTxData implements TxRecord {

    private static final String[] dummyCustomer = {
        "Shell", "Circle K", "Narvesen", "Power", "Elkjøp", "Kiwi", "Meny", "Joker"
    };

    private static Random r = new Random();

    private final int id;
    private final String customer;
    private final long txTime;
    private final int txMsToComplete;

    /**
     * Generates mock-data simulating a transaction. Used under dev-testing.
     */
    public DummyTxData() {
        id = r.nextInt(dummyCustomer.length);
        customer = dummyCustomer[id];
        txTime = System.currentTimeMillis();
        txMsToComplete = r.nextInt(200);
    }
}
```

FIGUR 7: UTDRAG FRA DUMMYTXDATA. DEN VISER HVORDAN VI REPRESENTERER FIKTIVE TRANSAKSJONSDATA

Grensesnittet TxRecord er laget for å gi utvikleren et enkelt grensesnitt på hvordan dataen til en transaksjon skal se ut. Fordelen med dette er å ha en standard på hva slags egenskaper en transaksjon skal ha. Når det i fremtiden benyttes ekte data fremfor fiktiv data, vil resten av systemet kunne behandle dette uten å behøve store endringer. Dette grensesnittet blir implementert av klassen DummyTxData.

```
public interface TxRecord {  
  
    /**  
     * Returns an unique integer value representing the numeric identification to the customer.  
     * @return Integer value of the customer id.  
     */  
    public int getId();  
  
    /**  
     * Returns a string containing the name of the customer.  
     * @return string with the customer name.  
     */  
    public String getCustomerName();  
  
    /**  
     * Returns the timestamp from when the transaction was made.  
     * @return Long value of the timestamp in Unix Epoch format.  
     */  
    public long getTransactionTime();  
  
    /**  
     * Returns number of milliseconds it took to process the transaction through the entire  
     * transaction chain. Note: The max value of the integer type puts a limit to ~24 days.  
     * @return Integer value representing how many milliseconds the transaction process took.  
     */  
    public int getTransactionLag();  
}
```

FIGUR 8: UTRAG FRA TxRECORD SOM DANNER GRUNNLAGET FOR REPRESENTASJONEN AV EN TRANSAKSJON

For å generere ny statistikk basert på de siste transaksjonene til kunden, og ikke minst generere nye transaksjoner som systemet kan jobbe mot, har vi laget flere klasser som tar seg av forskjellige oppgaver rundt dette.

Klasse	Hovedoppgaver	← Oversatt til kode
TxDataStore	<p>Inneholder en liste med alle kunder (TxCustomer).</p> <p>Bruker schedulers⁶ for:</p> <ul style="list-style-type: none"> • Generering av nye mock-transaksjoner • Kall på en metode i TxCustomer hvert sekund for å generere ny statistikk basert på de siste transaksjonene til kunden. 	<pre>@Scheduled(fixedRate = CHECK_CUSTOMER_EPOCH) public void runRoutines() { datastore.forEach((id, customer) -> { customer.runProcesses(); }); this.setChanged(); this.notifyObservers(); }</pre>
TxCustomer	<p>Lagrer alle transaksjoner til kunden for de siste 60 minutter. Transaksjonene er delt opp i epoker, der hver epoke er satt til å være 1 minutt.</p>	<pre>/** * Internal-private-method which will update the number of transactions per second for the * customer. */ private void updateStatistics() { int index = txBacklog.get(0).epochStart > System.currentTimeMillis() ? 1 : 0; txBacklog.get(index).getTxStatistic().refreshStats(); double secSinceLastCall = (System.currentTimeMillis() - txPrSecondTicker) / 1000d; this.txToday += txPrSecondCounter; txPerSecond = txPrSecondCounter / secSinceLastCall; txPrSecondTicker = System.currentTimeMillis(); txPrSecondCounter = 0; }</pre>
TxEpoch	<p>Inneholder et tidsavgrenset utvalg transaksjonsdata til tilhørende kunde. Epokene er lagret i en liste i hver kunde. TxStatistic blir brukt til å utføre statistikkberegning på transaksjonsdataen.</p>	<pre>public EpochAddRecordResult addTxRecord(TxRecord txRecord) { long txTime = txRecord.getTransactionTime(); if (txTime < epochEnd && txTime >= epochStart) { txBacklog.add(txRecord); return EpochAddRecordResult.OK; } if (txTime < epochStart) { return EpochAddRecordResult.OLDER; } }</pre>

⁶ En metode som automatisk kjøres med en viss frekvens

Klasse	Hovedoppgaver	← Oversatt til kode
TxStatistic	<p>Kalkulerer grunnleggende statistikk på transaksjonsdata. Verdier som blir kalkulert på transaksjonstiden:</p> <ul style="list-style-type: none"> • Median • Maksimum • Minimum • Øvre kvartil • Nedre kvartil 	<pre>public void refreshStats() { if (txRecords.size() > 0) { txRecordsCopy = txRecords.toArray(new TxRecord[txRecords.size()]); Arrays.sort(txRecordsCopy, (x, y) -> { return (int) (x.getTransactionLag() - y.getTransactionLag()); }); updateMaxMinTxPerSecond(); updateMedianTxPerSecond(); updateUpperQuartileTxPerSecond(); updateLowerQuartileTxPerSecond(); } }</pre>

Fiktive data

```
{
  "numTxSec": 2,
  "totalTxToday": 21,
  "customers": [{
    "id": 0,
    "customer": "Shell",
    "customerStatus": "OK",
    "txToday": 4,
    "txData": [{
      "upperQuartile": 0,
      "median": 0,
      "x": 1,
      "maximum": 0,
      "epochStart": 1526472038145,
      "minimum": 0,
      "txVolume": 0,
      "lowerQuartile": 0,
      "epochEnd": 1526472098145
    }]
  }]
}
```

FIGUR 9: UTSNITT AV FIKTIV DATA SOM SENDES UT AV RESSURSSERVEREN

Felt	Forklaring
numTxSec	Antall transaksjoner i sekundet fra alle kundegruppene. Verdien oppdateres og sendes ut en gang i sekundet.
totalTxToday	Totalt antall transaksjoner i dag fra alle kundegruppene.
Id	ID-nummeret på kundegruppen.
Customer	Navnet på kundegruppen.
customerStatus	Statusen på transaksjonene til kundegruppen som kommer inn. De kan variere fra: <ul style="list-style-type: none"> • OK • SLOW: Sakte flyt, men transaksjonene kommer inn. • DOWN: Det kommer ingen transaksjoner inn.
txData	Inneholder en liste med statistikk om transaksjonene som har blitt utført i løpet av gjeldende minutt.
upperQuartile	Øvre kvartil for epoken.
Median	Median for epoken.
X	Forteller om hvilket minutt det er i timen.
Maximum	Lengste prosesseringstid for en transaksjon i løpet av epoken.
epochStart	Når epoken startet. Tiden er i Unix time ⁷ .
Minimum	Korteste prosesseringstid for en transaksjon i løpet av minuttet.
txVolume	Volumet av transaksjoner – antallet transaksjoner for denne kundegruppen i denne epoken.
lowerQuartile	Nedre kvartil for epoken.
epochEnd	Tidspunktet når epoken ble avsluttet. Tiden er i Unix time.

⁷ Vanligvis sekunder siden Unix' epoch (1. januar 1970), her er presisjonen millisekunder.

Feilhåndtering og logging

Problem-biblioteket blir brukt for å automatisk ta seg av unntak når de blir kastet. Behandleren vil i tillegg sende en feilmelding i JSON-format tilbake til brukeren. Vi har to klasser som manuelt behandler unntak. Begge klassene bruker biblioteket, men de arver to forskjellige funksjonaliteter av biblioteket. Disse klassene er implementert for å ha et felles sted for å håndtere unntak. For ytterligere informasjon om hvilke feilmeldinger som blir automatisk sendt til frontend se [Vedlegg H: Utsendte feilmeldinger](#).

```
@ControllerAdvice
public class SecurityExceptionHandler implements SecurityAdviceTrait {

    // Example of how to manually handle an exception.

    @ExceptionHandler(BadCredentialsException.class)
    public Problem errorHandler(HttpServletRequest response, BadCredentialsException e) {
        return Problem.valueOf(Status.FORBIDDEN);
    }
}
```

FIGUR 10: VISER IMPLEMENTERINGEN AV SECURITY ADVICE TRAIT SOM TAR SEG AV AUTENTISERING OG AKSESS FEIL

Alt av forespørsler, påloggingsforsøk, mistenkelige handlinger og avvik i dataen blir logget i backend. Loggingen har forskjellige nivåer:

- Error (feilmelding)
 - Logger feil.

```
2018-05-14 11:58:19,154 ERROR [pool-4-thread-1] com.payex
.monitor.dataaccess.TxDataStore: Power not receiving any transactions
```

FIGUR 11: FEILLOGGING. LOGGER AT DET IKKE KOMMER NOE TRANSAKSJONER FRA KUNDEGRUPPEN POWER

- Warning (Advarsel)
 - Gir advarsel når det ikke er direkte feilmelding, men burde være observant på.

```
2018-05-14 11:58:22,164 WARN [pool-4-thread-1] com.payex
.monitor.dataaccess.TxDataStore: Kiwi transactions are slow
```

FIGUR 12: ADVARSELLOGGING. LOGGER AT TRANSAKSJONENE PROSESSERES SAKTE FOR KUNDEGRUPPEN KIWI

- Information (informasjon)
 - Vanlig informasjon brukeren får.

```
2018-05-14 11:58:20,436 INFO [ http-nio-8098-exec-6] monitor
.security.JwtAuthenticationFilter: User [ user ] authenticated successfully with
client-IP 0:0:0:0:0:0:1
```

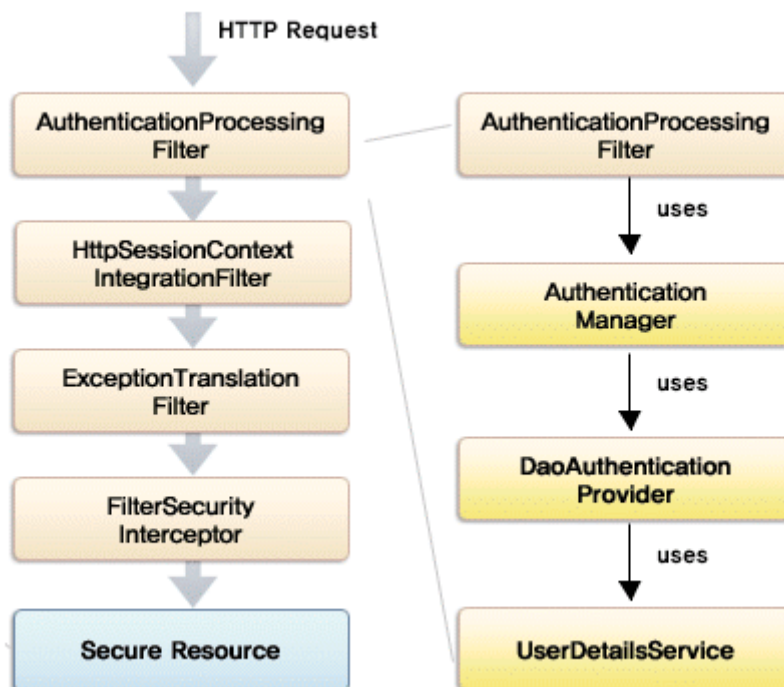
FIGUR 13: INFORMASJONSLOGGING. LOGGER AT EN BRUKER HAR BLITT AUTENTISERT MED BRUKERNAVN OG GITT IP-ADRESSE

Sikkerhet

Applikasjonen er sikret ved bruk av Spring Web Security som tar seg av autentisering og autorisering av brukere. Autentiserings- og autorisasjonsprosessen går igjennom en filterkjede kjent som Spring Security Filter Chain. Filtrene håndterer og sikrer at applikasjonens sikkerhetskrav er møtt.

Autentiseringsfilteret går igjennom en prosess for å identifisere brukeren og genererer et token hvis autentiseringen er vellykket.

Spring Security Filter Chain



FIGUR 14: FIGUREN VISER DEN GENERELLE FLYTEN GJENNOM FILTRENE NÅR DET BES OM TILGANG TIL EN RESSURS SIKRET AV SPRING SECURITY

Applikasjonen overskriver noen av standardfiltrene:

UsernamePasswordAuthenticationFilter, BasicAuthenticationFilter.

Disse filtrene ble overskrevet fordi innloggingsdataen blir sendt i JSON-formatet. Dette hadde Spring Security Filter Chain ikke direkte støtte for. Derfor måtte vi overskrive filteret med vår egen løsning for å kunne hente ut brukernavn og passord fra JSON-formatet. På den måten fikk vi muligheten til å sende ut token ved riktig innloggingsinformasjon, samt sjekke om tokenet er gyldig eller ikke ved forespørsel om tilgang til data.

I tillegg har vi implementert klassene UserDetailsService, UserDetails, og SecurityContext fra Spring. Vi overskrev UserDetailsServices for å gjøre det mulig å hente ut brukere fra hvilken som helst kilde. I vårt tilfelle er brukeren lagret i minnet og blir hentet ut derfra.

Autentiseringsfilteret

Autentiseringsfilteret er inngangspunktet for påloggingsforespørsler. Forespørselen blir sjekket for riktig mediatype⁸, oppbygging og format. Hvis alt er riktig vil filteret:

1. Henter ut brukernavn og passord fra innloggingsforespørselen.
2. AuthenticationManager bruker UserDetailsService for å hente ut legitimasjonen til brukeren fra innloggingsforespørselen.
3. Påloggingsinformasjonen blir validert.
4. Token blir generert og sendt ut til brukeren.

⁸ MIME-type, f.eks. image/png, text/html eller application/json

```
@Override
public Authentication attemptAuthentication(HttpServletRequest request,
    HttpServletResponse response)
    throws AuthenticationException {
    /*
     * We only attempt to authenticate if the HTTP request is a POST with JSON content.
     * {username:"",password:""} = 25 bytes (normally content-length should be greater)
     */
    logger.debug(request.getContentType());
    if (!request.getMethod().equals("POST")
        || !validContentTypes.contains(request.getContentType())) {
        try {
            response.setContentType("application/json");
            response.getWriter().print(response(400, "Invalid request"));
        } catch (IOException e1) {
            e1.printStackTrace();
        } catch (JSONException e1) {
            e1.printStackTrace();
        }
    }
    log.info("Invalid login attempt from client with IP " + request.getRemoteAddr()
        + ", missing POST or content type not JSON");
    } else {
        String username = "";
        String password = "";
        try {
            // We extract the body-content sent by the client.
            String formData = request.getReader().lines()
                .collect(Collectors.joining(System.lineSeparator()));
            JSONObject json = new JSONObject(formData);
            username = json.getString("username");
            password = json.getString("password");

            return authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(
                username,
                password,
                new ArrayList<>()
            ));
        }
    }
}
```

FIGUR 15: UTSNITT FRA JWTAUTHENTICATIONFILTER. VISER HVORDAN FILTERET KONTROLLERER AT BRUKERNAVN OG PASSORD ER SENDT SOM JSON

```
@Override
protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response,
    FilterChain chain, Authentication auth)
    throws IOException, ServletException {

    UserDetails user = userService.loadUserByUsername(auth.getName());
    String token = jwtService.generateToken(user);

    JwtSecurityContext jwtSecurityContext = new JwtSecurityContext(jwtService, userService);
    jwtSecurityContext.setAuthentication(auth);
    jwtSecurityContext.setToken(token);
    SecurityContextHolder.setContext(jwtSecurityContext);

    Log.info("User [ {} ] authenticated successfully with client-IP {}", auth.getName(),
        request.getRemoteAddr());

    chain.doFilter(request, response);
    getAllowSessionCreation();
}
```

FIGUR 16: UTSNITT FRA JWTAUTHENTICATIONFILTER. GENERERING OG UTSENDING AV NY TOKEN

Autorisasjonsfilter

Autorisasjonsfilteret blir tatt i bruk når det kommer inn forespørsler om tilgang til ressurser. Disse forespørslene inneholder tokenet og dette blir sjekket før autorisasjon prosessen starter. Prosessen er som følger:

1. Det blir sjekket om det bes om tilgang for utstrømming av data.
2. Det blir så hentet ut brukernavn og tokenets gyldighet ved bruk av servicen som håndterer tokenet.
3. Det blir så sjekket om tokenet er gyldig og brukeren har rettigheter til å få tilgang.
4. Det blir så gitt tilgang.

Autorisasjonsfilteret tar i bruk Java Spring sin AuthenticationManager for å hente ut rettigheter. Dette filteret blir brukt sammen med vår egen SecurityContext, som inneholder autorisasjonslegitimasjonen. SecurityContext ble implementert på nytt for å gjøre det lettere å hente ut brukerrettigheter i applikasjonen og fornye tokenet.

```
String formData = request.getReader().lines()
    .collect(Collectors.joining(System.lineSeparator()));

JSONObject json = new JSONObject(formData);
String token = json.getString("token");
Claims claims = jwtService.getClaims(token);

UsernamePasswordAuthenticationToken authentication =
    new UsernamePasswordAuthenticationToken(
        claims.getSubject(), null, new ArrayList<>());
authentication.setDetails(claims);

JwtSecurityContext jwtSecurityContext = new JwtSecurityContext(jwtService, userService);
jwtSecurityContext.setAuthentication(authentication);
jwtSecurityContext.setHttpServletRequest(request, response);
jwtSecurityContext.setToken(token);
```

FIGUR 17: KODEUTSNITT HENTET UT FRA JWTAUTHORIZATIONFILTER. VISER UTHENTING AV TOKEN OG LAGING AV JWTSECURITYCONTEXT

Henting og lagring av brukere

Brukeren blir hentet ut ved bruk av `ApplicationUserDetailsService`, som implementerer `SpringUserDetailsService`. Dette brukes for å hente ut brukeren med kryptert passord og rettigheter fra en tredjepart, som database, eller Active Directory. I vårt tilfelle hentet vi ut informasjonen fra brukere direkte lagret i minnet.

```
@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    return repository
        .findUser(username)
        .map(user -> new User(user.getUsername(), user.getPassword(), user.getAuthorities()))
        .orElseThrow(() -> new UsernameNotFoundException(username));
}
```

FIGUR 18: HENTET FRA APPLICATIONUSERDETAILSERVICE. VISER UTHENTING AV BRUKER LAGRET I APPLIKASJONEN

Brukeren blir først opprettet ved bruk av Spring sin UserDetails, som er et grensesnitt som inneholder de viktigste implementasjonen for en bruker. Brukeren blir manuelt opprettet ved applikasjonsstart og så lagret i minnet.

```
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    return authorities;
}

@Override
public String getPassword() {
    return password;
}

@Override
public String getUsername() {
    return username;
}

@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return enabled;
}

public Date getLastPasswordResetDate() {
    return lastPasswordResetDate;
}
```

FIGUR 19: UTSNITT FRA APPLICATIONUSER. VISER HVA EN BRUKER INNEHOLDER

Token

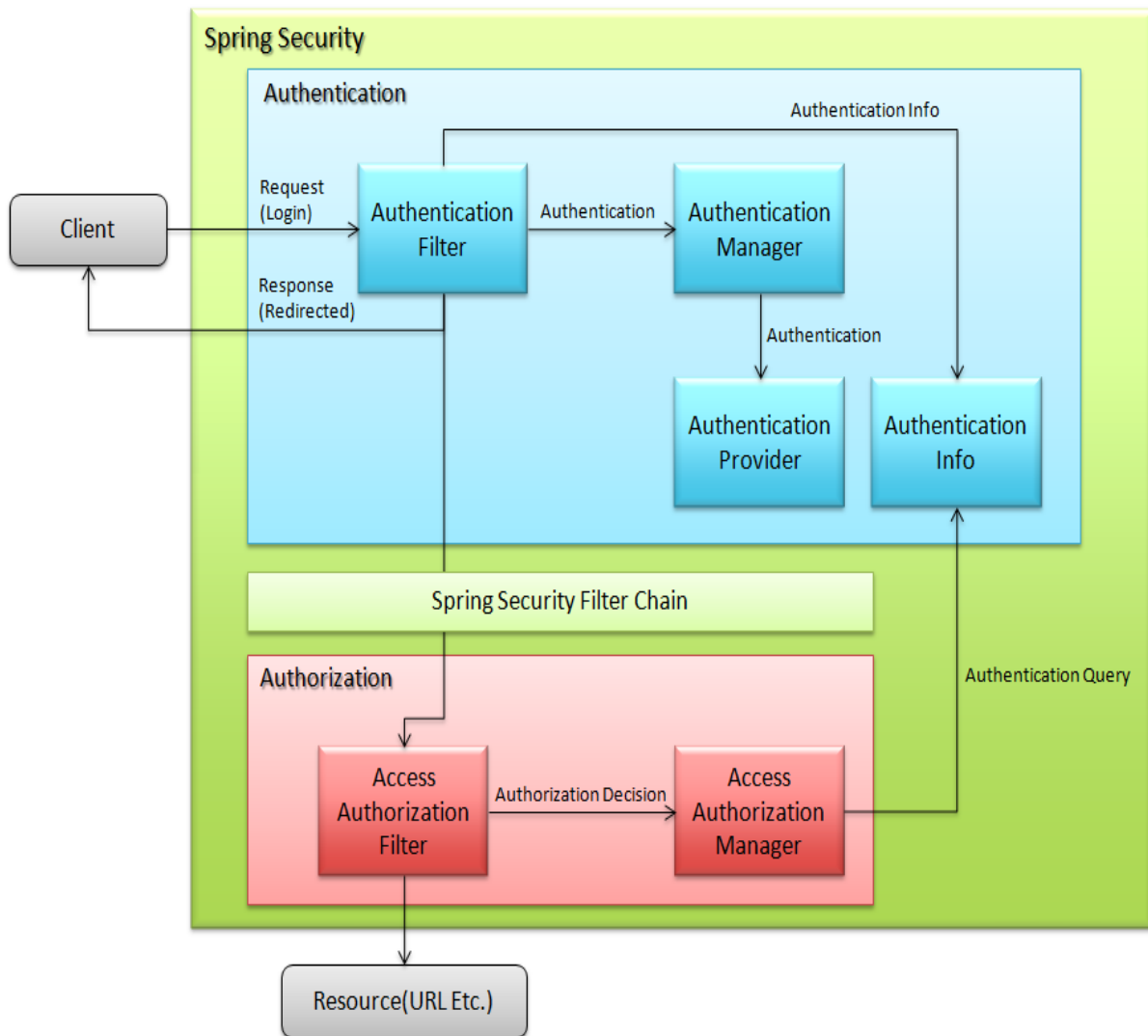
En token blir laget ved bruk av [JSON Web Token](#). Tokenet blir generert ved bruk av klassen `JwtService`. Klassen genererer en token ved bruk av brukers brukernavn, roller og en hemmelig nøkkel. Tokenet er satt til å være gyldig i 15 minutter.

```
public String generateToken(UserDetails user) {
    /*
     * Custom values we want the token to hold are put inside the Map claim variable.
     */
    Map<String, Object> claim = new HashMap<>();
    claim.put("USER_ROLE", user.getAuthorities());

    final Instant issuingTime = clock.instant();
    final Instant expirationTime = issuingTime.plus(tokenDuration);

    return Jwts.builder()
        .setClaims(claim)
        .setSubject(user.getUsername())
        .setIssuedAt(Date.from(issuingTime))
        .setExpiration(Date.from(expirationTime))
        .signWith(SignatureAlgorithm.HS512, secret.getBytes())
        .compact();
}
```

FIGUR 20: UTSNITT HENTET FRA JWTSERVICE. VISER GENERERING AV TOKEN VED BRUK AV JSON WEB TOKEN



FIGUR 21: AUTENTISERING OG AUTORISASJONSFLYTEN. SER HVORDAN SPRING HÅNDRER INNLOGGINGSFORESPØRSELER⁹.

⁹ Guideline, T. G. F. D. "Spring Security Overview." Retrieved 21 May, 2018, from [HTTPS://TERASOLUNAORG.GITHUB.IO/GUIDELINE/1.0.X/EN/SECURITY/SPRINGSECURITY.HTML](https://terasolunaorg.github.io/guideline/1.0.x/en/security/springsecurity.html)

Frontend

Når man bygger store og skalerbare webapplikasjoner er det fordelaktig å kunne dele opp funksjonaliteten i komponenter. Ikke bare for å forenkle gjenbruk av kode men også for å gi bedre oversikt. Det finnes flere rammeverk som tilbyr slik funksjonalitet, men i Vue blir også komponent-avhengigheter sporet slik at systemet vet nøyaktig hva som faktisk må gjengis når en tilstand endres. Dette forenkler utviklers arbeid med tanke på optimalisering og er en av hovedårsakene til at Vue.js ble valgt fremfor for eksempel React¹⁰. I tillegg kan Vue også benyttes side om side med andre biblioteker og egner seg spesielt godt til utvikling av SPA-er (Single Page Application).

Hovedoppgaven til frontend er å visualisere statistikk som mottas fra ressursserveren. For å utføre denne oppgaven effektivt, benytter frontend seg av JavaScript biblioteket D3 (Data Driven Documents). Dette er et visualiseringsbibliotek som bruker SVG, som egner seg godt til å produsere sanntidsgrafer og gir utvikler stor frihet over det visuelle resultatet. Ulempen er at utvikler må designe det visuelle helt fra bunn. D3 har også støtte for JSON, noe som bidrar til å forenkle og effektivisere prosessen med å tegne grafer.

Før vi landet på D3 testet vi 11 forskjellige JavaScript biblioteker for tegning av grafer. Under testingen merket vi oss at det var stor ytelsesforskjell mellom de forskjellige løsningene, se [Vedlegg C: Kravspesifikasjon](#). Et gjennomgående problem blant alle bibliotekene var at prosessen hadde en tendens til å henge seg opp dersom fanen i nettleseren de kjørte på stod i bakgrunnen. I motsetning til D3, ga den oss friheten til å styre alle mekanismene. Ulempen var at vi måtte skrive rutinene for håndtering og opptegning av grafene helt fra bunnen av.

Statistikken som sendes over fra ressurs serveren inneholder data som forteller bruker om antall transaksjoner gitt over et intervall på 1 minutt, og responstiden mellom PayEx sine systemer og innløser. Antall transaksjoner blir presentert som et søylediagram, mens responstiden mellom PayEx og innløser tegnes opp som boksdigram. Dette diagrammet har aggregert data over samme tidsintervall på responstiden, og tilbyr visualisering av øvre kvartil, nedre kvartil og median.

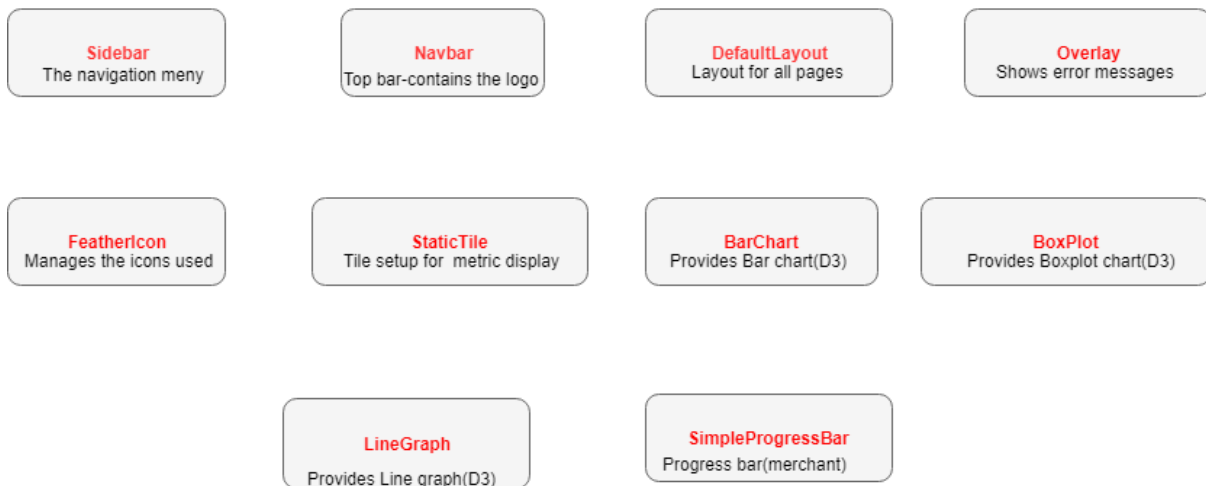
Fordelene med løsningen over er at man med et kort blick kan se om det er avvik i transaksjons-bildet. Om linjen mellom PayEx eller innløser er nede vil dette gi utslag i form av at boksdigrammet vokser entydig. Hvis bruker har problemer med sine transaksjoner vil dette også vises ved at søylediagrammet er fraværende.

¹⁰ Vue.js. "Comparison with Other Frameworks." Retrieved 22 May, 2018, from <https://vuejs.org/v2/guide/comparison.html>

Systemarkitektur for frontend

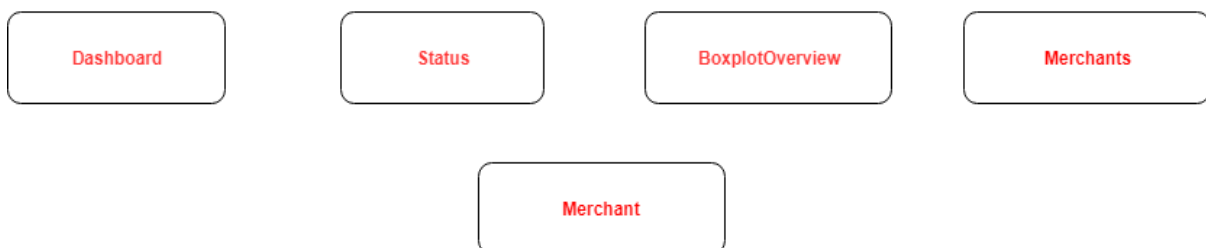
Frontend er delt opp i komponenter som enkelt kan legges inn og brukes hvor som helst i visningen. Dette gjør at vi kan gjenbruke funksjonalitet på nytt andre steder i applikasjonen.

Components



FIGUR 22: OVERSIKT OVER ALLE KOMPONENTENE SOM BLE UTVIKLET TIL FRONTEND. DISSE KOMPONENTENE ER BRUKT TIL Å LAGE FORSKJELLIGE SIDER SOM BRUKEREN KAN BESØKE.

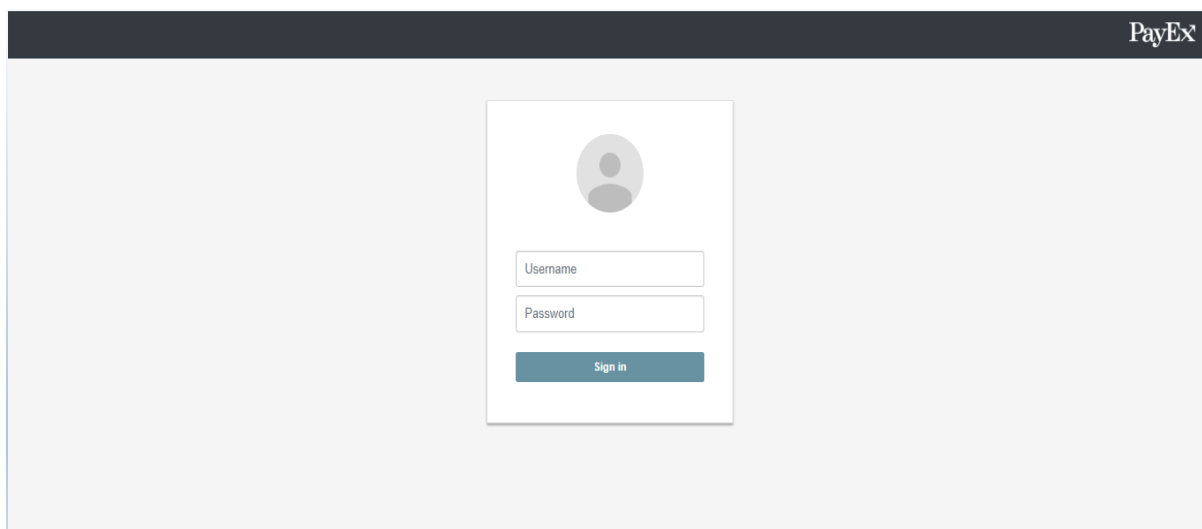
Views



FIGUR 23: OVERSIKT OVER ALLE BESØKBARE SIDER

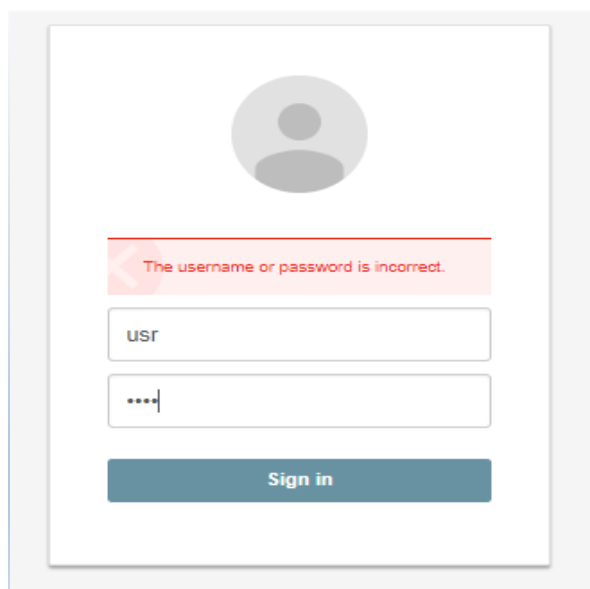
- Dashbordet viser overordnet status over alle utførte transaksjoner.
- Status er ansvarlig for å gi brukeren detaljer om nye transaksjoner og hvor lang tid det har gått siden sist transaksjon kom inn
- BoxPlotOverview viser boksdigrammer for alle kundegruppene.
- Merchants viser valg av kundene for å se detaljert informasjon.
- Merchant er et dynamisk view som viser informasjon om transaksjoner for hver enkelt kunde.

Innloggingssiden



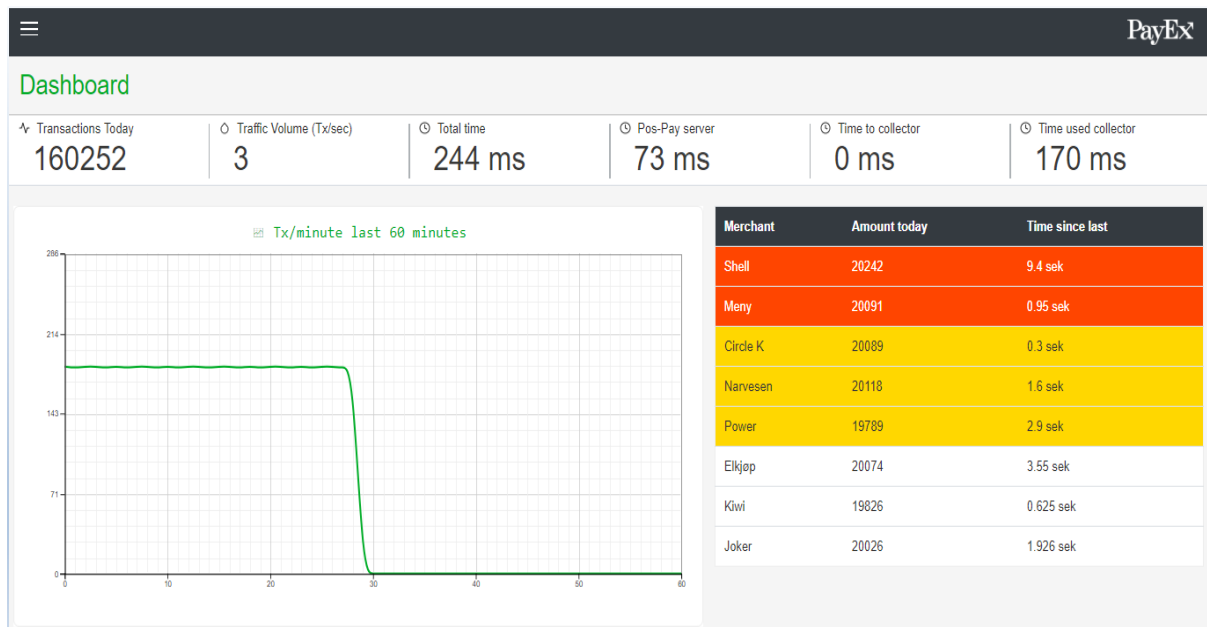
FIGUR 24: SKJERMDUMP HENTET FRA INNLOGGINGSSIDEN.

Når brukeren taster feil brukernavn og passord vil det vises på følgende måte:



FIGUR 25: UTSNITT FRA INNLOGGINGSSIDEN SOM ILLUSTRERER TILBAKEMELDINGEN VED FEIL BRUKERNAVN ELLER PASSORD.

Dashboard



FIGUR 26: SKJERMDUMP FRA DASHBOARD SIDEN.

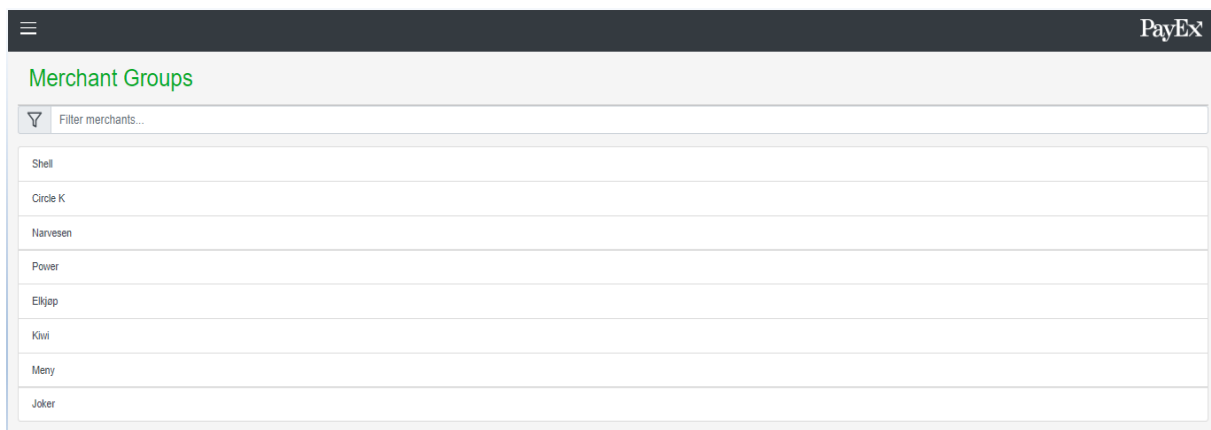
- Informasjonslinjen under Dashboard viser detaljerte opplysninger om antall transaksjoner i dag, gjennomsnittlig transaksjoner i sekundet, behandlingstid av serveren, tid brukt for overføring fra og til innløser.
- Grafen forteller om antall transaksjoner i minuttet den siste timen.
- Tabellen viser antall transaksjoner for hver kundegruppe.
 - Rød fargen viser stopp i mottakelse av transaksjoner.
 - Gul fargen viser at transaksjonene kommer inn tregt.
 - Hvit fargen illustrerer normal prosessering av transaksjonene.



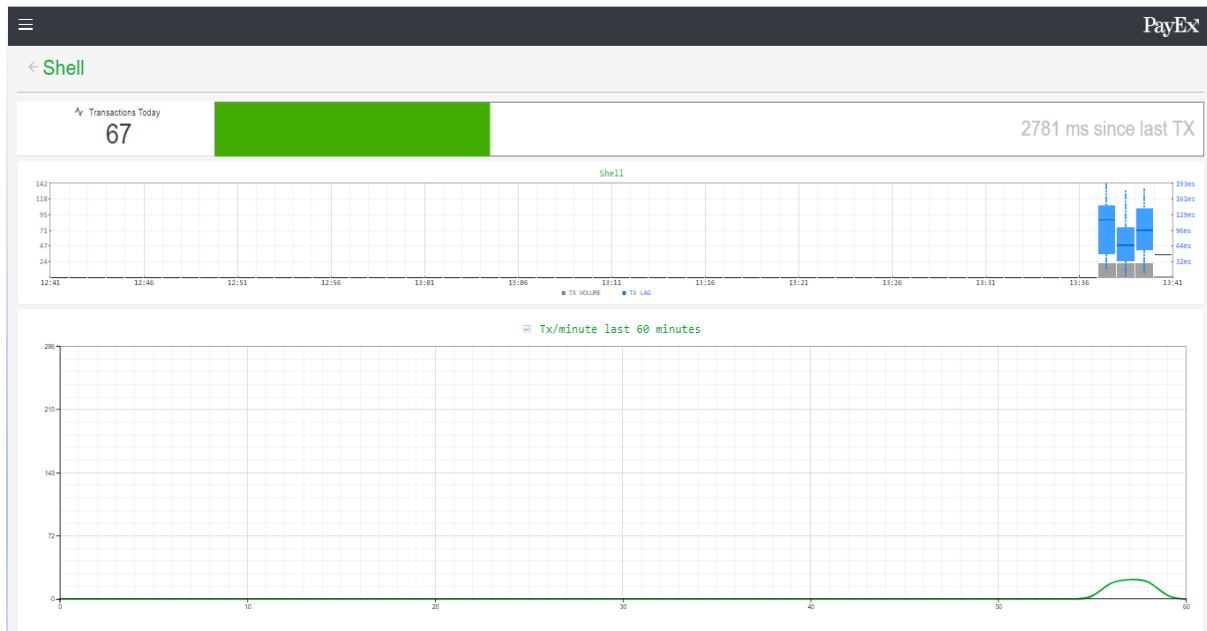
FIGUR 27: SKJERMDUMP FRA STATUSSIDEN.

- Viser hvor mange sekunder siden sist transaksjon ble utført hos kunden.
- Etter hvert som tiden går, glir fargen fra grønt til gult og så rødt.

Merchants



FIGUR 28: VISNING AV MERCHANT VIEW. VISER EN LISTE MED ALLE KUNDEGRUPPE. STØTTE FOR FILTRERING OG INTERAKTIVITET FOR MER INFORMASJON OM KUNDEGRUPPEN.

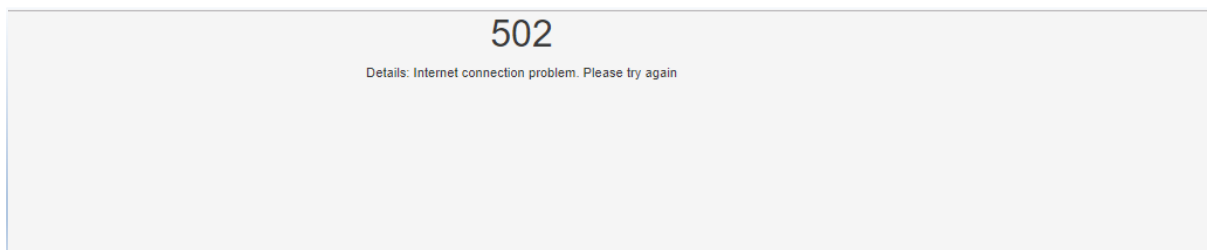


FIGUR 29: SKJERMDUMP FRA MERCHANT VIEW PÅ KUNDEGRUPPEN SHELL.

- Viser totalt antall utførte transaksjoner i dag.
- Viser hvor lang tid det har gått siden sist transaksjon.
- Detaljert informasjon over transaksjonsdata vist i form av boxplot.
- Linjediagram som viser transaksjoner de siste 60 minuttene.

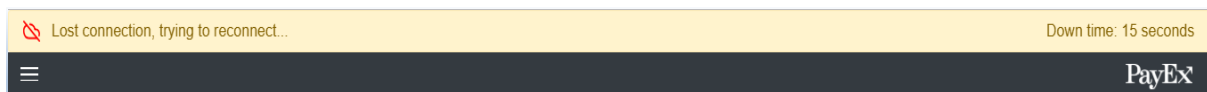
Visning av feilmeldinger

Hvis noe er galt og det fører til stopp i web applikasjonen, vil brukeren bli dirigert til en feilmeldingsside som forteller overordnet hva feilen er.

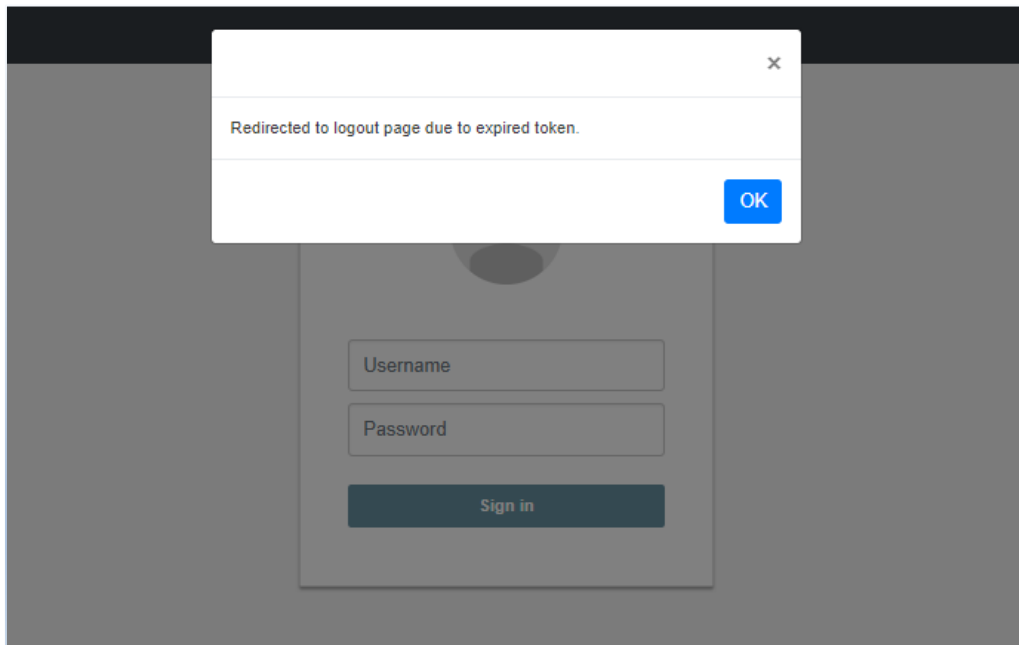


FIGUR 30: UTSNITT FRA FEILMELDINGEN. VISES NÅR DET IKKE OPPNÅS KONTAKT MED BACKEND.

Hvis bruker er innlogget og det er brudd i tilkoblingen til ressurs serveren som sender ut data, vil en feilmelding vises for brukeren. Feilmeldingen vil vise hvor lenge feilen har vart og prøve å koble seg til på nytt hvert 10 sekund så lenge tokenet er gyldig. Hvis tokenet utløper vil brukeren bli logget ut, og det vil vises et dialogvindu som forteller dette.



FIGUR 31: VISER FEILMELDINGEN VED TILKOBLINGSSVIKT



FIGUR 32: SKJERMDUMP AV FEILMELDINGEN SOM VISES VED UTGÅTT TOKEN

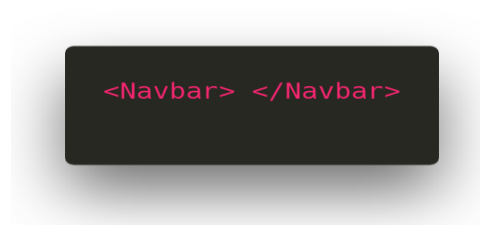
Rammeverk

Vue.js

[Vue](#) tilbyr en rekke funksjonaliteter som gjør det lettere å utføre oppgaver. Du kan oppdatere siden ved innlastning ved bruk av `mounted`, automatisk transformering av data ved bruk av `computed`, og bli varslet om endringer ved bruk av `watch`.

Komponenter

En av hovedfunksjonalitetene til Vue er komponenter. Du kan enkelt sette inn komponenter i form av den funksjonaliteten og visualiseringen de tilbyr:



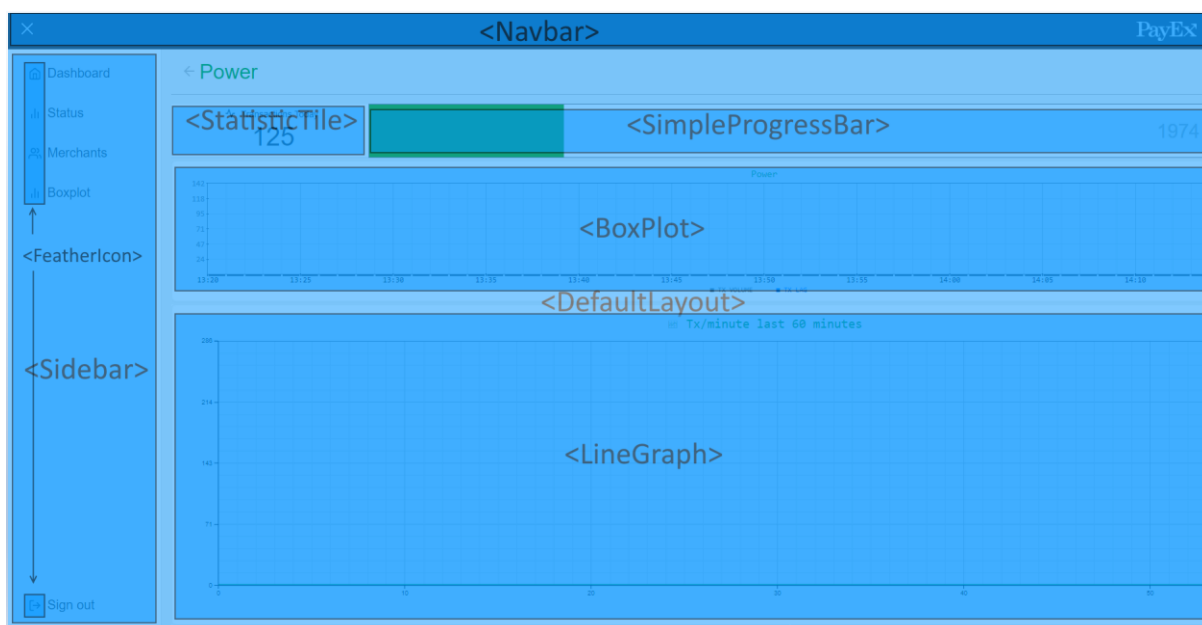
FIGUR 33: INNLEGGING AV EN KOMPONENT DEFINERES SLIK

Komponentene blir definert på følgende måte:

```
export default {
  name: 'Navbar',

  components: {
    Hamburger,
  },
}
```

FIGUR 34: DEFINERER EN KOMPONENT SOM HETER «NAVBAR» OG SOM SELV BRUKER EN «HAMBURGER»-KOMPONENTEN



FIGUR 35: SKJERMDUMP AV MERCHANT VIEW BRUTT NED I VÅRE DELKOMPONENTER

Mounted

Mounted er en metode som kjøres når gjeldende side laster inn. Vi har hovedsakelig brukt denne metoden for å utføre nødvendig funksjonalitet ved innlastning av view eller komponent. I dette eksemplet setter vi i gang en tidsmåler når feilmeldingen vises til brukeren. Tidsmålingen viser hvor lenge feilen har vart:

```
mounted() {  
  this.turnonInterval(this.$store.state.transactions.downforminutes);  
},
```

FIGUR 36: SETTER I GANG TIDSMÅLER DER VI HENTER UT ANTALL MINUTTER FRA FELLESLAGRET VUEX.

Computed

Computed er en metode som oppdaterer og sender ut signal når en endring oppstår for tilsvarende variabel. For eksempel når det kommer inn ny transaksjonsdata, eller annen data som er lagret i hovedlageret. Med computed kan vi enkelt forandre på tekst, og få den nyeste verdien når noe har endret seg.

```
TokenExpireDate() {  
  return this.$store.state.tokenExpireTime;  
},
```

FIGUR 37: RETURNERER TOKENETS UTLØPSTID HVER GANG DEN FORANDRER SEG

Watch

Watch blir kalt på når det skjer en endring i en variabel. Lytteren får den nye og den gamle verdien, og kan da utføre operasjoner basert på hvordan verdien har endret seg, eller bare basert på det faktum at verdien *har* endret seg (for eksempel ved integrering av tredjeparts biblioteker).

```
customers: function(cust) {
  this.SlowArray = [];
  this.DownArray = [];
  cust.map((customer)=>{
    if (customer.customerStatus == 'SLOW') {
      this.SlowArray.push(customer);
    } else if ( customer.customerStatus == 'DOWN') {
      this.DownArray.push(customer);
    }
  });
},
```

FIGUR 38: UTDRAK AV KODE SOM OPPDATERER STATUSEN TIL EN KUNDEGRUPPE PÅ DASHBOARD

Vue Router

Vue Router håndterer navigasjonen fra et view til et annet. Vue Router inneholder også sikkerhetsmekanismer som bestemmer om en bruker har lov til å navigere til valgt sted eller ikke. I *index.js* i Router-mappen definerer vi hvor det er mulig å navigere til.

```
this.$router.push('/');
```

FIGUR 39: NAVIGERER TIL INNLOGGINGSSIDEN.

```
const router = new Router({
  mode: 'history',
  linkActiveClass: 'active',
  linkExactActiveClass: 'active-exact',
  routes: [
    {
      path: '/',
      name: 'Login',
      component: Login,
      meta: {
        anonymous: true,
      },
    },
  ],
});
```

FIGUR 40: UTSNITT FRA ROUTER-DEFINISJONEN

- Mode: 'History'
 - Gir vanlige URL-er av typen `example.com/nested/resource` i stedet for `example.com/#nested/resource`. Krever støtte i nettleser, men er godt støttet av nettleserne vi trenger å støtte.
- LinkActiveClass og LinkExactActiveClass setter CSS-klasser som automatisk blir tilført lenker slik at vi kan utheve den aktive siden i navigasjonen.
- Vi definerer så routes som en liste av route-objekter.

Man har også muligheten til å legge navigasjonslenker direkte i HTML-koden. Disse lenkene vil ikke laste en helt ny side, som lenker på statiske HTML-sider gjør, men bytte ut komponenten som vises.

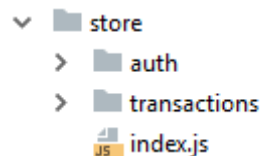
```
<router-link to="/dashboard" id="dashboard" class="nav-link">
```

FIGUR 41: VI KAN DA BRUKER ROUTER-LINK NOTASJONEN FOR OG SÅ GI KOMPONENT STIEN.

Vuex

Vuex er en tilstandshåndteringsutvidelse for Vue-applikasjoner. Vuex tilbyr et sentralisert lager som holder på data som kan bli brukt av alle komponenter. Det følges visse standarder for å endre på innholdet. Endringer i tilstanden blir automatisk videreført til komponenter som benytter tilstandsvariablene. Dette bidrar til at utvikler slipper å gjøre tilstandsendringer på komponentene manuelt.

Applikasjonen bruker Vuex til å håndtere autentisering, autorisering og mottakelse av data fra backend. Dette gjør at all den informasjonen som er nødvendig ligger samlet på ett sted.



FIGUR 42: STRUKTUR OVER HVORDAN HOVEDLAGERET SER UT. «AUTH» TAR SEG AV INNLOGGINGEN OG LAGRING AV TOKEN. «TRANSACTIONS» TAR SEG AV MOTTAKELSE AV DATA I TILLEGG TIL Å SE TIL AT RIKTIGE VARIABLER BLIR OPPDATERT. INDEX.JS HAR DEN GRUNNLEGGENDE INFORMASJONEN OM HOVEDLAGERET.

```
state: {  
  authenticated: false,  
  username: null,  
  authToken: null,  
  renewalToken: null,  
  error: null,  
  isExpired: false,  
},
```

FIGUR 43: DATA LAGRES I LAGRET I STATE – TILSTAND. UTDRAGET ER TATT FRA INDEX.JS DER VI HOLDER PÅ VERDIER OM AUTENTISERING ER UTFØRT, BRUKERNAVNET, TOKEN, NYESTE FEILMELDING, OG OM TOKEN ER GYLDIG ELLER IKKE.

Tilstandene blir forandret ved hjelp av mutasjoner, som for eksempel:

```
SET_TOKEN(state, {token}) {  
  state.authToken = token;  
},
```

FIGUR 44: HER LAGRES TOKENET I HOVEDLAGERET.

Mutasjoner sendes til lageret ved å bruke Vuex' *commit*. Mutasjonstypen gis som en streng.

```
commit('SET_TOKEN', {token: data.token});
```

FIGUR 45: SETTER TOKENET VED BRUK AV *COMMIT* OG MUTASJONEN 'SET_TOKEN'

Behandling av mottatt transaksjonsdata

Etter at brukeren har autentisert seg, vil transaksjonene som mottas fra backend bli lagret i hovedlageret til frontend. Dataene blir kontinuerlig oppdatert, det vil si en gang i sekundet. Funksjonene *computed* og *watch* kan videre benyttes til å oppdatere visualiseringen av dataene fortløpende.

```
customers() {  
  return this.$store.state.transactions.customers;  
},
```

FIGUR 46: UTSNITT FRA KLASSEN *BoxPlotOverview*. VISER AUTOMATISK UTHENTING AV TRANSAKSJONSDATA


```
<BoxPlot
  v-for="(customer, index) in customers"
  :key="customer.id"
  :customer="customer"
  :color="colors[index % colors.length]"
/>
```

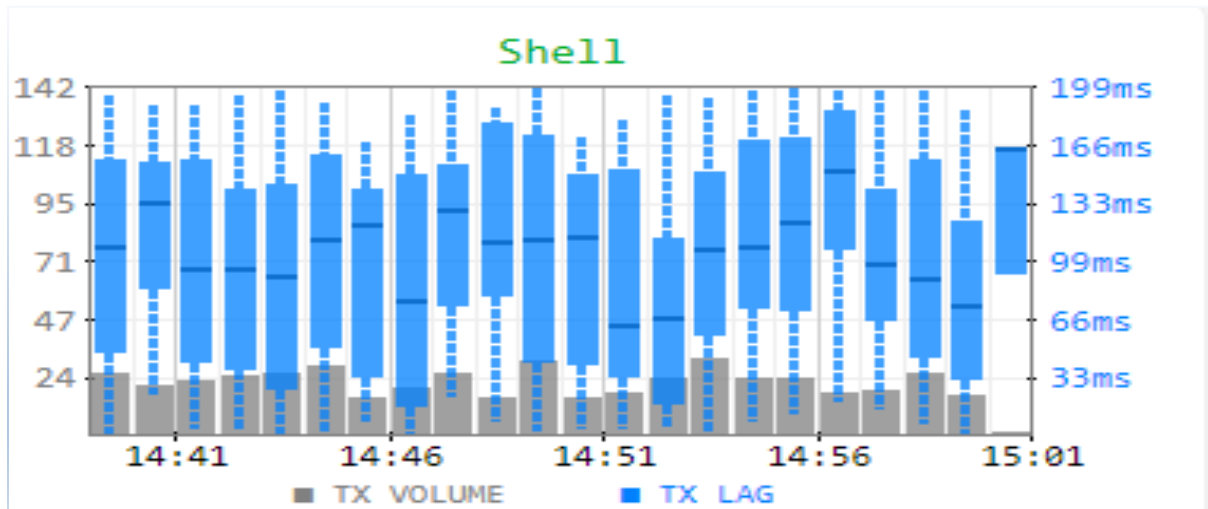
FIGUR 47: UTSNITT FRA BOXPLOTOVERVIEW. OPPRETNING AV BOXPLOT FOR HVER KUNDEGRUPPE.

```
if (index === 0) {
  let dateTime;
  if (col < chart.maxEpochs) {
    dateTime = new Date(customer.txData[col].epochStart);
  } else if (col === chart.maxEpochs) {
    dateTime = new Date(
      customer.txData[chart.maxEpochs - 1].epochStart + 60000
    );
  }
}
```

FIGUR 48: UTSNITT FRA BOXPLOT. VISER BRUKEN AV STARTTIDEN OG SLUTTIDEN FOR EPOKEN.

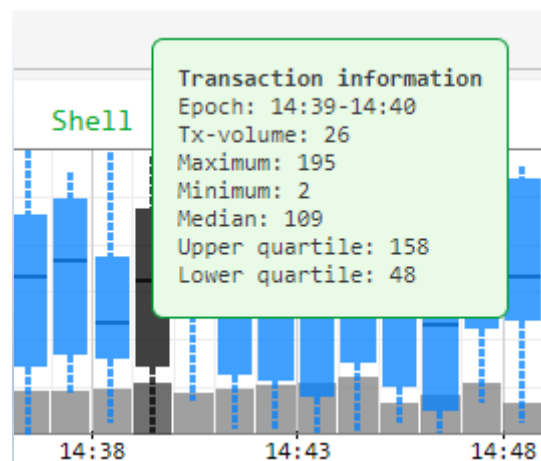
D3

For å visualisere transaksjonsdataene benytter applikasjonen seg av [D3-biblioteket](#). Opptegningen av linjegrafer, stolpediagrammer og boksdiagrammer er kodet fra bunnen av.



FIGUR 49: UTSNITT FRA BOXPLOTOVERVIEW. VISER BOXPLOT FOR KUNDEGRUPPEN SHELL.

Vi kan en se at grafen er sammensatt av data for *TX VOLUME* og *TX LAG*. *Volume* viser til antall transaksjoner i minuttet. *Lag* viser prosesseringstiden minutt for minutt, der den mørkeblå streken er medianen. Ved å holde musepekeren over grafen dukker et lite vindu opp som gir ekstra informasjon om epoken under pekeren:



FIGUR 50: HER SER VI TRANSAKSJONS INFORMASJONEN SOM HAR BLITT BRUKT TIL Å LAGE EPOKEN SOM VARTE FRA 14:39-14:40.

Mye av kreftene gikk på å implementere egendefinert visualisering ved bruk av D3. Alt ble programmert fra bunnen av:

- Opptegning av selveste grunnrammen
- Opptegning av rutenettet.
- Opptegning av boksene
- Visning av tekst ved aksene
- Dynamisk oppdatering
- Animasjon ved oppdatering av epoke.
- Visning av en boks med ekstra informasjon.

All implementasjonen førte til at det ble en god del matematiske beregninger, med selvkodete funksjoner.

```
c.axis.e = epochs;
c.axis.wf = c.axis.w / c.axis.e;
c.label.x = c.width / 2;
c.label.y = c.padTop + c.label.s * 0.33;
c.axis.x = c.padLeft;
c.axis.y = c.padTop + c.label.s;
c.axis.h = c.height - c.axis.y - c.axis.s - c.info.s - c.padBot;
c.info.x = c.label.x;
c.info.y = c.padTop + c.label.s + c.axis.h + c.axis.s + c.info.s + 4;
```

FIGUR 51: BRUKES FOR Å FINNE HVOR BREDT OG HØYT VISNINGEN KAN VÆRE I FORHOLD TIL SKJERMEN DEN VISES PÅ.

Bootstrap

Bootstrap ble benyttet for å gi frontend et solid grunndesign. Bootstrap ble også brukt for å forenkle prosessen med å tilpasse siden forskjellige skjermstørrelser og enkelt gi nettsiden en meny.

```
<nav
  class="navbar navbar-dark sticky-top flex-md-nowrap p-2"
  style="background-color: #343a40"
>
```

FIGUR 52: HER SER VI ET UTRAG FRA NAVBAR-KOMPONENTEN, DER VI HAR BRUKT BOOTSTRAP SIN NAVBAR MED MØRK FARGE.

Vi har for eksempel brukt Bootstrap sine formateringsmuligheter i implementeringen av statistikken som vises på toppen av dashbordet.

```
<div class="col-md-2 col-sm-4 col-xs-6 tile_stats_count">
```

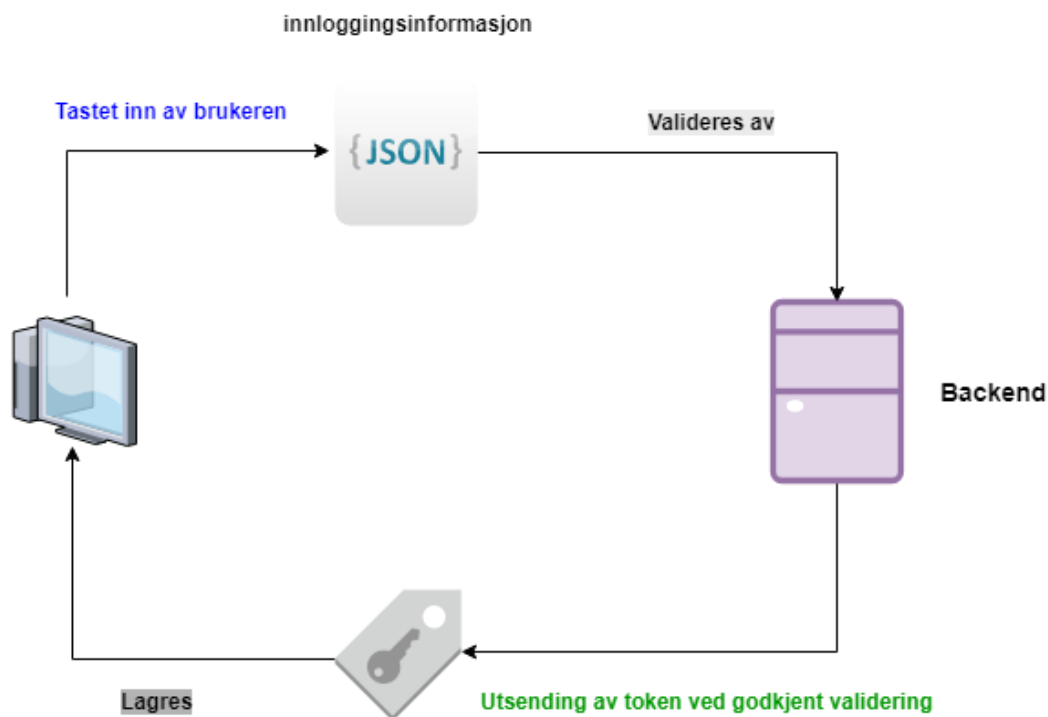
FIGUR 53: HER HAR VI DEFINERT HVOR MYE PLESS ELEMENTET SKAL TA I FORHOLD TIL HVILKEN STØRRELSE DEN VISES PÅ.

Transactions Today 22228	Traffic Volume (Tx/sec) 3	Total time 244 ms	Pos-Pay server 73 ms	Time to collector 0 ms	Time used collector 170 ms
-----------------------------	------------------------------	----------------------	-------------------------	---------------------------	-------------------------------

FIGUR 54: PÅ EN STOR SKJERM VIL ALLE RUTENE VISES PÅ ÉN LINJE

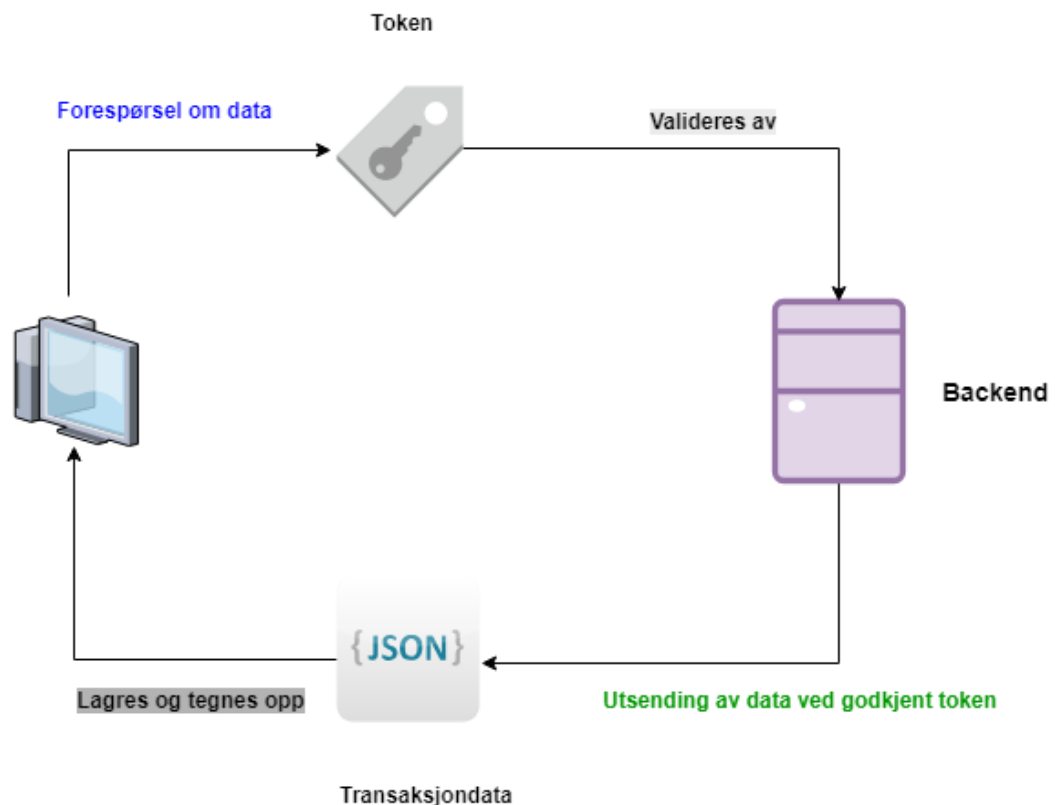
Systemet i helhet

Ressursene (HTML, CSS, JS) som utgjør frontend serveres fra en statisk webserver. Backend blir brukt til å autentisere, autorisere og sende ressursdata. Disse dataene blir brukt av frontend til å visualisere statusen på en praktisk måte. Utsvekslingen av informasjonen skjer på JSON-format. For en kort innføring i teknologiene som benyttes i systemet, se [Vedlegg D: Forklaring av brukte teknologier](#).



FIGUR 55: VISER HVORDAN INNLOGGINGEN FOREGÅR.

Når frontend vil ha transaksjonsstatistikk åpner det en forbindelse til backend over websocket. Etter å ha identifisert seg over socketen med token vil data sendes ut én gang i sekundet så lenge tokenet er gyldig.



FIGUR 56: FIGUR SOME VISER FLYTEN FOR MOTTAKELSE AV TRANSAKSJONSDATA.

Tokenet blir fornyet når det er ett minutt igjen av gyldighetstiden. Backend avslutter sesjonen og sender en kode som forteller at det er på tide å fornye tokenet. Det gamle tokenet blir sendt til en bestemt adresse i backend, sjekket og et nytt token blir utstedt. Det vil så bli opprettet en ny websocket-sesjon.

Konklusjon

Under prosjektperioden har vi klart å innføre de viktigste kravene til applikasjonen i henhold til oppdragsgivers ønsker. Produktet er en funksjonell webapplikasjon bestående av en separert backend og frontend. Løsningen benytter tokens for stateless autentisering og autorisering av brukere, utfører statistiske beregninger og sender transaksjonsdata i JSON-format fra backend. Frontend genererer informative grafer av denne dataen.

Det automatiske varslingsystemet som skulle bruke anomalier i transaksjonsbildet (eng: anomaly detection¹¹) for å oppdage stopp i transaksjonsflyten, ble ikke ferdigstilt under prosjektperioden. En slik løsning vil benytte seg av historisk transaksjonsdata og muligens maskinlæring. Denne funksjonaliteten er noe PayEx jobber med å implementere i API-et som applikasjonen skal benytte og vil ta noe tid å innføre.

Nedenfor er det en liste over hvilken funksjonalitet vi har implementert:

Funksjonelle krav

Bruker

- Webapplikasjon som er tilgjengelig på nettet.
- Dashboard for overordnet blick over transaksjonsbildet.
- Grafisk fremstilling av data ved bruk av linjediagram, stolpediagram og boksdiagram.
- Responsiv webapplikasjon som har blitt testet på mobiltelefon, nettbrett og veggdisplay/TV.

Utvikler

- Separert frontend fra backend for å forenkle videreutvikling av applikasjonen.
- Backend er lite avhengig av andre tjenester slik at flere instanser kan startes for å håndtere et varierende antall klienter.
- Informasjonen blir utvekslet mellom sikre kanaler (delvis ferdig).
- Transaksjonsdata blir sendt ut og vist i real-time slik at man kan se trender fortløpende.

Admin

- Vise ulik data på frontend basert på rettighetsnivåer (delvis ferdig).
- Andre systemer skal ikke bli påvirket av produktet.

¹¹ Sanjay Chawla, V. C. (2011). "Anomaly Detection: A Tutorial ". Retrieved 22 May, 2018, from http://icdm2011.cs.ualberta.ca/downloads/ICDM2011_anomaly_detection_tutorial.pdf

I løpet av arbeidsprosessen ble det lagt til flere krav angående feilhåndtering og logging. Disse kravene ble dermed implementert:

- Utveksling av feilmeldinger på et felles format.
- Visning av feilmeldinger til brukeren.

Ikke-funksjonelle krav

- Får overblikk over status ved å se på skjermen i mindre enn 5 sekunder.
- Data blir tegnet opp på frontend en gang i sekundet.
- Innloggingssesjonen er gyldig i 15 minutter.

Hva vi har lært

Oppgaven har vært en lærerik prosess som har gitt oss innsikt i programvareutvikling og utviklingsprosess i et arbeidsnært miljø. Vi har fått praktisere arbeidsmetodikken Scrum, deltatt i planlegging og brukt styringsverktøy som Jira. Gruppen har fått hevet kompetanse innen utvikling av webapplikasjoner med JSON Web Token, D3, og rammeverkene Spring og Vue. I tillegg har vi fått erfaring med å utføre tester med Selenium og Spring Testing.

Erfaringene vi har samlet under denne prosessen har gitt oss nyttig kompetanse for videre bruk i arbeidslivet.

Vedlegg

Vedlegg A: Testdokumentasjon	58
Vedlegg B: Prosessdokumentasjon	69
Vedlegg C: Kravspesifikasjon.....	90
Vedlegg D: Forklaring av brukte teknologier	98
Vedlegg E: Oppsett av servermiljø	102
Vedlegg F: Oppsett av utviklingsmiljø for backend.....	104
Vedlegg G: Oppsett av utviklingsmiljø for frontend	106
Vedlegg H: Feilmeldinger som sendes til frontend	109
Vedlegg I: Forslag/ønsker fra PayEx.....	111
Vedlegg J: Dagbok.....	112

Vedlegg A: Testdokumentasjon

For å gjøre videreutvikling enklere, samt ha en sikkerhetsmekanisme for å sørge for at endringer i programvare koden ikke ødelegger andre funksjoner, har vi implementert tester. Vi har brukt WebdriverIO med Silenum og for backend har vi skrevet tester med Spring Testing rammeverket.

Frontend

Vi har testet innlogging ved riktig og feil legitimasjon. Vi har testet om det er mulig å navigere fra forskjellige views ved bruk av menyen. Vi sjekket også om tittelen til de forskjellige sidene var riktig. Trenger en del konfigurasjon og installasjon før det kan tas i bruk.

Oppsett av WebdriverIO

WebdriverIO og Selenium ble installert ved bruk av npm:

```
npm install webdriverio
```

FIGUR 57: KODESNUTT SOM VISER INSTALLASJON AV WEBDRIVERIO.

```
npm install selenium-standalone
```

FIGUR 58: KODESNUTT SOM VISER INSTALLASJON AV SELENIUM ENKELT VERSJON.

Når installasjonen var ferdig var det nødvendig å sette opp WebdriverIO miljøet ved bruk av powershell:

```
PS C:\codebase\pos-monitor-frontend> ./node_modules/.bin/wdio config
=====
WDIO Configuration Helper
=====

? Where do you want to execute your tests? (Use arrow keys)
> On my local machine
  In the cloud using Sauce Labs, Browserstack or Testingbot
  In the cloud using a different service
  I have my own Selenium cloud
```

FIGUR 59: NÅR VI KJØRTE GITT KOMMANDOLINJE FIKK VI BESKJED OM Å VELGE STED Å INSTALLERE. VERKTØYET BLE INSTALLERT LOKALT PÅ MASKINEN.

Installasjonsveilederen ba oss om å velge nettleser for å kjøre testene i. Standard nettleseren ble satt til Google Chrome. Vi valgte så hvor testene og konfigurasjonen skulle ligge. Testene ble satt til å kjøres i **pos-monitor-frontend/test/specs/**.

Konfigurasjonsfilen til WebdriverIO ligger i prosjektmappen til frontend koden med navnet **./wdio.conf.js**. I konfigurasjonsfilen har vi definert hvilken port servicen skal kjøre, på hvilken driver, og hvor testene ligger. Maksimumstiden en test kan bruke blir også definert i konfigurasjonsfilen.

```
port: '9515',
path: '/',
services: ['chromedriver],
```

FIGUR 60: UTSNITT FRA KONFIGURASJONSFILEN.

```
PS C:\codebase\pos-monitor-frontend> ./node_modules/.bin/wdio wdio.conf.js
DevTools listening on ws://127.0.0.1:12333/devtools/browser/df5add4f-c131-4603-9a03-820d0230b3a5
DevTools listening on ws://127.0.0.1:12274/devtools/browser/a95b6173-1413-403a-8deb-3a0289fe140f
DevTools listening on ws://127.0.0.1:12182/devtools/browser/184a23a4-128f-4796-a364-0d7c0d944068
.....
!! passing (39.50s)
```

FIGUR 61: KODESNUTT SOM VISER AT TESTENE HAR PASSERT.

Det er opprettet tre testklasser. Hver av disse testklassene har sin hovedoppgave:

Testklasse	Funksjonalitet
LoginFormationTest	Sjekker om innloggingssiden har riktig oppbygging.
LoginFunctionTest	Sjekker om det er mulig å logge inn, samt hva som skjer om feil brukernavn eller passord blir tastet.
sidebarTest	Sjekker om det er mulig å navigere ved bruk av sidemenyen

Testene skrives ved å bruke WebDriverIO sine funksjoner. Disse funksjonene kan simulere brukerhandlinger som å trykke på knapper, fylle ut skjemaer, hente ut tekst fra et element eller klasse med mer.

```
describe('Correct display of Information', function() {
  it('render placeholder Username', function() {
    browser.url('http://localhost:8080');
    let placeholder = browser.getAttribute('#inputUsername', 'placeholder');
    assert.equal(placeholder, 'Username');
  });
});
```

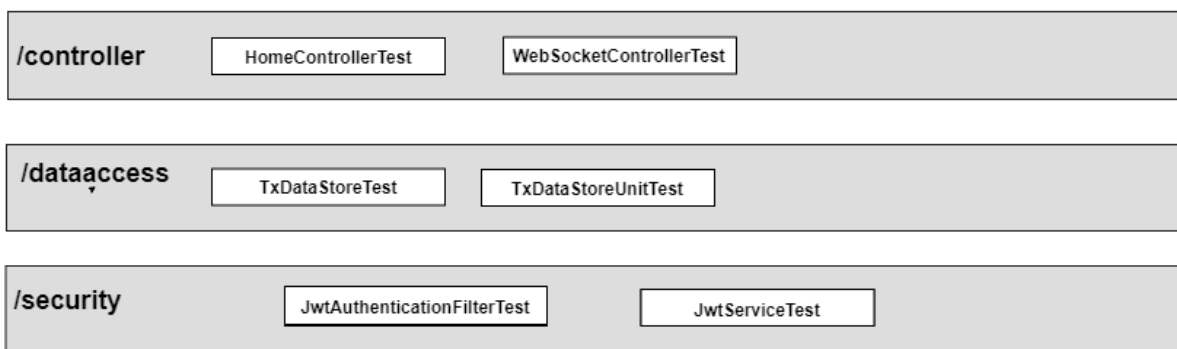
FIGUR 62: UTSNITT FRA TESTKLASSEN LOGININFORMATIONTEST. HENTER UT DATA FRA BRUKERNAVN FELTET OG SJEKKER HVA DEN ER VED INNLASTING.

Tester utført og deres status

Hva	Ønsket hendelse	Status
Innlogging ved riktig brukernavn og passord	Innloggingen blir vellykket, og du blir sendt videre til status siden	OK
Innlogging ved feil brukernavn og passord	Innloggingen feiler, og det kommer opp en melding om at tastet brukernavn og passord er feil.	OK
Sjekker tittelen til hovedsiden	Hovedsidens tittel ved innlastning er "POS Monitor".	OK
Sjekker om hovedsiden inneholder felt for inntasting av brukernavn og passord	Hovedsiden skal til enhver tid inneholde felt for inntasting av brukernavn og passord.	OK
Sjekker teksten til påloggingsknappen	Hovedsidens innloggingsknapp har teksten "SIGN IN"	OK
Visning av meny	Ved å trykke på åpne/lukke knappen skal menyen kunne åpnes og lukkes	OK
Sjekker navigasjonen til hovedmenyen: -Klikking på "merchants" linken	Etter suksessfull innlogging, skal du kunne åpne menyen trykke på "merchants" linken og komme til graf siden.	OK
Sjekker navigasjonen til hovedmenyen: -Klikking på "dashboard" linken	Etter suksessfull innlogging, skal du kunne åpne menyen trykke på "dashboard" linken og komme til dashboard.	OK
Sjekker navigasjonen til hovedmenyen:	Etter suksessfull innlogging, skal du kunne	OK

Hva	Ønsket hendelse	Status
-Klikking på "BoxPlot" linken	åpne menyen trykke på "Boxplot" linken og komme til visning av boxplots.	
Trykke på "SIGN OUT"	Ved å trykke på sign out skal du komme tilbake til innloggingssiden.	OK

Backend



FIGUR 63: OVERSIKT OVER TEST KLASSENE SOM BLE UTVIKLET.

Spring Testing

Under utviklingen ble Spring Testing benyttet for å utføre integrasjonstesting av de forskjellige klassene. Spring Testing bygger på design mønsteret IoC (Inversion of Control) og er en naturlig del av Spring rammeverket. Design mønsteret gjør koblingen løsere ved at klassene implementerer et eller flere interface som benyttes ved dependency injection. Ved at koblingen blir løsere forenkles også testingen av programmet slik at man enklere kan isolere komponenter og dens avhengigheter.

Unit testing

Vi har benyttet unit testing som er å teste individuelle enheter av programmet. Dette er for å verifisere at funksjonene virker som de skal. For å forenkle prosessen med å lage tester, laget vi en abstrakt klasse for unit testing.

Denne abstrakte klassen vil bli arvet av de klassene som utfører unit testing og gjenbruke metodene som er skrevet i den abstrakte klassen.

```

/**
 * Base class for functionality that can be tested as separate tests without involving Spring.
 */
@RunWith(JUnit4.class)
public abstract class UnitTest {

    protected final Logger log = LoggerFactory.getLogger(this.getClass());

    /**
     * Used for asserting that a known exception is thrown.
     */
    @Rule
    public final ExpectedException thrown = ExpectedException.none();

    /**
     * Runs before every test is invoked.
     */
    @Before
    public void setup() throws Exception {
        // Common setup
    }

    /**
     * Runs after each test is completed.
     */
    @After
    public void teardown() throws Exception {
        // Common teardown
    }

    /**
     * Customized {@link Assert#assertThat(Object, Matcher)} with a more useful signature, allowing
     * say a Map to be compared to an ImmutableMap without any casting.
     */
    @SuppressWarnings("unchecked")
    protected static <T> void assertThat(T actual, Matcher<? extends T> matcher) {
        Assert.assertThat(actual, (Matcher<? super T>) matcher);
    }

    /**
     * Helper for matching empty optionals.
     */
    protected static Matcher<Optional<Object>> isEmpty() {
        return Matchers.equalTo(Optional.empty());
    }
}

```

FIGUR 64: UTDRAK FRA EN ABSTRAKT KLASSE UNITTEST. VISER HVORDAN EN STANDARD UNIT TEST KLASSE SKAL SE UT.

Vi har laget unit testing av klassene som skal ta seg av sikkerhet, lagring og oppdatering av transaksjonsdata. Det er laget en unit test for klassen JwtService som er en tjeneste for generering og validering av JSON Web tokens.

Integrasjonstesting

Integrasjonstesting er å teste systemets individuelle komponenter. Vi starter først Spring i test modus, deretter kjører vi de nødvendige testene.

```
/**
 * Starts Spring in "test mode".
 */
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@ActiveProfiles({"integration-test", "test"})
public abstract class IntegrationTest extends UnitTest {

}
```

FIGUR 65: UTSNITT FRA INTEGRATIONTEST. STARTER TEST APPLIKASJONEN PÅ EN TILFELDIG RANDOM PORT OG SETTER DE NØDVENDIGE PROFILENE.

I integrasjonstesting har vi utført testing på:

- WebSocket
 - Utsending av data ved riktig og feil token.
 - Formatet på utsendt data.
 - Test på feil forespørsel.
- DataStore
 - Testing av riktig epoke tid.
 - Testing av generert data og dens format.
 - Testing på at observante oppdaterer seg.
- Autentisering
 - Påloggingsforespørsel ved feil og riktig brukernavn.
 - Utsending av tmjoken.


```

@Test
public void it_should_connect_to_websocket() throws ExecutionException, InterruptedException {
    WebSocketClient client = new StandardWebSocketClient();
    CountdownLatch id = new CountdownLatch(1);
    TextWebSocketHandler wsHandler = new TextWebSocketHandler() {
        @Override
        public void afterConnectionClosed(WebSocketSession session, CloseStatus status)
            throws Exception {
            super.afterConnectionClosed(session, status);
            closeStatus = status;
        }
    };
    WebSocketSession session = client.doHandshake(wsHandler, wsuri).get();
    assertThat(session.isOpen(), equalTo(true));
}
}

```

FIGUR 66: UTSNITT FRA WEBSOCKETCONTROLLERTEST. SJEKKER OM MAN FÅR EN FORBINDELSE TIL BACKEND OVER WEBSOCKET

Testene og deres status

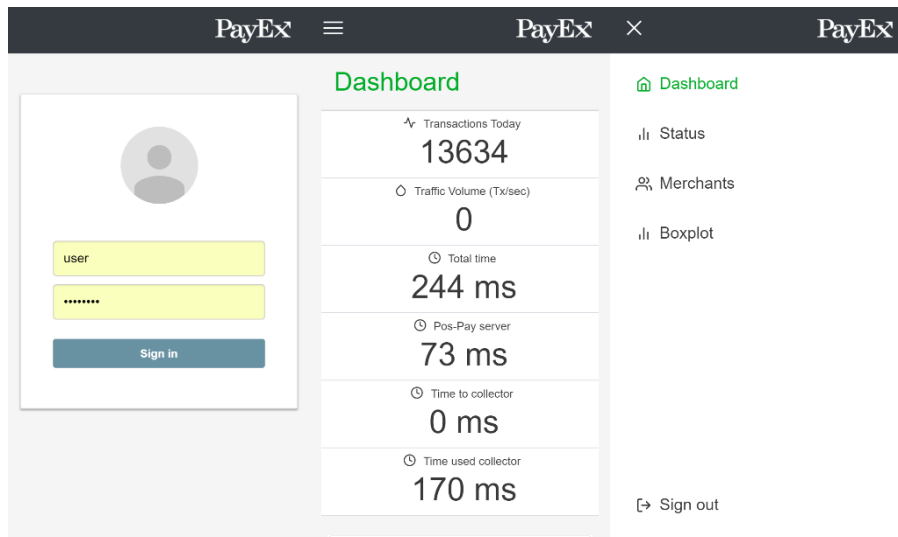
Hva	Ønsket hendelse	Status
Sende en forespørsel til localhost:8098/exec	Det blir kastet et unntak, og brukeren får feilmelding sendt tilbake automatisk.	OK
Koble seg til websocket	Testen klarer å koble seg til websocket på adressen localhost:8098/Stream	OK
Sende forespørsel med dårlig header til websocket	Tilkoblingen blir avsluttet og du får status "Closestatus.BAD_DATA" tilbake som tilsvarer kode 1007.	OK
Sende en misformet header til webscoket	Tilkoblingen blir avsluttet og du får status "Closestatus.BAD_DATA" tilbake som tilsvarer kode 1007.	OK

Hva	Ønsket hendelse	Status
Sende korrekt token til websocket	Tokenet blir validert, og det kommer inn strøm med data.	OK
Sende korrekt token til websocket, og få riktig formet data.	Tokenet blir validert, og det kommer inn korrekt formatert strøm med data.	OK
Sende feil forespørsel imens dataen strømmer.	Tilkoblingen blir avsluttet og du får status "Closestatus.BAD_DATA" tilbake som tilsvarer kode 1007.	OK
Mottakelse av Epoker.	Epokere skal returneres i JSON-format.	OK
Mottakelse av nåværende epoke.	Mottar nåværende epoke i JSON-format.	OK
Generasjon av transaksjoner.	Transaksjonene blir generert og lagt inn i en liste.	OK
Mottatt data har korrekt format.	Data som blir mottatt er i riktig format og har mediatype JSON.	OK
Notifikasjon av observasjoner.	De objektene som observerer dataene i TxDataStore blir oppdatert.	OK
Nullstille transaksjoner for kunder.	Nullstiller alle transaksjonene for en gitt kunde.	OK
Sender pålogging forespørsel med feil media-type.	Brukeren skal få en feilmelding tilbake med kode 400.	OK
Påloggingsforsøk med feil brukernavn og passord.	Brukeren får returnert feilmelding med kode 403.	OK

Hva	Ønsket hendelse	Status
Sender påloggings forespørsel med misformet header.	Brukeren skal få en feilmelding tilbake med kode 400.	OK
Påloggingsforsøk med riktig brukernavn og passord.	Brukeren får token tilbake.	OK
Tokenet blir generert.	Servicen generer et token.	OK
Tokenet blir generert med riktig utløpstid.	Servicen genererer et token med utløpstid på 15 minutter.	OK
Tokenet blir generert med riktige roller.	Servicen generer et token med riktig roller som brukeren har.	OK
Forespørsel med utløpt token	Det blir kastet et unntak hvis det kommer en forespørsel med utløpt token.	OK
Fornyelse av token.	Fornyelse av token skal bare gå ann i løpet av et minuttts intervaller før utløpstiden.	OK
Fornyelse av utløpt token.	Det skal ikke gå an å fornye et utløpt token.	OK
Validering av token.	Validering av token ved gyldig og ugyldig tid.	OK

Testing av applikasjonen

Applikasjonen ble testet både på mobil og PC. Navigasjonen ble testet, fargekontrast samt responsiviteten av designet.



FIGUR 67: SKJERMDUMP AV INNLOGGINGSSIDEN, DASHBOARD OG MENYEN AV HVORDAN DET SER UT PÅ MOBILTELEFON.

Testing av fargekontraster

Fargekontrastene på siden ble testet ved bruk av nettsiden:

<http://www.checkmycolours.com/>

Her testet vi til enhver tid om fargene var kompatible og hadde riktig kontrast.

Testing done on 2 elements

Congratulations!
All the elements pass the test!

full report only errors Click on the rows to test other colours

Node	Foreground	Background	Sample	Contrast Ratio	Brightness difference	Color difference
BODY	#000000	#FFFFFF	Sample Text	21:1 AAA	255	765
• DIV id='app'	#000000	#FFFFFF	Sample Text	21:1 AAA	255	765

FIGUR 68: PASSERING AV FARGEKONTRAST TESTER.

Vedlegg B: Prosessdokumentasjon

Forord

I denne delen av rapporten vil vi ta for oss alle utviklingsfaser av prosjektet. Vi vil gi et kort resyme av prosjektets rammebetingelser og verktøyene som ble brukt til prosjektstyring. Vi vil gi innsikt i beslutningene tatt sammen med oppdragsgiver på utfordringer som oppstod underveis.

Valg av oppgave

Vi startet høsten 2017 med å lete etter en oppdragsgiver til bacheloroppgaven. Under denne prosessen var det viktig å finne en oppgave som passet vårt nivå samt utfordrende nok til å gi oss et kompetanse- og kunnskapsløft. Det var viktig at prosjektets størrelse dekket semesterets lengde slik at vi ikke ble stående med et for lite prosjekt.

I løpet av september hadde vi vært i kontakt med flere oppdragsgivere, deriblant PayEx. Fra PayEx fikk vi et profesjonelt og meget positivt inntrykk med ganske differensierte oppgaver. Valget vårt falt på "Develop a new transaction monitor" fordi oppgaven hadde en fin balanse av kompleksitet, omfang og fleksibilitet. Under møtet med PayEx i november ble vi presentert mer i detalj om deres ønsker og hvordan de ville tilrettelegge oppgaven for oss. Vi ble anmodet til å jobbe smidig og å være frimodige med forslag til funksjonalitet for applikasjonen. Det ble presisert at det ville bli holdt regelmessige møter gjennom hele arbeidsprosessen for å sørge for god kommunikasjon mellom oppdragsgiver og oss.

Prosjektstart og plan

Prosjektet startet i begynnelsen av januar hvor kravspesifikasjonenpes og fremdriftsplanen ble utarbeidet. I retningslinjene for prosjektet ble det bestemt at vi skulle bruke Scrum som arbeidsmetodikk og programmet Jira for prosjektstyring.

Fremdriftsplan:

Periode	Arbeid
Januar	Planlegging, styringsdokumenter
Februar – Mai	Utvikling & dokumentasjon
Mai	Levering av bachelor rapport
Slutten av Mai - Juni	Presentasjon

Timeplan:

Dag	Tid
Mandag	0900-1700, sprint planleggingsmøte annenhver uke.
Onsdag	0900-1700
Torsdag	0900-1700
Fredag	0900-1700, sprint review annenhver uke.

Arbeidsmetodikk

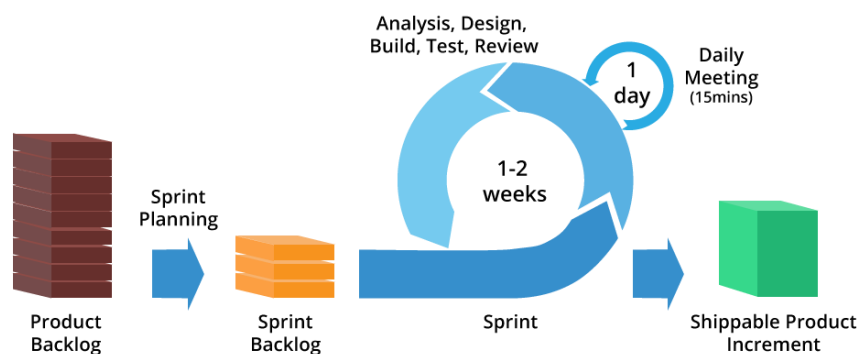
Scrum er en smidig (eng: agile) arbeidsmetodikk hvor man utvikler systemet iterativt gjennom en produktkø (eng: backlog). Metodikken går ut på at man legger til “brukerhistorier” med funksjoner man ønsker implementert, i produktkøen. Når utviklingsteamet så setter sammen en ny sprint, velger teamet funksjoner fra produktkøen basert på prioritet og et estimat. Dette blir målene for sprinten. Ved slutten av sprinten demonstrerer Scrum-teamet inkrementet til produkteier. Dette kalles for en “sprint review”.

Det finnes en rekke rammeverk som utdyper og fastslår hvordan man jobber smidig, blant annet Kanban og Extreme Programming. I vårt tilfelle falt utviklingsmetodikken på Scrum fordi metodikken tar høyde for at man har en aktiv kommunikasjon mellom produkteier og utviklingsteamet. På den måten reduseres misforståelser og samtidig kan produkteier gi tilbakemeldinger og innspill på sprint presentasjonene.

Scrum-teamet består av produkteier, scrum-master og utviklingsteamet. Scrum-masteren skal lede daglig standup møter, fjerne hindringer som demper lagets fremgang og beskytte teamet fra distraksjoner.

Rolle	Navn
Scrum master	Ali Arfan
Produkteier	Dani Berntzen
Utviklingsteam	Ali Arfan John Husfloen Håvardstun Kent Erlend Bratteng Knudsen Shamil Magomadov

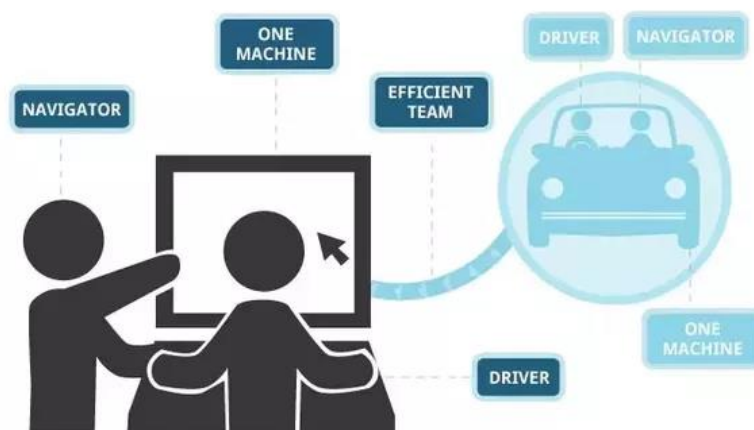
Agile Software Development



FIGUR 69: DE FORSKJELLIGE FASENE UNDER SMIDIG UTVIKLING I ET PROSJEKT¹².

Parprogrammering

Utviklingsteamet jobbet ofte i par. Dette bidro til at vi fant løsninger på problemer raskere enn om vi hadde sittet hver for oss. Parprogrammeringen virket ved at et medlem hadde rollen som koder mens den andre veiledet og hjalp til. Etter en stund ble rollene byttet om. I parprogrammering blir personen som koder betegnet som sjåføren og den som veileder omtalt som navigator.



FIGUR 70: ILLUSTRASJON AV PARPROGRAMMERING I AKSJON¹³

¹² Offshore Software Solutions (2017). "Agile Software Development". Retrieved 22 May, 2018, from <http://www.offshoresoftware.solutions/blog/tag/agile/>

Daily stand up

Gruppen holdt daily standup møter hver dag kl. 0900. Møtet varte i opptil 15 minutter der vi diskuterte hva vi hadde jobbet med dagen før, hvilken problemer vi hadde støtt på og hva vi hadde tenkt å jobbe med. Møtet fungerte som et hjelpemiddel der vi visste hva alle drev med og at alle fikk hjelp med det de trengte. Når møte var ferdig ble det plukket ut oppgaver fra produktkøen og disse ble utført.

Dagbok

Under hele prosjektperioden ble det ført dagbok. Der ble ned nedskrevet hver uke hva vi hadde jobbet med, oppnådd og tanker vi satt igjen med rundt arbeidet. Dagboken kan finnes som [Vedlegg J: Dagbok](#).

¹³ Jain, P. (2017). "What is the difference between pair programming and parallel programming?". Retrieved 22 May, 2018, from <https://www.quora.com/What-is-the-difference-between-pair-programming-and-parallel-programming>

Verktøy og hjelpemidler

Prosjektstyringsverktøyet Jira

Vi brukte Jira til å planlegge sprinter og distribuere oppgaver på tvers av utviklingsteamet. En av de store fordelene med Jira var at vi kunne prioritere og diskutere arbeidsfordelingen, se status på hva som var blitt utført og hva som ble jobbet med. Jira var også veldig praktisk og enkelt å bruke, noe som gjorde at vi kom raskt i gang med sprint planleggingen.

The screenshot shows a Jira sprint interface. At the top, there's a header for 'PSB2 Sprint 8' with '0 issues' and buttons for 'Start sprint', 'Plan sprint', and a menu icon. Below this is a dashed box with the instruction: 'Plan a sprint by dragging the sprint footer down below some issues, or by dragging issues here.' There's a '+ Create issue' button and a separator line. Below the separator, it says '0 issues Estimate 0'. A 'Backlog 4 issues' section is visible with a 'Create sprint' button. The backlog contains four issues:

Issue Description	Category	Key	Priority	Estimate
As a user, I want to search for a customer so that I get quick navigation.	Frontend	MNTR-9	High	-
As a developer I want dynamic discovery of existing meta-data to push data to frontend so I don't need to hardcode a lot of changes for new metadata.		MNTR-80	High	-
As a developer I want to run frontend tests in isolation.		MNTR-87	Medium	20
As an Admin I want secure information exchange between services.		MNTR-17	High	-

There's a '+ Create issue' button at the bottom of the backlog.

FIGUR 71: UTSNITT FRA JIRA. VISER PRODUKTKØEN OG EN TOM SPRINT I JIRA. HER KAN EN ESTIMERE TIDEN, GI KOMMENTARER, FLYTTE OVER FUNKSJONELLE KRAV OG OPPGAVER MED MER.

The screenshot shows a Jira Kanban board with three columns: 'TO DO', 'IN PROGRESS', and 'DONE'. The 'TO DO' column has one issue: 'MNTR-8' with '2 sub-tasks'. The 'IN PROGRESS' column has three issues:

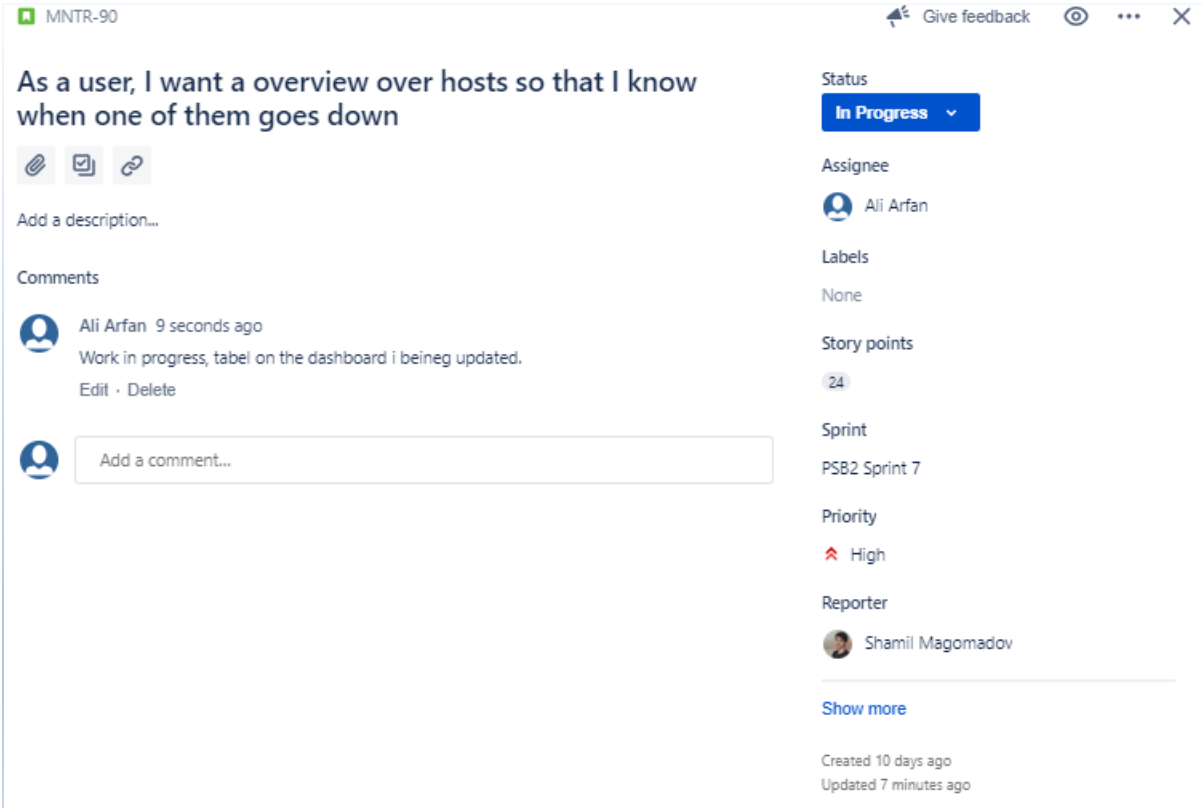
- 'Show various graphs for a customer' (MNTR-68)
- 'Information about a specific customer' (MNTR-75)
- 'As a user, I want an overview over hosts so that I know when one of them goes down' (MNTR-90, estimate 24)

The 'DONE' column is empty. Below the board, there's a section for 'Other Issues 5 issues' with three issues:

- 'As a user, I want to see daily/weekly/etc metrics (avg/max throughput/latency/...) /for customers' (MNTR-16, Frontend, estimate 32)
- 'As a sentry [vakt], I want to be alerted when something is not OK.' (MNTR-13, Backend, estimate 24)
- 'Create new issues to backlog' (MNTR-91)

FIGUR 72: UTSNITT FRA JIRA. VISER HVORDAN DET SER UT NÅR EN SPRING ER I GANG.

I løpet av sprinten kan en velge hvem som skal utføre oppgaven, kommentere og mye mer.



MNTR-90

Give feedback

As a user, I want a overview over hosts so that I know when one of them goes down

Add a description...

Comments

Ali Arfan 9 seconds ago
Work in progress, tabel on the dashboard i beineg updated.
Edit · Delete

Add a comment...

Status
In Progress

Assignee
Ali Arfan

Labels
None

Story points
24

Sprint
PSB2 Sprint 7

Priority
High

Reporter
Shamil Magomadov

Show more

Created 10 days ago
Updated 7 minutes ago

FIGUR 73: VISER ENDRINGSMULIGHETER FOR BRUKERHISTORIE NUMMER 90.

Integrasjon av Jira og GitHub

Hos PayEx er Jira integrert med GitHub. Dette gjør at man kan angi nummer til brukerhistorien i commit meldingen når koden pushes mot Git. Dette vil vises i Jira.

```
git commit -m "MNTR-81: Added meta in json data stream for pulling highest tx delay"
```

FIGUR 74: VISER EN COMMIT MELDING MED NUMMERET TIL BRUKERHISTORIEN. DETTE GJØR AT JIRA KAN REGISTRERE ARBEIDET I LOGGEN.

MNTR-81: 9 unique commits

pos-monitor-backend

[Show all files](#)

Author	Commit	Message	Date	Files
288552c		MNTR-81: Added meta in json data stream for pulling highest tx delay	04/Apr/18	2 files

pos-monitor-frontend

[Show all files](#)

Author	Commit	Message	Date	Files
b148cd3		MNTR-81: Added the opacity member - chartColor - to gridchart in initChart function.	06/Apr/18	1 file
848d981		MNTR-81: Added yellow color transition to the Status-graph when time since last transaction is closin...	06/Apr/18	1 file
fde9ba2		MNTR-81: Status graph is now using D3 library. Graphs get increasingly red as time since last tx increa...	06/Apr/18	1 file
ab4f2ea		MNTR-81: Added two descriptive labels to D3 chart - tx volume and tx lag.	06/Apr/18	1 file
85a7ca5		MNTR-81: The lag-bars on the D3 chart should no longer draw outside the chart.	04/Apr/18	1 file
499a83a		MNTR-81: D3 chart now display the median value also on its chart.	04/Apr/18	1 file
ccd463c		MNTR-81: Changed some formatting on text	04/Apr/18	1 file
e7814e6		MNTR-81: Added y-axis values on the right side of D3 chart. Not showing correct values yet.	04/Apr/18	1 file

[Close](#)

FIGUR 75: UTSNITT FRA JIRA SOM VISER ALLE ENDRINGER FOR BRUKERHISTORIE NUMMER 81.

WebStorm

WebStorm er et integrert utviklingsmiljø levert av JetBrains mye brukt av frontend utviklere. Programmet kommer med et rikt utvalg av plugins og støtter flere typer formateringsverktøy. En fordel med WebStorm er at programmet automatisk re-kompilerer koden ved endringer slik at en kan se disse i sanntid. WebStorm har støtte for webpack, en smart og kompakt webserver, for å hoste frontend kode.

IntelliJ IDEA

IntelliJ IDEA er et integrert utviklingsmiljø levert av JetBrains med støtte for Maven, Java Spring med mer. Under prosjektet ble IntelliJ IDEA brukt for å utvikle backend. Verktøyet kommer med integrert formateringsverktøy og har også et rikt utvalg av plugins.

Postman

Postman er et hjelpeverktøy som sender og mottar HTTP pakker fra en web service. Applikasjonen ble tidlig brukt til i prosjektfasen for å simulere forespørsler til backend. Dette var nødvendig da vi ikke hadde noen frontend løsning, men ønsket å prototype forespørsler om innlogging, mottakelse av transaksjonsdata og token. Dette gjorde at vi kunne oppdage formateringsfeil, samt se om det ikke blir sendt unødig data.

GitHub

GitHub er et versjonskontrollverktøy for å holde orden på endringer som er gjort. Utviklingsteamet brukte verktøyet hyppig for å laste opp kode slik at det kunne bli lagret, få en versjon og se hvilke endringer som har blitt gjort. Dette gjorde at vi kunne gå tilbake til gamle versjoner av kode for å hente ut det vi trengte, eller eventuell få fram en versjon der en feil ikke eksisterer.

For å bruke GitHub brukte vi kommandolinjen for å hente ned, laste opp og se endringer på koden. Gruppen var medlem av oppdragsgiver sin GitHub-organisasjon.

Google Docs

Google Docs er et tekstredigeringsverktøy, som lar flere personer redigere på et dokument samtidig. Dette gjorde at vi ikke trengte å slå sammen flere rapporter, og alt var tilgjengelig over nettet. Google Docs var et nyttig verktøy brukt under arbeidsprosessen til å skrive dagbok og andre nødvendige dokumenter.

Slack

Slack er et kommunikasjonsverktøy brukt mellom oppdragsgiver og utviklingsteamet. PayEx har integrert Slack med TeamCity slik at tilbakemeldinger på testene TeamCity kjører er enklere tilgjengelig. Dette bidrar til å heve kvaliteten på koden fordi koden blir analysert av et ekstra lag. PayEx har også integrert

Slack med GitHub slik at en kan se de nyeste endringene, når de ble gjort og av hvem.

NPM

NPM er et programvareregister som tilbyr utviklere over 600,000 JavaScript-utvidelser. Utvidelsene lastes ned i form av pakker som utvikler importerer inn i applikasjonen.

NodeJS

NodeJS (Node JavaScript) brukes for å kjøre JavaScript kode på servere. Dette brukte vi for å kjøre frontend applikasjonen lokalt ved bruk av Webpack. Dermed slapp vi å laste opp kode på en webserver for å kjøre den, noe som sparer tid og krefter, og lar alle utviklerne jobbe med forskjellige ting lokalt på sin egen maskin.

Webpack

Webpack pakker og prosesserer kildekoden til frontend ved hjelp av forskjellige plugins. Blant annet brukes Babel til å transpilere kildekoden skrevet i nye EcmaScript-versjoner til JavaScript som kan kjøres på eldre nettlesere. Webpack sin dev-server brukes for å kjøre en lokal server som automatisk bygger prosjektet når en fil endres.

Arbeidsfordeling

Utviklingen av applikasjonen skjedde i et grupperom hos PayEx. Dette gjorde at vi fikk arbeidsroen som vi trengte, samt gjorde kommunikasjon innad i gruppen veldig enkel.

Arbeidet ble fordelt med hensyn til funksjonene som ble opprettet i sprinten og nivået til hvert enkelt gruppemedlem. Som regel var det forskjellige oppgaver som skulle bli utført. Oppgavene kunne blant annet gå ut på design, jobbe i frontend, lage tester i backend eller utforske graf biblioteker. Vi diskuterte i gruppen hvem som egnest seg best til hver oppgave og brukte dette til å fordele oppgavene mellom gruppemedlemmene. Hvis et medlem av gruppen ble ferdig med sin egen oppgave, bidro medlemmet med å hjelpe sidemann eller fortsette på en annen oppgave.

Utviklingsprosess

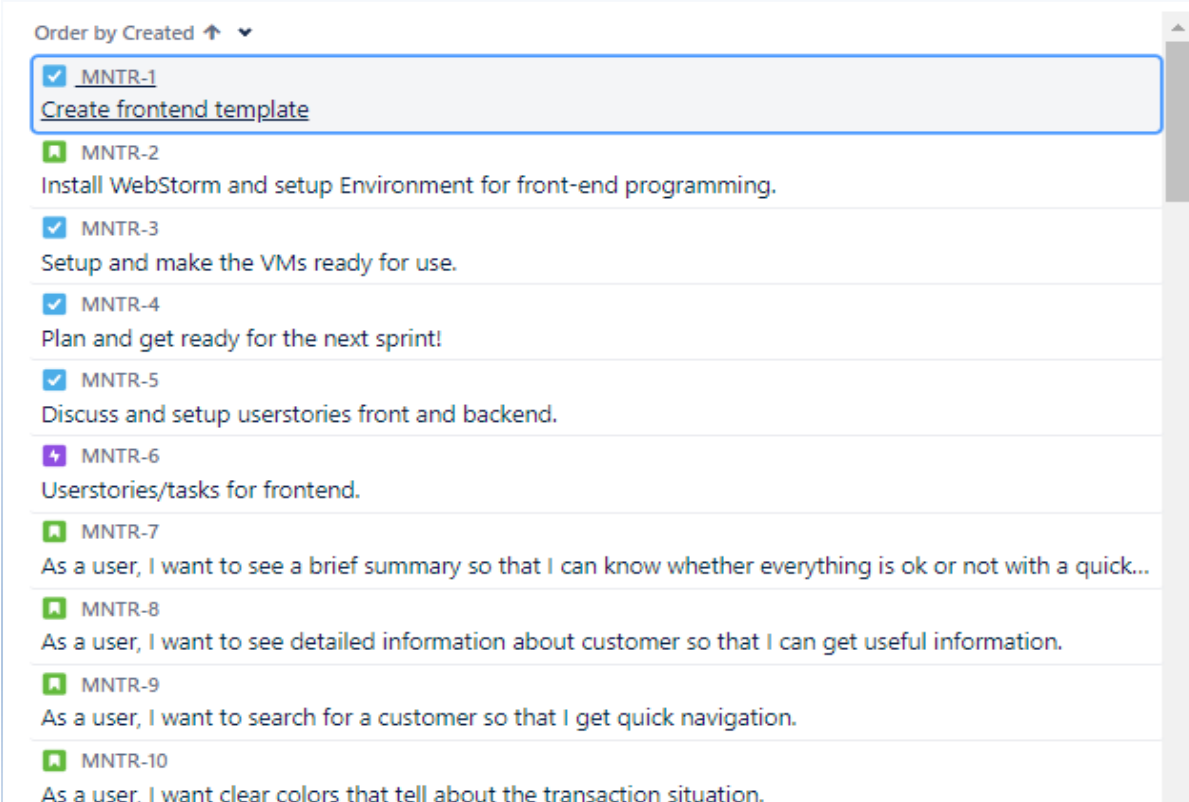
Vi startet utviklingsprosessen tidlig og la til fortløpende nye funksjonaliteter. Planleggingen startet tidlig i Januar der vi hadde møte med PayEx og diskuterte oppgaven i detalj. Vi ble enige om kommunikasjonskanaler, hovedfunksjoner i webapplikasjonen, samt prosessen og fremtidsplanen. Utviklingsprosessen hadde iterasjoner på 2 uker.

I denne delen av rapporten går vi igjennom iterasjonene som vi fullførte gjennom utviklingsprosessen. Vi vil gi et innblikk i forholdene rundt arbeidet, hva vi har fått til og hvilke utfordringer vi støttet på.

Oppstart

Planleggingen og arbeidet med forprosjektrapporten startet tidlig. Vi hadde et bilde over hvordan webapplikasjonen skulle se ut, men vi hadde lite kunnskap om de forskjellige leddene i applikasjonen. Dette førte til at vi bestemte oss for å ha hyppige dialoger med oppdragsgiver. Slik visste vi hva som skulle utvikles og tilegnet oss kunnskapen deretter for å komme i mål. Mye av tiden i begynnelsen av januar gikk til å sette seg inn i rammeverkene Java Spring og Vue.js.

Når vi var ferdig med å lære det grunnleggende om de forskjellige rammeverkene, satte vi i gang med å legge til brukerhistorier og oppgaver i Jira. Vi hadde fått klar beskjed om at funksjonaliteten og prioriteringen på utførelsen av oppgavene kunne forandre seg. Dette tok vi høyde for og kom fram til noe som viste seg å være de grunnleggende funksjonene i applikasjonen:



Order by Created ↑ ▾

- MNTR-1
Create frontend template
- MNTR-2
Install WebStorm and setup Environment for front-end programming.
- MNTR-3
Setup and make the VMs ready for use.
- MNTR-4
Plan and get ready for the next sprint!
- MNTR-5
Discuss and setup userstories front and backend.
- MNTR-6
Userstories/tasks for frontend.
- MNTR-7
As a user, I want to see a brief summary so that I can know whether everything is ok or not with a quick...
- MNTR-8
As a user, I want to see detailed information about customer so that I can get useful information.
- MNTR-9
As a user, I want to search for a customer so that I get quick navigation.
- MNTR-10
As a user, I want clear colors that tell about the transaction situation.

MNTR-10	As a user, I want clear colors that tell about the transaction situation.
MNTR-11	As a user, I want to be able to use any device to check the monitor.
MNTR-12	As a admin, I want only authorized users to view appropriate content.
MNTR-13	As a sentry [vakt], I want to be alerted when something is not OK.
MNTR-14	Tasks/user stories for the backend.
MNTR-15	As a user, I want real-time updates.
MNTR-16	As a user, I want to see daily/weekly/etc metrics (avg/max throughput/latency/...) /for customers
MNTR-17	As an Admin I want secure information exchange between services.
MNTR-18	As an Admin I want safety guards against unwanted incidents.
MNTR-19	As a developer, I want a mock data source, so that I can start on the frontend

FIGUR 76: UTSNITT FRA JIRA. VISER BRUKERHISTORIER OG OPPGAVER SOM BLE LAGT TIL VED PROSJEKTSTART.

Sprint 1

Første oppstart med utviklingen var hos oppdragsgiver den 24 Januar. Dette var den eneste iterasjonen som gikk over tre uker. De første dagene fikk vi en presentasjon av systemene til PayEx. Vi fikk en kort introduksjon i hvordan transaksjonsmonitoren virket og hvilke standarder som ble brukt. Resten av tiden gikk på å sette seg inn i utviklingsmiljøet deres. Vi satte opp blant annet Outlook-konto, Git-miljøet, standardene satt av oppdragsgiver for kode og konfigurerte prosjektstyringsverktøyet Jira. Deretter gikk vi løs på planleggingen av vår første sprint.

Planlegging

Vi hadde et planleggingsmøte 29 Januar med sprintavslutning 16 februar. I planleggingsmøte ble det satt oppgaver som skulle utføres for denne sprinten og historie poeng (eng: story points) til hver av oppgavene. Poengene er representert i antall timer som estimerer arbeidstiden for hver funksjon.

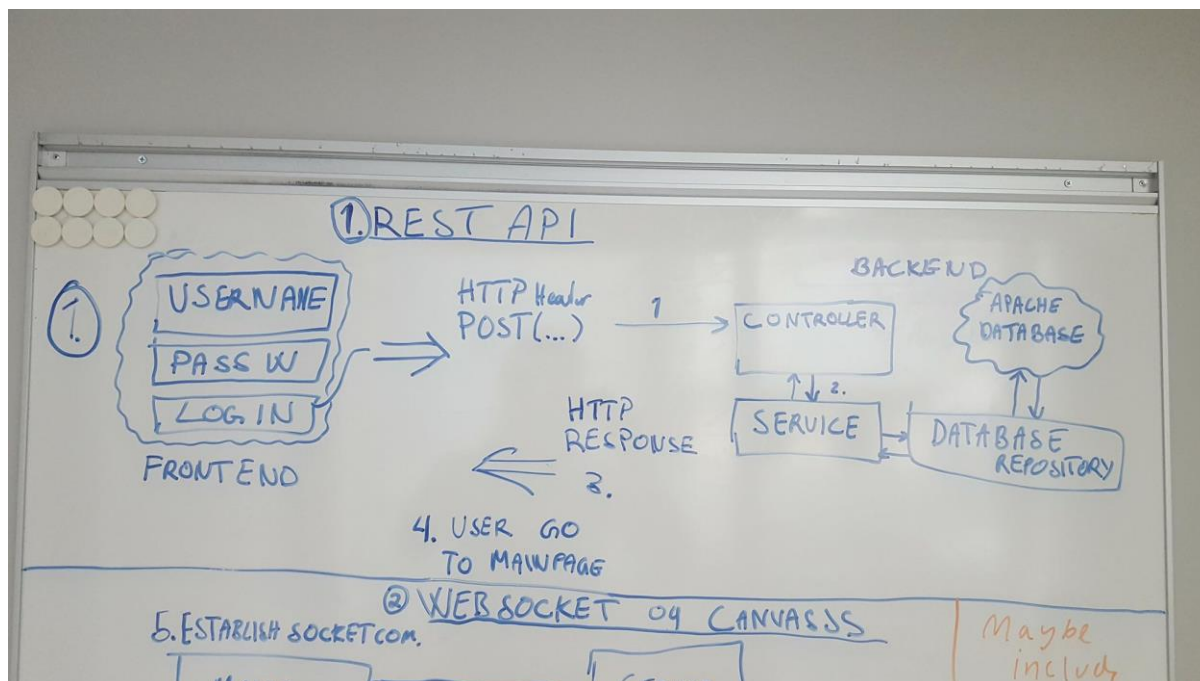
Key	Summary	Issue Type	Priority	Status	Story Points (46)
MNTR-19	As a developer, I want a mock data source, so that I can start on the frontend	Story	High	DONE	6
MNTR-20	As a user, I want to log into the system	Story	High	DONE	24
MNTR-21	Configure Spring Security with dummy users	Task	High	DONE	-
MNTR-22	Easy socket communication	Task	Low	DONE	-
MNTR-24	Experiment with various graph drawing libraries	Task	High	DONE	-
MNTR-25	As a user, I want see a simple dynamic graph, so that I can get a overview	Story	High	DONE	16
MNTR-26	Test and logging	Task	High	DONE	-

FIGUR 77: UTSNITT FOR JIRA. VISER OPPGAVENE SOM SKULLE BLI FULLFØRT I LØPE AV

Vi ble enige om å implementere grunnleggende funksjoner og prøve ut forskjellige typer graf biblioteker som kan brukes til å representere transaksjonsdata.

Hovedfokuset var først å implementere innlogging samt konfigurere Spring Security slik at den tillot utsending av data. Deretter tok vi for oss opptegningen av dataen.

Vi la planer for hvordan oppgavene kunne løses på best mulig måte. Whiteboard ble brukt aktivt for å diskutere hvordan en mulig løsning kunne se ut.



FIGUR 78: BILDE AV TAVLEN. HER DISKUTERTE VI HVORDAN APPLIKASJONEN KAN SENDE OG MOTTA FORESPØRSEL OG RESPONS.

Utførelse

Vi merket at ting tok lengre tid enn forventet når vi skulle sette opp Java Spring med Spring Security i backend. Problemet var Spring Security som automatisk implementerte filtre som ikke lot oss å hente ut data. Etter prøving og feiling fant vi ut hvilke filtre som måtte endres. Løsningen ble å overskrive filtrene slik at de godtok JSON-formatert data. Deretter gikk vi videre med å sende ut data i JSON format.

Vi fikk tildelt to virtuelle maskiner fra Oslo Metropolitan University for å teste applikasjonen. Det ble installert Java, Maven og Nginx webserver. Webserveren sendte trafikken videre til en service som kjørte på samme maskin for å autentisere og motta transaksjonsdata.

Resultat

Gruppen holdt sprint demo ved avslutningen der vi viste frem inkrementet som hadde blitt utviklet. Vi hadde kommet i mål med de planlagte oppgavene, men vi fikk ikke tid til å prøve ut forskjellige grafbiblioteker. En konsekvens ble at vi bestemte oss for å implementere D2B biblioteket for opptegning av grafene. Det skulle vise seg at D2B brukte mye ressurser, noe som førte til at applikasjonen hang seg i nettleseren. Vi diskuterte da med produkteier der det ble vedtatt at vi kunne komme med forslag til løsninger ved neste sprint planlegging.

Sprint 2

Denne iterasjonen hadde fokus på å lande på et bestemt grafbibliotek. Det ble derfor utført tester på forskjellige grafbiblioteker. Vi fokuserte også på å implementere et overordnet design for applikasjonen, der vi brukte rammeverket Bootstrap. Håndtering av innlogging med JSON Web Tokens med tilhørende tilbakemeldinger ble også implementert.

Planlegging

Planleggingsmøte ble holdt mandag den 19 februar. Sprintavslutningen ble satt til 2 mars. Under møtet diskuterte vi løsninger til problemet som gjorde at applikasjonen hang seg i nettleser. Vi kom fram til å prøve forskjellige graf biblioteker som krevde mindre ressurser enn D2B. Det ble også satt fokus på å få opp token-autentisering; ved pålogging skal det bli sendt ut et token som skal lagres i frontend. Dette tokenet skulle være gyldig i 15 minutter, for og så bli fornyet.

Key	Summary	Issue Type	Priority	Status	Story Points (48)
MNTR-28	Cleanup backend code	Task	High	DONE	-
MNTR-29	Stabilize backend (branch)	Task	High	DONE	-
MNTR-30	Write log information in backend	Task	Medium	DONE	-
MNTR-32	As a user, I want a conventional user interface	Story	High	DONE	16
MNTR-37	Research graph libraries.	Task	Critical	DONE	-
MNTR-38	As a user I want my login-session to be renewed so I dont need to login every 15 minutes.	Story	High	DONE	32
MNTR-50 *	As a User I want to see status of transactions using bar-charts to see if everything works	Story	High	DONE	-
MNTR-51 *	Clean up the frontend code, since we decided to use D2B	Task	High	DONE	-

FIGUR 79: UTSNITT FOR JIRA. VISER OPPGAVENE SOM SKULLE BLI FULLFØRT I LØPE AV SPRINT 2

Utførelse

Ved utførelsen av sprinten jobbet to av gruppemedlemmene med å teste ut forskjellige grafbiblioteker. De to andre jobbet med utsending av token og implementasjon av Bootstrap rammeverket for å designe webapplikasjonen.

Testing av grafbibliotekene skjedde ved at vi målte hvor lang tid hvert grafbibliotek brukte på å tegne opp et bestemt antall dummy-transaksjoner. Implementasjon av selve Bootstrap rammeverket gikk greit, men utsendingen av token bydde på noen problemer. Autentiseringsfilteret i Spring Security hadde ikke støtte for JSON Web Token og vi måtte derfor overskrive eksisterende autentiserings- og autoriseringsfiltre. Utførelsen av dette tok litt lengre tid enn vi så for oss.

Resultat

Funksjonene som ble implementert ble presentert på sprintavslutningen den 02 Mars. Vi viste oppdragsgiver resultatene fra stresstestene på de forskjellige grafbibliotekene, se vedlegg [Vedlegg C: Kravspesifikasjon](#). Ut ifra resultatene tok produkteier en beslutning om å videreutvikle D2B med fokus på å optimalisere koden og finne kilden til at D2B fikk nettleseren til å henge. Token-utsendelsen var godt gjennomført, men fornyelsen av token inneholdt noen implementasjonsfeil. Vi presenterte også for oppdragsgiver hvordan feilmeldinger ved innlogging ble vist og håndtert. Her ville arbeidsgiveren ha en standardisert utveksling på feilmeldinger. Dette ble da et tema for neste planleggingsmøte.

Sprint 3

Hovedfokuset i denne iterasjonen ble implementeringen av de forskjellige grafene som stolpediagrammer og linjediagrammer for presentasjon av transaksjonsbildet. Vi fokuserte også på fornyelse av token samt implementere et eksternt bibliotek som kunne håndtere feilmeldinger på et universelt format.

Planlegging

Planleggingsmøte var 05 Mars med avtalt sprintavslutning 19 Mars. Under møte diskuterte vi forskjellige måter å representere transaksjonsbildet på. Oppdragsgiver ønsket et overordnet Dashboard med status over alle kundegruppene og totalt antall transaksjoner som ble utført. Vi diskuterte også å ta i bruk Zalando biblioteket for å representere feilmeldinger i JSON-formatet. Til slutt var det et fokus på å få skrevet testing til backend i form av integrasjon- og funksjonstesting.

Key	Summary	Issue Type	Priority	Status	Story Points (80)
MNTR-7	As a user, I want to see a brief summary so that I can know whether everything is ok or not with a quick glance.	Story	High	DONE	16
MNTR-27	As a user, I want proper graphs that shows the transaction flow	Story	High	DONE	24
MNTR-35	As a user, I want clear and understandable error messages	Story	Medium	DONE	16
MNTR-53	As a user, I want to stay logged in when I have a valid token	Story	High	DONE	8
MNTR-65	Upgrade dummy data on the backend and frontend	Task	High	DONE	-
MNTR-77	As a developer, I want a central data/state store, for uniform access to data and to avoid duplicating functionality	Story	High	DONE	16

FIGUR 80: UTSNITT FOR JIRA. VISER OPPGAVENE SOM SKULLE BLI FULLFØRT I LØPE AV SPRINT 3

Utførelse

Under prosessen med å utvide implementasjon av grafbiblioteket oppdaget vi at fanen i nettleseren mistet responsivitet på grunn av den store ressursbruken med å oppdatere flere grafer hvert sekund. Problemet oppstår bare når man bytter aktivt vindu eller fane. Nettleseren får fanen til "å sove" når det ikke er aktivt, for å så vekke den når du kommer tilbake. Problemet førte til at vi innførte en ny oppgave i produktkøen. Dette ble videre diskutert mer på avslutningsdemo. Vi fortsatte arbeidet videre med å designe Dashboard, samt lese om hvordan tester i Spring skulle bli skrevet.

Resultat

Vi presenterte først Dashboard, de forskjellige visualiseringene og til slutt om problemet som oppstod når man gikk ut av fanen. Gjennom diskusjoner var det en idé om å prøve ut rammeverket D3 for å løse problemet. Ulempen med D3 er at de forskjellige grafene må skrives fra bunn av; det grafiske, oppteigningen, oppdateringen og informasjonsvisningen må skrives av utviklerne. Vi ble så enige om å ha samme type design, men implementere dette i D3. På grunn av problemet fikk vi ikke tid til alle oppgavene som skulle gjøres i sprinten. Vi fikk ikke tid til å skrive tester. Det ble heller ikke lagd en struktur for komponentene i Vue. Dette ble tatt med videre til neste sprint.

Issues Not Completed			
Key	Summary	Issue Type	Priority
MNTR-52	As a developer I want tests so I can refactor with confidence.	Story	High
MNTR-60	Have a structure for the components in Vue	Task	High

FIGUR 81: UTSNITT FRA JIRA. VISER OPPGAVENE SOM IKKE BLE FULLFØRT I LØPET AV SPRINT 3

Sprint 4

Denne iterasjonen hadde fokus på å skrive tester, rydde opp i koden og utvide feilmeldings-visningen. Når brukeren mistet kontakt med servicen, skulle han få vite hvor lenge kontakten hadde vært borte. Vi satt oss også inn i hvordan tester for frontend i form av WebDriverIO og Selenium skulle gjøres.

Planlegging

Planleggingsmøte var 19 Mars og avslutningen ble avtalt til å være 02 April. Vi diskuterte testing, opprydning, implementasjon av D3 biblioteket og utvidelse av feilmeldingsvisningen. Det ble også diskutert om å koble oss til en reell generert datastrøm av transaksjoner, og hvordan vi ville ha denne gjengitt. Vi ville gjerne at dataen skulle være så likt som mulig vår dummy data, og vi ble fortalt at dette ikke skulle være et problem.

Key	Summary	Issue Type	Priority
MNTR-52	As a developer I want tests so I can refactor with confidence.	Story	High
MNTR-62	Clean up frontend, delete garbage code	Task	High
MNTR-78	Frontend hangs when webpack service and backend killed	Bug	Low
MNTR-81	Finish boxplot and volume graphs.	Task	High
MNTR-83	Status indicator for reconnect to the backend.	Task	High
MNTR-84	Checkout tests for frontend.	Task	High

FIGUR 82: UTSNITT FOR JIRA. VISER OPPGAVENE SOM SKULLE BLI FULLFØRT I LØPE AV SPRINT 4

Utførelse

Under utførelsen støttet vi ikke på noen problemer, men det å implementere D3 biblioteket og skrive alle de forskjellige elementene tok lengre tid enn det som var estimert. Ved implementasjon av komponenten som viste feilmeldinger brukte vi rammeverket “Feather-Icons” som inneholder et sett med ikoner for å lettere visualisere det vi ønsker å vise.

Resultat

Under demovisningen viste vi frem det vi hadde gjort, spesielt implementasjonen av D3 biblioteket, fordi det var en prioritert oppgave for denne sprinten. Siden grafene ble skrevet fra bunn av, åpnet dette opp for personlige preferanser over hvordan dataene skulle vises. Vi kunne vise statistikk data som median, høyest og lavest transaksjonsprosessering tid og avviks tid.

Ved slutten av demovisningen viste vi en demo over feilmeldingsvisningen og testene som vi hadde laget. Oppdragsgiver var fornøyd med gjennomføringen av testene siden disse ville gjøre det lettere å videreutvikle applikasjonen i en senere tid.

Sprint 5

Denne iterasjonen varte i to uker. Hovedfokus for denne iterasjonen var å få implementert få ferdig grafene i D3 og i tillegg skrive nye tester både for backend og frontend.

Planlegging

Planleggingsmøte var 03 April med sprintavslutning 13 April. Vi diskuterte forskjellige måter å vise transaksjonsbilde på og hvilke farger PayEx ønsket å vise dataene med. Det ble også diskutert om essensielle tester som var nødvendig for produktet. Da kom det fram at oppdragsgiver ønsket tester på frontend, blant annet navigasjon, autentisering, token-utsending og datastrøm på backend.

Key	Summary	Issue Type	Priority
MNTR-10	As a user, I want clear colors that tell about the transaction situation.	Story	Low
MNTR-52	As a developer I want tests so I can refactor with confidence.	Story	High
MNTR-81	Finish boxplot and volume graphs.	Task	High
MNTR-83	Status indicator for reconnect to the backend.	Task	High

FIGUR 83: UTSNITT FOR JIRA. VISER OPPGAVENE SOM SKULLE BLI FULLFØRT I LØPE AV SPRINT 5

Utførelse

Mens en av gruppemedlemmene jobbet med å implementere grafene ferdig i D3, jobbet resten med design og testing.

Resultat

For denne sprinten kom vi i mål med alle oppgavene. Under demovising diskuterte vi om flere ønskelige tester, spesielt på formatering av transaksjonsdata. Vi viste fram D3 implementasjonen og ble fortalt om flere ønskelige preferanser og utseendemessig kosmetikk som kunne forbedres. Som siste framvisning av inkrementet tok vi en demo på responsiviteten til applikasjonen og det viste seg at den trengte finpusning. Til slutt ble vi enige om å komme med forslag til oppgaver for neste planleggingsmøte.

Sprint 6

Denne iterasjonen varte i 2 uker og hovedfokuset var på å få applikasjonen responsiv. Siden vi nærmet oss prosjektslutt måtte vi begynne å fokusere på å rydde opp i koden og finpusse detaljer rundt applikasjonen. Vi måtte også sette av tid til å skrive første utkast til sluttrapporten.

Planlegging

Planleggingsmøte var 13 April og avslutningsdemoen ble satt til 30 April. Under møtet diskuterte vi om å tilrettelegge webapplikasjonen forskjellige enheter som mobiltelefoner, nettbrett og store monitorer.

Oppdragsgiver hadde ønske om å gjøre applikasjonen dynamisk og støtte flest enheter som mulig. For å implementere dette sa vi ifra at skaleringen i D3 biblioteket måtte bli skrevet selv, noe som kunne ta tid. Dermed ble denne oppgaven høyest prioritert i sprint backloggen. Til slutt bestemte vi oss for å legge til en "utkastelse" med en melding, hvis innlogging sesjonen til bruker blir ugyldig.

Key	Summary	Issue Type	Priority
MNTR-11	As a user, I want to be able to use any device to check the monitor.	Story	High
MNTR-85	Status bar icon dissepars on refresh	Bug	High
MNTR-86	As a user I want a redirect to the login page with a message, when token cannot be refreshed.	Story	High
MNTR-88	replace system out with logger	Task	High

FIGUR 84: UTSNITT FOR JIRA. VISER OPPGAVENE SOM SKULLE BLI FULLFØRT I LØPE AV SPRINT 6

Utførelse

Å gjøre D3 responsivt var som forventet en tidkrevende oppgave. I dokumentasjonen til D3 biblioteket var det beskrevet at D3 deaktiverte animasjonseffekter når siden ikke hadde fokus i nettleseren. Dette for å hindre at den skulle henge seg. Under utviklingen viste det seg at animasjonseffektene likevel ikke ble deaktivert, men satt i pause. Det vil si at hvert bilde i animasjonen ble lag til en kø. Når nettsiden fikk fokus igjen, ville den tegne opp alle bildene fra animasjonen, og derav hang nettleseren seg. Løsningen ble å skrive en algoritme som kunne ta høyde for dette. Programvare feilen med status bar ikonet ble enkelt fikset og utlogging ved ugyldig token ble implementert.

Resultat

Under demovisningen viste vi fram det vi hadde fått til. Tilbakemeldingene fra oppdragsgiver var positive, og ga uttrykk for at vi nærmet oss et sluttprodukt. Videre ble det diskutert om at det var ønskelig å ha en egen side for hver kundegruppe. Denne siden skulle vise statusen og informasjonen rundt transaksjonsgruppene. Møtet avsluttet med at oppdragsgiver informerte om utrulling av API-et som vår webapplikasjon skal koble seg mot var i gang.

Sprint 7

Fokuset for denne sprinten ble å legge til en egen side for de forskjellige kundegruppene med tilhørende funksjonalitet. Dette ble den siste sprinten i utviklingsprosessen hvor vi la til ny funksjonalitet.

Planlegging

Planleggingsmøte ble holdt den 30 April med avslutningsdemo 11 Mai. Under møte diskuterte vi hvordan data fra oppdragsgivers API skulle se ut og hva slags funksjonalitet de ønsket på kundegruppen sin side. Det viktigste med siden var at man fikk nyttig og detaljert informasjon om kundegruppen. Til slutt ble vi enige om å ferdigstille sluttproduktet ved å finpusse, kommentere, og hvis API-et fra PayEx som skal bli brukt for å hente ut transaksjonsdata var ferdig, skulle vi integrere dette med applikasjonen.

Key	Summary	Issue Type	Priority
MNTR-8	As a user, I want to see detailed information about customer so that I can get useful information.	Story	High
MNTR-13	As a sentry [vakt], I want to be alerted when something is not OK.	Story	High
MNTR-16	As a user, I want to see daily/weekly/etc metrics (avg/max throughput/latency/...) /for customers	Story	High
MNTR-89	As a user, i want a responsive design	Story	High
MNTR-90	As a user, I want a overview over hosts so that I know when one of them goes down	Story	High
MNTR-91	Create new issues to backlog	Task	High

FIGUR 85: UTSNITT FOR JIRA. VISER OPPGAVENE SOM SKULLE BLI FULLFØRT I LØPE AV SPRINT 7

Utførelse

På grunn av en del inneklemt dager, ønsker oppdragsgiver at avslutningsdemoen for sprinten ble flyttet til fredag den 18 mai. Videre bestemte vi oss for å gjøre ferdig de funksjonalitetene som var halvveis implementert. Ny funksjonalitet ville kun bli lagt til dersom det gjenstod tid til det.

Resultat

Applikasjonen ble ferdigstilt med den nødvendige funksjonaliteten. Vi fikk lagt til en side som viser detaljert informasjon om en kundes betalingstransaksjoner, applikasjonen fungerer på forskjellige enheter, og vi kan nå se informasjon om innløser. På grunn av strenge krav stilt til PayEx sine API-er, fikk de ikke ferdigstilt det API-et vi skulle koble oss mot. Dermed fikk vi ikke muligheten til å utvikle et varslingsystem.

Utfordringer i utviklingsfasen

Siden ingen av gruppemedlemmene hadde erfaring i Java Spring var den største utfordringen i utviklingsprosessen den bratte læringskurven til rammeverket. Vi fant tidlig ut at vi måtte implementere og overskrive en del funksjonaliteter for å få støtte til JSON Web Token. Dette førte til at vi måtte sette oss inn i oppbyggingen og strukturen til et rammeverk som vi ikke hadde rørt før. Vi håndterte utfordringen ved å lese artikler, offisiell dokumentasjon og en bok om Java Spring som vi fikk låne fra oppdragsgiver. Vi prøvde og feilet med en del implementasjoner av filtrene i begynnelsen, men kom til slutt i mål ved å lese oss opp på dokumentasjon, veiledninger, og samarbeid oss imellom.

En annen utfordring var at grafbiblioteket D2B ikke var optimalisert for en oppdateringsfrekvens av det nivået vi behøvde. Problemet var at oppdateringer av grafen, spesielt hvis de skjedde når fanen var i bakgrunnen, ikke slettet gamle grafelementer. Dette førte til økende ressursbruk og til slutt ble fanen med applikasjonen i nettleseren uresponsiv. Vi prøvde først å løse problemet ved å teste ut en mengde forskjellige grafbiblioteker. Det ble testet ut tilsammen elleve grafbiblioteker der ingen av de var optimalisert for sanntidstegning. I tillegg dekket ikke bibliotekene funksjonaliteten som oppdragsgiveren ønsket seg. Vi satt i en liten knipe, men et av gruppemedlemmene fikk en ide; istedenfor for å bruke de ferdiglagde grafene, kunne vi heller skrive vår egen implementasjon med D3-biblioteket. En implementasjon som tok høyde for sanntidstegning og i tillegg hadde de funksjonalitetene som oppdragsgiver ønsket.

Oppsummering

Utførelsen av arbeidsmetodikken Scrum som ble benyttet under utviklingsprosessen var vårt første møte med metodikken i praksis. Selv om vi har fått teoretisk kunnskap om Scrum i utdanningen, var det å gjennomføre metodikken en erfaring som tok oss utenfor vår komfortsone. Vi har gjennom denne arbeidsprosessen lært oss at sprintene som utføres vanligvis ikke går som planlagt. Enten fordi man feilestimerer oppgaver eller at det skjer endringer underveis i kravspesifikasjonen som man må ta hensyn til. Det er ikke realistisk at alt som vi planla å utføre ble gjennomført i arbeidsperioden. Vi lærte dermed å ta beslutninger på strak hånd og å gjøre justeringer underveis for å gjennomføre prosjektet så godt som mulig.

Hvis vi fikk muligheten til å utføre prosjektet på nytt hadde vi brukt mer tid på rammeverket Spring før vi startet med arbeidsprosessen. Som tidligere nevnt brukte vi en del tid på å bli kjent med rammeverket underveis i arbeidsprosessen, fordi ingen av oss hadde tidligere erfaring med det. Spesielt ville vi blitt kjent med den underliggende funksjonaliteten i Spring når det gjelder Spring Security Filter Chain, for å manuelt endre på filtrene. Dette kunne vi ha spart tid på hvis vi hadde forberedt oss tidlig i arbeidsprosessen ved å studere Spring og Spring Security.

Vedlegg C: Kravspesifikasjon

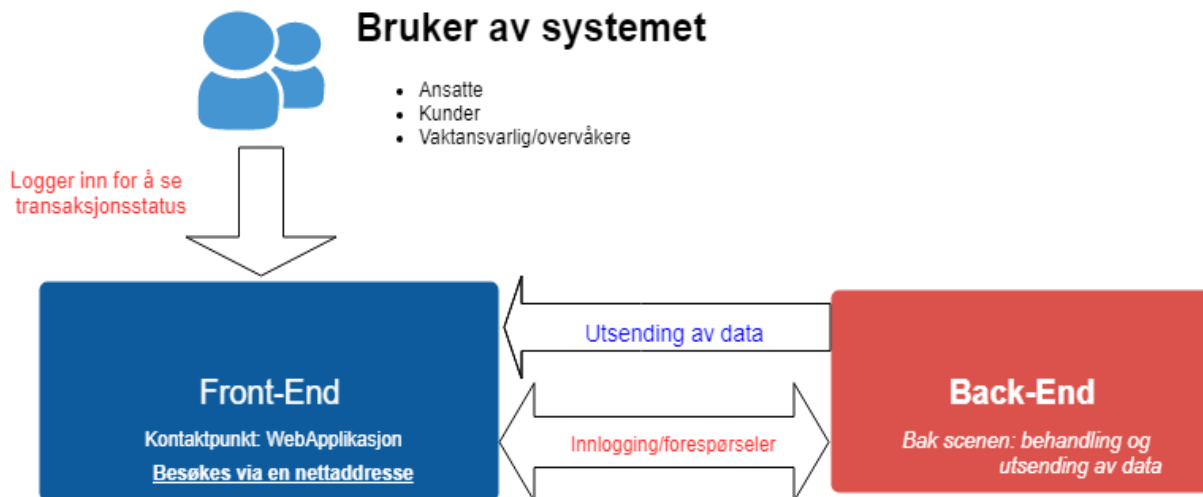
I denne delen av rapporten tar vi for oss formålet med prosjektet, inklusivt rammer og krav som ble stilt til sluttproduktet. Hensikten er å gi interessenter en innføring i hvordan produktet ble bygget opp i tillegg til hvorfor valgte rammeverk og teknologier ble tatt i bruk. Kravspesifikasjonen er utarbeidet etter [Vedlegg I: Forslag/ønsker fra PayEx](#) og planleggingsmøtene.

Kort systembeskrivelse

PayEx sine ambisjoner for applikasjonen er et verktøy som lar dem tidlig oppdage stopp i transaksjonsflyten ved å fange opp anomalier i transaksjonene før feilen innmeldes av kunden. Ved feil skal systemet varsle de ansvarlige i form av mail eller som en notifikasjon på Slack. Applikasjonen må også kunne presentere enkel statistikk av transaksjonsbildet for de ulike kundene og være tilgjengelig for ansatte utenfor PayEx sine lokaler.

Systemet skal bestå av to deler: en nettside (frontend) og en service (backend) som gir nettsiden de nødvendige dataene. Nettsiden skal vise informasjon om betalingstransaksjonene. Servicen skal utføre statistiske beregninger, håndtere utsending av data, sende varsler ved avvik i tillegg til å kunne autentisere og autorisere brukere. Det er også viktig at applikasjonen er utformet på en slik måte at den enkelt lar seg spinne opp i nye instanser dersom belastningen på systemet er stort.

- Webapplikasjonen skal:
 - Vise opplysninger om transaksjonsbildet.
 - Hvor mange transaksjoner som har blitt utført i dag.
 - Hvor lang tid det tar å utføre en transaksjon.
 - Hvilke kundegrupper som ikke har mottatt transaksjoner på en stund.
 - Hvilke kundegrupper omfattes av treg transaksjonsprosessering.
 - Vise stopp i transaksjonsbilde og hvor lenge det har vært stopp.
- Servicen skal:
 - Behandle autentiseringsforespørsler.
 - Sjekke at brukernavn og passord er gyldig.
 - Hvor lenge en bruker kan være logget inn.
 - Utstede "token" som forteller at innloggingsøkten er gyldig.
- Behandle autoriseringsforespørsler.
 - Gi brukeren tilgang til det de har tilgang til.
 - Utsending av data.
 - Sørge for at data sendes ut jevnlig til tilkoblede brukere.
- Data sendes ut i et universelt format.
- Varsle vakter ved avvik
 - Sende ut e-mail / SMS.



FIGUR 86: VISER OVERORDNET HVORDAN FRONTEND OG BACKEND KOMMUNISERER.

Systemrammer

Systemrammene er krav og tiltak satt i bruk for å forhindre at arbeidet med applikasjonen fører til tap, ødeleggelse, misbruk av data og for å sikre brukervennlighet.

Sikkerhet

For å forhindre tap og misbruk av data har vi en rekke tiltak. Hver programutvikler har en brukerkonto som er sikret med brukernavn og sterkt passord, der passordkriteriene er bestemt i form av retningslinjer satt hos oppdragsgiveren. Passordet må skiftes hver tredje måned. For å få tilgang til interne systemer hos oppdragsgiveren blir det tatt i bruk et RSA-genererings brikke, der vi logger inn med tilsvarende nøkkelkode som har blitt generert. I det interne systemet har vi tilgang til databasen, UI/UX design retningslinjer og mye mer.

Det blir brukt versjonskontroll for å forhindre tap av kode. Vi har egen repositories hos organisasjonen PayEx, der koden blir kjørt igjennom TeamCity som er PayEx sitt kontrollsystem for kode. Der blir det sjekket om koden følger en viss standard og at det ikke ligger noe ondsinnet i koden som kan forårsake problemer.

Hvert gruppelem har skrevet under en taushetskontrakt, der vi skrev under på at vi ikke skal dele forretningsmessige, personlige og sensitive opplysninger. Dette er for å forhindre at kritisk informasjon skal bli lekket ut.

Sikkerhet i applikasjonen

Det som utvikles skal være så sikkert som mulig. Informasjonen som er tilgjengelig i backend skal være lukket og en utenforstående skal ikke kunne vite hva slags data som finnes der. De kan bare nå informasjonen ved å ha en gyldig innlogging sesjon. Informasjonen som utveksles skal være kryptert og sikkert.

Det skal være så lite informasjon som mulig om dataene som mottas i frontend. Dette er for å forhindre at en bruker kan se igjennom kildekoden på nettsiden og få sensitive opplysninger om betalingstransaksjonene.

Ved pålogging mottas det ett token som forteller om gyldigheten til sesjonen, samt hvilke rettigheter brukeren har. Hvis brukeren er logget inn blir tokenet fornyet hvert 15 minutt, hvis ikke må man logge inn på nytt.

Roller

Brukere har forskjellige roller som definerer hva slags informasjon brukerne kan se. Noen brukere vil kunne se hele transaksjons bilde mens andre kan bare se informasjon om sin tilhørende organisasjon. Det skal legges til funksjonalitet med tanke på hvilken rolle man får tildelt, men dette er noe oppdragsgiver kommer til å videreutvikle på grunn av sikkerhetsmessige grunner.

Effektivitet

Applikasjonen skal være effektiv nok til å se transaksjonsbildet i sanntid. Det blir utført tusenvis av transaksjoner daglig og hver transaksjon skal tegnes og vises på applikasjonen. Dermed er det viktig med effektivitet for at applikasjonen ikke er ressurskrevende. Dette er også viktig for at applikasjonen skal kunne kjøres på andre enheter som for eksempel mobiltelefoner, slik at det ikke har stor effekt på batteriet og ytelsen til telefonen.

Skalerbarhet og fremtidig utvikling

Applikasjonen skal være skalerbar. I utgangspunktet blir applikasjonen utviklet som en webapplikasjon, men det er satt en standard om at informasjonen som utveksles skal være på et universell standard; JSON objekter. Dette åpner opp for å koble til flere enheter på backend for å få tilgang til nødvendig informasjon.

Hensikten er at det skal være lettere å videreutvikle, legge til funksjoner og at implementasjonen skal være dokumentert.

Bruk og brukervennlighet

Designet til applikasjonen skal være brukervennlig og representert i bruk av PayEx sine farger. Brukeren til applikasjonen skal kunne lett navigere og få nødvendig informasjon som er formidlet på lettest mulig måte. Informasjonen kan være formidlet i form av grafer, vises som rådata eller ha forskjellige typer barer som for eksempel en status bar. Applikasjonen skal også kunne brukes på flere enheter og gi brukeren muligheten for å se transaksjons bilde etter sin preferanse.

Funksjonelle krav

Funksjonelle krav kan være en tjeneste et system tilbyr. Oppdragsgiveren har gitt oss grunnleggende funksjonelle krav som kommer til å bli brukt som grunnmur for applikasjonen og hovedfunksjonaliteten som den tilbyr. Kravene som er satt av oppdragsgiver nedenfor, er representert i form av brukerhistorier:

Som	Ønsker jeg	Slik at
Bruker	Å ha en webapplikasjon	jeg kan nå tjenesten fra hvor som helst.
Bruker	Å ha et overordnet blick over transaksjons bilde	Jeg kan se at alt er i orden eller ikke ved et kort blick.
Bruker	Å ha grafisk fremstilling av data.	dataen blir på lettest mulig måte presentert til meg.
Bruker	Å ha forskjellige typer fremstillinger av dataen	jeg kan velge hvilken fremstilling jeg vil bruke.
Bruker	Å ha en responsiv web applikasjon	jeg kan bruke applikasjonen på forskjellige enheter.
Bruker	ha god design	løsningen kan brukes.

Som	Ønsker jeg	Slik at
Utvikler	Å ha en separert frontend fra backend	muligheten åpner seg for å koble til andre typer klienter.
Utvikler	At backend er avhengig så lite som mulig fra andre tjenester	det kan startes flere instanser av backend.
Utvikler	at informasjonen utveksles mellom sikre kanaler	informasjonen som blir utvekslet er sikker som mulig.
Utvikler	at transaksjonsdata sendes og vises i real-time	slik at man kan fortløpende se trender.

Som	Ønsker jeg	Slik at
Admin	Frontend må kunne vise ulik data basert på rettighetsnivå.	Brukere som ikke har lov til å se dataen burde ikke se den.
Admin	Ønsker jeg at andre systemer ikke blir påvirket	Det ikke blir endret noe på de systemene

Ikke-funksjonelle krav

Som	Ønsker jeg	Slik at
Bruker	Ønsker jeg å få status overblikk ved å se på skjermen i mindre enn 5 sekunder.	Jeg kjapt får vite at alt er i orden.
Utvikler	at innlogging sesjonen er gyldig i 15 minutter.	Det er sikkert at de som ikke kan være logget inn lenge blir kastet ut. De som har lov får en ny sesjon.
Bruker	at dataen blir tegnet opp på frontend en gang i sekundet.	slik at vi får med oss endringer så fort som mulig.

Krav til systemkonstruksjon

Det ble utformet krav i begynnelsen av prosjektstart. Gruppen og oppdragsgiver gikk sammen og bestemte de generelle kravene for systemet.

- Backend skal utvikles i rammeverket Java Spring
- Det skal brukes JSON Web Token ved innlogging som forteller om innloggings sesjonens gyldighet.
- Frontend skal bli utviklet i HTML, CSS og JavaScript (Vue.js).
- Det skal bli brukt et tredjeparts bibliotek for å grafisk presentere transaksjonsdata.
- Det skal bli brukt PayEx sine kodestandarder som setter standard på hvordan koden er bygd opp.

Krav til dokumentasjon

Det er ikke satt noe skriftlig krav til dokumentasjon, men vi har blitt minnet på om å ha kommentarer og beskrivelser av koden underveis. I tillegg har vi blitt minnet om at det er viktig å dokumentere sprintprosessen. Ved bruk av prosjektstyringsverktøyet Jira dokumenterer vi hvilke oppgaver som har blitt utført av hvem og beskrivelse av oppgavene.

Stresstest

Under finnes resultatet gjort fra stresstestingen til de forskjellige graf bibliotekene. Testen foregikk på to plattformer:

- Hele systemet ble kjørt på en PC og testet
- Hele systemet kjørte på en ekstern nettside, som ble så besøkt via en url.

Kolonne	Beskrivelse
Library	Hvilket bibliotek som ble testet og implementert.
Pull rate	Hvor ofte dataen ble tegnet opp.
Total elements drawn	Hvor mange transaksjoner som ble tegnet opp talt.
Duration	Hvor mye tid det tok å tegne opp. Unresponsive: betyr at nettleseren sluttet å svare på grunn av for mye ressursbruk.
Memory	Hvor mye minne som ble brukt

Opptegning på lokal pc

Library	Pull rate	Total elements drawn	Duration	Memory	PC
D2B	1 ms	1000	0,65s	377MB	KENT
Highchart	1 ms	1000	UNRESPONSIV E	?	KENT
C3	1 ms	1000	17,263s	491MB	KENT
Frappe-charts	1 ms	1000	11,06 s	1100MB	KENT
Chartjs	1 ms	1000	0,4 s	350MB	KENT
Echarts	1 ms	1000	UNRESPONSIV E	?	KENT
Chartist	1 ms	1000	2,1 s	382MB	KENT
Plotly	1 ms	1000	16 s	463MB	KENT
Google 1 (vuecharts)	1 ms	1000	UNRESPONSIV E	?	KENT
Google 2 (chartkick)	1 ms	1000	17.6 s	264MB	KENT
Google 3 (VanillaJS)	1 ms	1000	7,1 s	770MB	KENT

Library	Pull-rate	Total elements drawn	Duration	Memory	PC
D2B	1 ms	10000	7,5s	589MB	KENT
Highchart	1 ms	10000	UNRESPONSIVE	?	KENT
C3	1 ms	10000	UNRESPONSIVE	?	KENT
Frappe-charts	1 ms	10000	UNRESPONSIVE	?	KENT
Chartjs	1 ms	10000	11,55 s	285MB	KENT
Echarts	1 ms	10000	UNRESPONSIVE	?	KENT
Chartist	1 ms	10000	22s	285MB	KENT
Plotly	1 ms	10000	UNRESPONSIVE	?	KENT
Google 1(vuecharts)	1 ms	10000	UNRESPONSIVE	?	KENT
Google 2 (chartkick)	1 ms	10000	UNRESPONSIVE	?	KENT
Google 3(VanillaJS)	1 ms	10000	73 S	3200MB	KENT

Opptegning på ekstern maskin (Nginx webserver)

Library	Pull-rate	Elements drawn (per pull)	Memory / CPU	PC
D2B	500ms	10	200MB/ 35%	KENT
Highchart	500ms	10	178MB/ 95%	KENT
C3	500ms	10	222MB/ 55%	KENT
Frappe-charts	500ms	10	300MB / 80%	KENT
Chartjs	500ms	10	186MB / 57%	KENT
Echarts	500ms	10	400MB/ 90%	KENT
Chartist	500ms	10	145MB/ 6.2 %	KENT
Plotly	500ms	10	200MB / 60%	KENT
Google 1(vuecharts)	500ms	10	?	KENT

Google 2 (chartkick)	500ms	10	?	KENT
Google 3(VanillaJS)	500ms	10	221MB / 43%	KENT

Vedlegg D: Forklaring av brukte teknologier

Backend

Backend er den delen av tjenesten som tjener en direkte støtte ved å utføre nødvendige oppgaver for å få et helhetlig system. Backend ligger nærmere transaksjon dataen og kan ha evnen til å kommunisere med dataene som er lagret i en database. I vårt tilfelle skal backend håndtere blant annet autentisering, autorisering og utsending av data til klientene.

Spring

Spring er et rammeverk som er egnet for å møte behovet til store bedrift applikasjoner. En av hoved fordelene med Spring er at det er bygget opp modulært, det vil si at rammeverket er bygget opp i flere lag som tilbyr en spesiell funksjon. Rammeverket tilbyr blant annet disse modulene:

- Spring Core
 - Tilbyr muligheten til å håndtere avhengigheter ved bruk av komponenter og beans.
- Spring Web
 - Tilbyr utvikling rettet mot webapplikasjoner.
- Spring Context
 - Bygger på Spring Core, der du har muligheten for å automatisk merke endringer og gjøre endringer på tilknyttet komponenter og beans.

En slik arkitektur gjør det lett å ta i bruk nødvendige moduler og videreutvikle på disse. I applikasjonen bruker vi disse modulene men har videre implementert de for å best mulig passe behovet vårt.

Applikasjonen bruker Spring Security og Spring Boot som er rammeverker tilegnet for utvikling av sikkerhet og enkel oppstart av applikasjonen.

Spring Security er rammeverket som beskytter applikasjonen med omfattende bruker autorisering og autentisering. En stor fordel er at det er enkelt å utvide og tilpasse møtende krav. Rammeverket har flere egenskaper som blant annet:

- Omfattende og utvidbar støtte for både autorisering og autentisering.
- Beskyttelse for flere typer angrep mot datasystemet ved bruk av sikkerhetsmekanismer.

Spring Boot er designet slik at det skal være enkelt å starte opp en Spring applikasjon. Spring Boot er forhåndsinnstilt for hurtig oppstart av applikasjonen ved bruk av tredjeparts biblioteker.

JSON

JSON står for “JavaScript Object Notation” og er en tekstbasert standard for datautveksling. Vi bruker JSON som hoved standarden for utveksling av data i applikasjonen. Java Spring har muligheten til å automatisk konvertere data til JSON-format og omvendt. Når det kommer inn data til backend i JSON-format, kan vi direkte aksessere variablene og feltene uten å hardkode uthenting av data.

JSON Web Token

JSON Web Token (JWT) er en åpen bransjestandard som brukes til å representere de kravene som er satt mellom to parter, som for eksempel mellom en klient og en server. Når en klient skriver inn brukernavn og passord for å logge inn på serveren, vil serveren først validere om informasjonen er gyldig. Deretter vil serveren generere en token til brukeren som vil være brukerens identitet. Brukeren vil bruke dette tokenet til å få aksess til serveren og deres tjenester. Fordelen med JSON Web Tokens er at de følger en viss signatur som gjør det ekstra sikkert. Det vil si at en angriper kan ikke for eksempel misbruke og endre tokenet til en bruker fordi det er bare brukeren som har signaturen til tokenet.

JWT har generelt tre deler:

- Header
 - Forteller at dette er en JSON Web Token.
 - Forteller hva slags krypteringsalgoritme som er brukt
 - Vi bruker HS512 (HMAC-SHA256)
- Payload
 - Inneholder brukernavn
 - Inneholder når tokenet ble utgitt.
 - Denne delen kan da inneholde informasjon som forteller oss grunnleggende informasjon.
- Signature
 - Inneholder selveste tokenet, som er kalkulert i henhold til satt algoritme, brukernavn og hemmelig nøkkel.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4  
gRG9lIiwiaXNtb2NpYWwiOnRydWV9.  
4pcPyMD09o1PSyXnrXCjTwXyr4BsezDI1AVTmud2fU4
```

FIGUR 87: HER SER DERE ET EKSEMPEL PÅ JSON WEB TOKEN, DER HVER DEL ER ADSKILT MED PUNKTUM.

Problem

Problem er et tredjepartsbibliotek for feilhåndtering i webtjenester i Java og utveksler informasjon i JSON. Biblioteket tar seg av unntak når de blir kastet i backend og

sender tilbakemelding til brukeren angående feilmeldingen i JSON-formatet. Dette biblioteket ble valgt for å ha et felles grunnlag på hvordan feilmeldinger skulle bli håndtert og utveksles.

Apache Maven

Backend tar i bruk Apache Maven som er et verktøy som kan brukes i Java-prosjekter for å bygge systemet. Maven er basert på konseptet "Project Object Model" (POM), som er en XML-fil som inneholder konfigurasjonen til prosjektet, avhengigheter til eksterne moduler og komponenter, byggeprosessen, og strukturen på katalogen. Denne informasjon blir brukt av Maven for å bygge prosjektet.

Frontend

Frontend er den delen av applikasjonen som brukeren forholder seg til. Brukeren kan logge inn, se status og utføre handlinger for å få mer informasjon om transaksjonsbildet. Et eksempel er ved innlogging når en bruker trykker på innloggingsknappen. Det vil da bli sendt brukernavn og passord til backend som validerer informasjonen. Ut ifra resultatet vil brukeren enten få beskjed om mislykket innloggingsforsøk eller bli logget inn.

Frontend er primært utviklet i HTML, CSS og JavaScript. Det er brukt tredjeparts biblioteker for å gjøre utviklingsprosessen lettere. Bibliotekene vi har tatt i bruk er Vue.js, Bootstrap, Feather og D3.

HTML og CSS

HTML (Hyper Text Markup Language) er et standardisert markeringsspråk som definerer hvordan nettsiden er bygd opp. Man kan blant annet definere overskrifter, brødtekst, tabeller, knapper, skjemaer og mye mer til nettsiden.

HTML definerer bare elementene som vises. Det vil si at du kan ikke bestemme utseende på disse. Dette gjør du ved bruk av CSS (Cascading Style Sheets) som kan designe elementene etter dine egne ønsker.

JavaScript og Vue.js

JavaScript er et programmeringsspråk som lar deg implementere komplekse funksjoner til din webapplikasjon. JavaScript utfører beregninger, funksjoner og andre nødvendigheter for å få en fullstendig webapplikasjon. Et eksempel er håndteringen av mottatt data fra backend serveren.

Vue.js eller Vue er et enkelt og effektivt rammeverk utviklet i JavaScript for å bygge store og skalerbare webapplikasjoner. I motsetning til de fleste andre rammeverk, er Vue bygget opp fra bunnen av til å være enkelt å adoptere trinnvis. Det vil si at det er enkelt å benytte side om side med andre biblioteker eller i eksisterende prosjekter. Vue også veldig godt egnet til utvikling av SPA-er (Single Page Application) - websider som oppdaterer og bytter ut deler av seg selv i stedet for å laste ned en helt ny side fra serveren.

Bootstrap


Bootstrap er et rammeverk som tilbyr grafiske komponenter til utviklerne. Bootstrap er utviklet i HTML, CSS og JavaScript. Komponentene er utviklet for å gjøre det enklere for utviklerne å implementere grunn komponenter som for eksempel navigasjonsmeny, tabeller, popup-bokser og mye mer. Ved bruk av Bootstrap slipper du å kode funksjonaliteten fra bunnen av.

Vue-Bootstrap er et rammeverk som er egnet for Vue. Dette rammeverket er brukt noen steder i applikasjonen, men primært har det blitt brukt Bootstrap.

Feather

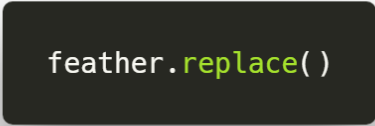
Feather er en samling av ikoner som er gratis og lett å bruke. Du kan velge mellom flere hundre ikoner ved å definere et attributt som heter data-feather, og gi denne navnet på ikonet som verdi. Et eksempel som viser et sirkel-ikon:

```
<!-- example icon -->
<i data-feather="circle"></i>
```



FIGUR 88: VISER HVORDAN IKONER BLIR LAGT TIL.

For å bytte ut dette elementet med det faktiske ikonet i SVG-format må man kjøre:



```
feather.replace()
```

FIGUR 89: INITIALISERING AV IKONER

På grunn av dette ble det opprettet en Vue-komponent for å ta seg av denne initialiseringen automatisk.

D3

D3 (Data Driven Documents) er et JavaScript rammeverk for å visualisere data ved bruk av HTML, CSS og SVG. D3 kan enkelt bruke JSON formatert data for å visualisere transaksjonsdata. D3 lar deg definere og visualisere data fra bunnen av. Dette gjør at man har kontroll over hva som tegnes opp og hvor ofte, og kan da ta høyde for ønsket oppdateringsfrekvens.

D3 tilbyr også grunnfunksjonalitet for opptegning som for eksempel opptegning av akser, linjegrafer også videre.

Vedlegg E: Oppsett av servermiljø

Applikasjonen tilbys over nettet, og er satt opp ved bruk av Nginx webserver, Java, og Maven. Serveren kjører Linux-distribusjonen Debian 4.14.13.

Java og Maven

```
sudo apt-get install oracle-java8-installer
```

FIGUR 90: INSTALLERER JAVA 8 PÅ ET LINUX-BASERT SYSTEM.

```
sudo apt-get install maven
```

FIGUR 91: INSTALLASJON AV MAVEN PÅ ET LINUX-BASERT SYSTEM.

Installasjon av Nginx

```
sudo apt-get install nginx
```

FIGUR 92: INSTALLASJON AV NGINX WEB-SERVER PÅ ET LINUX BASERT SYSTEM.

Når Nginx var ferdig med å installere kunne vi legge filene som webserveren skal bruke for å under **/var/www/html**.

Oppsett av Nginx

Konfigurasjonsfilen som ligger i **/etc/nginx/sites-available/default** ble forandret på for å sette hovedkatalogen til **/var/www/html/dist**. Der dist mappen inneholder de

filene som gjør vår frontend tilgjengelig på web applikasjonen. Vi kjører også backend på samme maskin, så for å videreføre forespørselen tar vi i bruk Nginx proxy:

```
root /var/www/html/dist;

# Add index.php to the list if you are using PHP
index index.html index.htm index.nginx-debian.html;

server_name _;
    error_page 404 /index.html;
location / {
    # First attempt to serve request as file, then
    # as directory, then fall back to displaying a 404.
    #     try_files $uri $uri/ @rewrites;
}
location /auth {
    proxy_pass http://127.0.0.1:8098/auth;
    proxy_pass_request_headers on;
}

location /stream {
    proxy_pass http://127.0.0.1:8098/stream;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}

location /tokenrenewal {
    proxy_pass http://127.0.0.1:8098/tokenrenewal;
    proxy_pass_request_headers on;
}
```

FIGUR 93: VISER HVORDAN PROXY KONFIGURASJONEN ER SATT OPP, OG HVOR FORESPØRSLER VIDEREFØRES.

Oppstart av backend

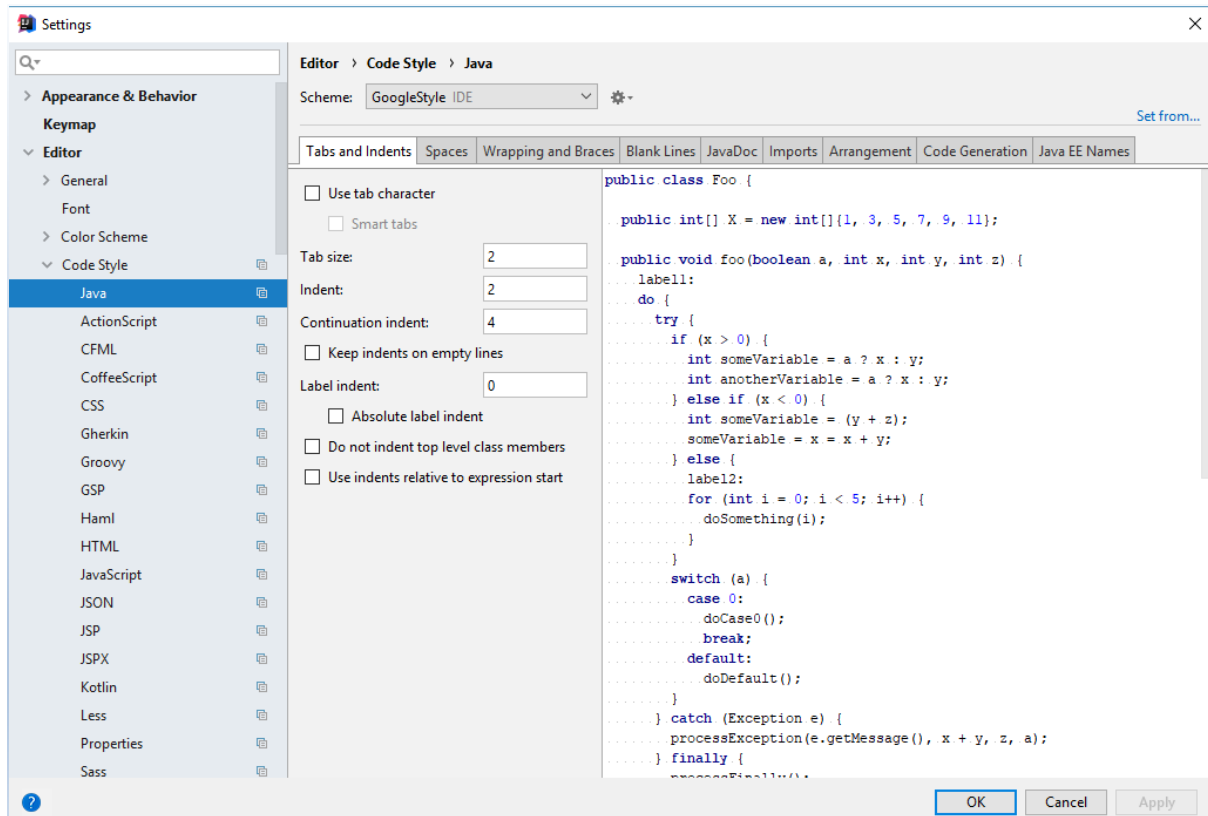
Backend APIet kjøres lokalt på maskinen. Servicen er ikke tilgjengelig for brukeren direkte, unntatt de forespørsel URL-en som er satt til å være tilgjengelig. Applikasjonen kjøres ved at koden lastes ned manuelt fra GitHub, navigerer til nedlastet mappe og starter opp ved følgende kommando:

```
(sudo mvn spring-boot:run)&
```

FIGUR 94: KOMMANDO FOR OPPSTART AV BACKEND APIET PÅ ET LINUX-OPERATIVSYSTEM.

Vedlegg F: Oppsett av utviklingsmiljø for backend

Utviklingen av backend foregikk for det meste i IntelliJ IDEA. IDEen ble satt opp for kode formatering ved bruk av CheckStyle. CheckStyle benytter seg av “Style guides” og vi bruker Google sitt regelsett for å sørge for at koden er riktig formatert. Dette regelsettet er det som brukes i PayEx.



FIGUR 95: BILDE SOM VISER HVORDAN CHECKSTYLE BLIR SATT, OG HVORDAN VI KAN FORANDRE PÅ STANDARD REGLER.

Oppsett av Maven

[Maven](#) sin konfigurasjonsfil ligger i grunnmappen ved navn *pom.xml*. Her konfigurere vi hvilke avhengigheter prosjektet har.


```
<!-- Inherit Spring Boot defaults -->
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.0.0.RELEASE</version>
  <relativePath/>
</parent>

<!-- Spring Boot for web application -->
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
</dependencies>
```

FIGUR 96: HER SER VI ET UTRAG FRA FILEN POM.XML SOM VISER HVORDAN AVHENGIGHETEN SPRING BOOT ER LAGT TIL.

Konfigurasjon av applikasjonen

Konfigurasjon for applikasjonen ligger i *application.properties*. Her definerer vi hvilken port applikasjonen skal kjøre på, valgmuligheter for mappingen av data til JSON, og hvor loggfilene skal lagres hen.

```
# Common properties that all profiles inherit from
spring.application.name=pospay-backend-monitor
server.port=8098

# Jackson
spring.jackson.serialization.indent-output=true

# Spring MVC
spring.mvc.throw-exception-if-no-handler-found=true
spring.resources.add-mappings=false

# Logging
logging.file=/var/log/pospay/pospay-monitor.log
pospay.monitor.jwt.secret=SecretPayExTokenKey
```

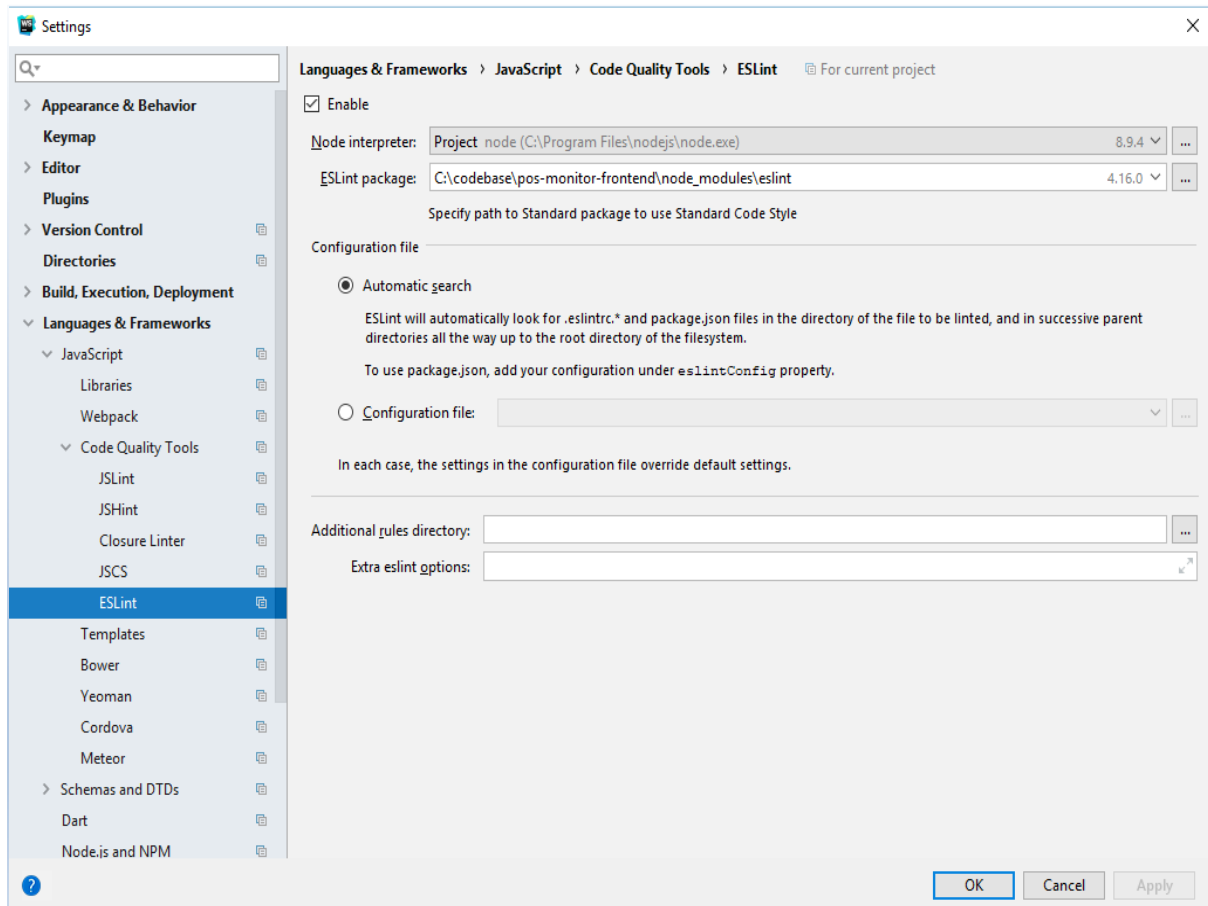
FIGUR 97: UTRAG FRA APPLICATION.PROPERTIES.

Vedlegg G: Oppsett av utviklingsmiljø for frontend

Vi har satt opp validering av kode formatering, en service som enkeltstående kjøres som webserver på datamaskinen og dynamisk oppdatering av det som vises på webserveren når koden forandrer seg. Alle disse funksjonene kjøres via WebStorm.

ESLint

For å ta i bruk ESLint må [NodeJS](#) være installert. Eslint ble lastet ned og lagt til WebStorm ved å trykke på **File->Settings**:



FIGUR 98: VELGER FILEN DER ESLINT PAKKEN LIGGER. PAKKEN HAR BLITT LASTET NED FRA ESLINT SIN OFFISIELLE SIDE.

Hver gang vi formaterte koden feilen fikk vi opp en feilmelding som fortalte oss om hva som var feil:

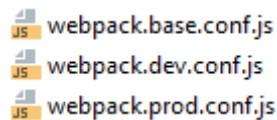
```
Errors:
  1  http://eslint.org/docs/rules/key-spacing

  X  http://eslint.org/docs/rules/no-unused-vars  'Graph' is defined but never used
     src\router\index.js:7:8
     import Graph from '@components/Graph';
     ^
```

FIGUR 99: HER HAR VI IMPORTERT ET KOMPONENT, MEN IKKE TATT DET I BRUK. DET FORTELLER ESLINT OSS OM.

webpack

[Webpack](#) ble installert ved bruk av NPM og konfigurert. Vi har en grunnleggende konfigurasjonsfil. Konfigurasjonsfilen kjøres når applikasjonen er under utvikling, og når applikasjonen gis ut i produksjon.



FIGUR 100: OVERSIKT OVER DE FORSKJELLIGE KONFIGURASJONSFILENE.

Konfigurasjonsfilene definerer hvor applikasjonen skal kjøres på en port. Dersom ingen port er definert vil applikasjonen kjøre på en tilfeldig port:

```
module.exports = new Promise((resolve, reject) => {
  portfinder.basePort = process.env.PORT || config.dev.port;
  portfinder.getPort((err, port) => {
    if (err) {
      reject(err)
    } else {
      // publish the new Port, necessary for e2e tests
      process.env.PORT = port;
      // add port to devServer config
      devWebpackConfig.devServer.port = port;
    }
  });
});
```

FIGUR 101: VISER ET LITE UTRAG FRA FILEN WEBPACK.DEV.CONF.JS. DER VI SETTER PORTEN PROSESSEN SKAL KJØRE PÅ.

Vi kjører applikasjonen ved bruk av kommandoen “**npm run dev**”. Applikasjonen kjøres på maskinen med port 8080. Hvis det gis en feilmelding om at avhengigheter ikke er installert, blir de installert ved å kjøre kommandoen “**npm install**”. Webpack kan brukes til å generere filer som skal kjøres over en egen webserver. Disse filene blir generert ved å ta i bruk konfigurasjonsfilen `webpack.prod.conf.js`.

```
C:\codebase\pos-monitor-frontend>npm run build

> pos-monitor-frontend@1.0.0 build C:\codebase\pos-monitor-frontend
> node build/build.js

Hash: dc0101ece9ce105165ec
Version: webpack 3.10.0
Time: 38779ms

Asset      Size  Chunks  Chunk Na
mes
static/js/vendor.3e8345ea670f8928aa13.js  781 kB    0  [emitted]  [big]  vendor
static/js/app.d8ab8d764e517da549ac.js    65.1 kB    1  [emitted]  app
static/js/manifest.c58f3fdc6d665e9105ea.js  1.49 kB    2  [emitted]  manifest
static/css/app.c4c5263d21fb5bfccaa0003fec32d99e.css  162 kB    1  [emitted]  app
static/css/app.c4c5263d21fb5bfccaa0003fec32d99e.css.map  235 kB    1  [emitted]
static/js/vendor.3e8345ea670f8928aa13.js.map  3.28 MB    0  [emitted]  vendor
static/js/app.d8ab8d764e517da549ac.js.map    205 kB    1  [emitted]  app
static/js/manifest.c58f3fdc6d665e9105ea.js.map  7.79 kB    2  [emitted]  manifest
index.html  585 bytes  [emitted]
static/myfavicon.ico  27 kB    [emitted]
```

Build complete.

FIGUR 102: FILENE SOM KAN TILBYS OVER EN WEBSERVER BLIR GENERERT, OG LAGT TIL I "DIST" MAPPEN.

Vedlegg H: Feilmeldinger som sendes til frontend

```
{
  "title": "Internal Server Error",
  "status": 500,
  "detail": "Exceptional!"
}
```

FIGUR 103: VISER RETURNERT FEILMELDING VED Å BESØKE URLLEN LOCALHOST:8098/EXEC BRUKES FOR Å TESTE OM FEILMELDINGER BLIR RETURNERT PÅ JSON-FORMAT.

```
{
  "detail": "Wrong username and password",
  "status": 403
}
```

FIGUR 104: FEILMELDING SOM MOTTAS NÅR BRUKEREN TASTER INN FEIL BRUKERNAVN OG PASSORD.

```
{
  "detail": "Invalid request",
  "status": 403
}
```

FIGUR 105: FEILMELDING SOM MOTTAS NÅR EN FORESPØRSEL MED FEIL MEDIA-TYPE SENDES.

```
{  
  "detail": "Bad body",  
  "status": 400  
}
```

FIGUR 106: FEILMELDING SOM MOTTAS NÅR HTTP KROPPEN HAR FEIL FORMATTERING.

Vedlegg I: Forslag/ønsker fra PayEx



1.1 Forslag/ønsker fra PayEx

- Ha en reaktiv frontend, slik at dataene blir tegnet så fort som mulig i browser med minst mulig overhead
- Separert frontend fra backend (muliggjøre andre typer klienter senere)
- High performance serialisering (protobuf eller andre binære protokoller)?
- Frontend må kunne vise ulik data basert på rettighetsnivå og/eller egne brukerpreferanser
- Må ikke påvirke systemene som overvåkes (ikke kunne ta ned/påvirke systemene unødvendig negativt)
- Må være real-time, slik at man kan se trender ettersom de oppstår
- Må kunne være tilgjengelig over offentlig internett, gjennom sikre kanaler (TLS, autentisering/autorisering)
- Så lite state i backend som mulig. Dette muliggjør at man kan spinne opp nye instanser av backenden hvor som helst hvis en eller flere instanser dør.
- Ha et godt UI/UX, slik at løsningen faktisk kan brukes
- Gi muligheten til å gi et godt oversiktsbilde når man tar et kort kikk (<5 sek) på monitoren
- Hvis mulig: Mulighet for å sette opp notifikasjoner for gitte events som man kan definere. F.eks. hvis det tar over N millisekunder å prosessere en transaksjon, send en SMS til vakttelefonen, eller hvis innløser X har lang svarstid, send et varsel på Slack.

Vedlegg J: Dagbok

Uke 2

8. Januar Møte: PayEx viste fram kravspekken, hvordan vi skal gå frem og hvordan kommunikasjonen kommer til å være fremover: Kommunikasjonen er i hovedsak med Even, som kvalitet sjekker koden men Dani er produkteier, samt er med på sprint planleggingen.

Verktøy og utstyr:

- Frontend: Vue, Javascript, UI/UX
- Begge deler: Office 365
- Backend: Influxdb, Nginx, Java: Spring, springboot., Linux, RabbitMQ (AMQP), Servere
- Kommunikasjon: fikk payexkonto (mail), Slack, GitHub, Teamleder Dani, systemutvikler Even og får en front end designer.
- Utviklingsmetodikk: smidig Scrum, jobbe i 2 ukers sprinter

Uke 3

Jobbet med forprosjektrapport, laget først vår egen skisse over systemet, etter samarbeid med med PayEx så fikk vi den endelige skissen over systemet.

OBS! Vi skulle ikke touche transaksjonsserveren, databasen influxdb og annet sensitiv informasjon. Disse ligger i en secure zone. Dette skal PayEx ta seg av.

Vår oppgave i backend:

- Ta imot pakker fra monitor serveren. Videreføre de ved å bruke rabbitMQ og gjennom fis og flas til slutt vise fram informasjonen til brukeren.
- Brukere skal grupperes i grupper der de forskjellige gruppene har forskjellige rettigheter. Disse rettighetene bestemmer hva/hvor mye de kan gjøre på nettsiden.

Uke 4

Møte 24. januar: Dagens oppgave var å sette opp miljøet slik at vi kan komme i gang med prosjektet:

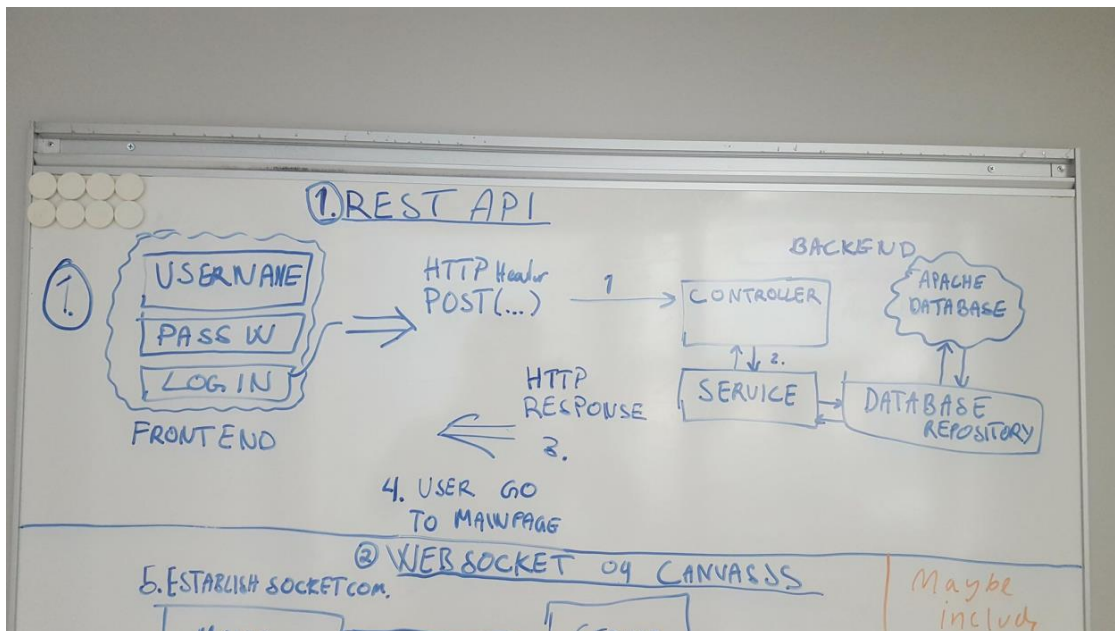
- Tilgang til git-miljøet til PayEx og repositorer som backend. Laster ned og kobler sammen repositoret til IntelliJ IDEA: ssh key osv og oppdaterer .git cfg med kommandoer som ligger i slack.
- Last ned JetBrains toolbox: laste ned IntelliJ IDEA ultimate edition og Datagrip, samt webstorm om ønskelig.
- Fikk brukerkontoer i Atlassian Jira, RSA-brikke og adgangskort.
- Lastet ned Node.js, Webstorm og satt opp en Vue-template frontend ved å sette opp front end miljøet til Vue.
- TeamCity: tester vår kode automatisk når den pushes til GitHub
- PayEx remote services: nettleserbasert VPN-tilgang til PayEx' interne nettverk

På torsdagen snakket produkteierne også om hvordan transaksjonsdataen kommuniserer. I dag sendes data i batcher, det kommer til å være ny standard som kommer til spesifiseres.

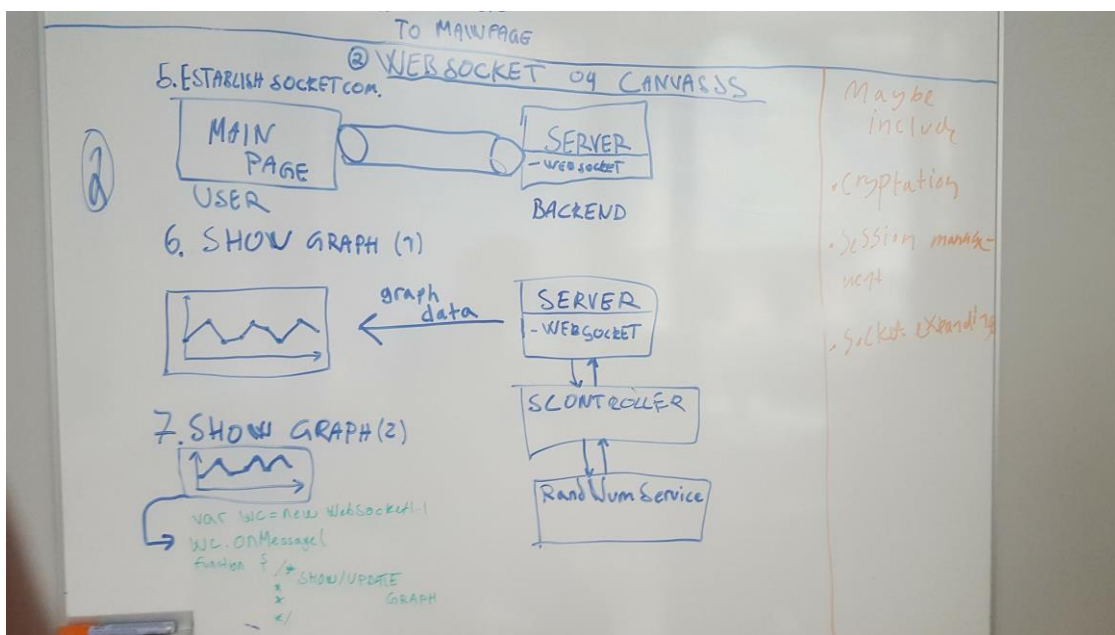
Uke 5

Først lager vi oversikt over hvilke oppgaver/steps som må gjennomføres.

Part 1: Steg 1-4



Part 2: Steg 5-7h



Med produkteier avtalte vi planleggingsmøte til neste uke, men alle gruppe medlemmene satt og jobba med å bli kjent med systemet samt lage utkast til hvordan systemet kan se ut.

Uke 6

I denne uken startet vi sprintmøte med Payex der vi finpusset på storypoints og tasks i sprinten og backloggen. Vi lærte oss ordentlig godt om hvordan man setter opp storypoints, hva en storypoint egentlig er og fjernet visse storypoints som egentlig ikke var storypoints.

Storypoint skal ha en funksjonalitet for brukeieren, noe som kan brukes. Dette gjorde ikke vi når vi var altfor spesifikke og detaljert. Vi jobber agilt så dette trengtes ikke.

Vi fant ut at vi skulle bruke Spring security med JWT (Json web tokens) istedenfor vår egen hardkodet security som sjekker mot en database. Brukerdatabasen ble fjernet fordi vi brukte for mye tid på den for å få den opp, men, siden det skulle kobles opp mot LDAP senere i prosjektet.

Vi implementerte JWT Security oppå DB løsningene.

Uke 7

Denne uken var levering av første sprint.

12-13 februar: Vi implementer og rydder koden (fjernet alt som hadde med DB løsningene å gjøre), fått til handshake med koden. Vi går inn til kode review med Even og får beskjed om at mye kan optimaliseres, i løpet av disse to dagene finpusser vi koden og gjør klar til sprint slutt.

14-15 februar: dokumenter koden og fikser små feil, det ble da javadoc. Vi kobler frontend til backend og får grafene til å tegne ut.

16 februar: Sprint Review: får beskjed om at det vi har gjort er bra hittil men de ønsker et graf bibliotek som har mer funksjonell støtte. Vi brukte resten av dagen på å planlegge til neste sprint hovedfokuset på neste sprint skal være frontenden, det har også kommet inn en UX ressurser som skal guide eos gjennom dette.

Uke 8

Mandag: Sprint planlegging, med valg av user stories som skal representeres og tasker som skal gjøres i denne sprinten. UX-ressurssen kom inn i bilde, og det ble da valgt bootstrap v. 4.0 som UX-rammeverk.

Onsdag-Fredag: Kent jobber med backendt med renewal of token, Ali og John bruker tiden på å researche grafene. Resultatet kan sees på som vedlegg; [stresstest](#)

Uke 9

Shamil fullførte styling av websiden og kodet error messaging.

John og Ali fullførte researching og sammen med teamet kom fram til et graf bibliotek som vi skal bruke framover(møtet ble tatt på onsdag).

Kent fullførte token review og nå vil brukeren sin token valideres etter 15 min.

Det var sprint demo på fredag der det kom fram at stilen er bra, og de ønsker nå og se mer data rundt selveste gjennomsnittstid, median, osv.

Grafer som viser status over et gitt tidsintervall som for eksempel 60 minutter.

Uke 10

Mandag: Sprint planlegging, med valg av user stories som skal representeres og tasker som skal gjøres i denne sprinten. Det skal da være fokus på grafer, samt backend skal regne ut gjennomsnittstid, median o.l. Det ble da valgt at denne sprinten skal fokusere på error-håndtering, global Vuex storage, mer info rundt transaksjonene og dataene som sendes ut.

Onsdag: Shamil: fikset opp colors, Ali: fikset view av feil brukernavn og passord, noe Javadoc samt bug fix av renewal of token. Kent: Oppgraderte sending av dataene i formen av:

```
private int maximumTxPerSecond;  
private int minimumTxPerSecond;  
private int medianTxPerSecond;  
private int upperTxPerSecondQuartile;  
private int lowerTxPerSecondQuartile;
```

John: gjorde ferdig global storage, samt redirect on token expiry og fått expr date ut av token.

Uke 11

Kent fullførte oppgradering av “dummy” data som skal regne ut median, gjennomsnitt, epoker og regne ut og vise informasjon om antallet transaksjoner.

De siste dagene før demovising finpusset vi på styling på nettsidene og grafene, og vi klarte til demoen å vise mer transaksjonsinformasjon til de forskjellige kundene på dashboard, og de andre graf sidene. Det som var viktig var at grafene skulle se ut som en graf, den skulle se logisk ut og ha en historikk.

Demovising: De var fornøyde med sprinten og vi avtalte sprint møte på mandag.

Uke 12

Mandag var det sprint møte og vi ble enige om oppgaver rundt sprinten.

Vi jobbet mindre på produktet og kom sakte i gang, og ble ikke helt ferdig med deler av sprinten.

Frontend: Vi stylet ferdig menyen og gjorde den responsive til alle enheter. I tillegg la til vi en status indicator som kommer opp når backend har blitt skrudd av. Når backend skrus på igjen, vil monitoren kjøre opp igjen. Vi kom litt i gang med å rydde opp og slette kode.

Backend: Optimalisere litt og finpusset på backend koden. Mest av koden som ble endret var noe som gjaldt transaksjons dataen.

Uke 14

Fullføre: grafene som boxplots og volume graphs, testing i både frontend og backend. Fullføre statusindikator og litt styling på farger.

Onsdag: skrev tester i backend mot datalageren som inneholder transaksjonsdataen.

Uke 15

Jobbet videre med å få til boxplot og volum grafer. Vi skulle ferdigstille, fikse på fargen og få ferdig funksjonen rundt dette. Vi jobbet også med status indikatoren, der vi tok den ut til egen komponent slik at den kan brukes der det er ønskelig.

Vi hadde så et avslutningsmøte der vi snakket om D3 implementasjonen. Vi snakket om fargene, responsivitet og hva som kan forbedres. Vi fikk beskjed om å bruke PayEx sine farger. Vi testet så ut responsiviteten på D3 implementasjonen, og det viste seg at det ikke var optimal til små enheter. Vi gikk så videre med å sjekke responsiviteten og designet for små enheter, og det var del som kunne forbedres.

Uke 16

Vi hadde et planleggingsmøte der vi diskuterte mye av det vi gjorde på avslutningsmøte uken før. Vi skrev ned oppgaver til produktkøen og ga dem story points. Vi jobbet så resten av uken for å få på plass responsiviteten, rydde opp ved å ta ut koden til komponenter slik at den kan gjenbrukes, samt finpusse på det kosmetiske utseende.

Uke 17

Denne uken jobbet vi med å få plass responsiviteten, samt få en pop up boks som forteller brukeren om diverse. PayEx selv skal implementere flere feilmeldinger, men vi fikk beskjed om å implementere en feilmelding når token er ugyldig og du blir kastet ut.

Vi hadde så et avslutningsmøte på fredag, der vi viste fram det vi hadde gjort og ble enige om å jobbe videre med responsivitet, ha egen views for merchants, samt sette grunnlaget for utsending av varsler ved avvik.

Uke 18

Denne uken hadde vi ingen planleggingsmøte, da det ble holdt på fredagen. To av grupped medlemmene jobbet for å få plass responsiv og merchant view, imens Ali og Shamil satt og jobbet med rapporten.

Uke 19

Denne uken jobbet Shamil med responsiviteten, John satt med Merchan View. Kent og Ali satt med rapportskrivningen. Vi hadde også et møte med veileder der vi fikk

beskjed om å skrive litt om/ utdype ord og uttrykk i rapporten. Vi fikk beskjed fra arbeidsgiver at avslutningsmøte er flyttet framover på grunn av kristenhimmelfartsdag og inneklemdag.

Uke 20

Denne uken ble det i all hovedsak jobbet med rapporten, vi lagde diagrammer, skrev videre på funksjonalitet, og rettet opp i skrivefeil. Vi jobbet med å dokumentere ferdig applikasjonen.

Uke 21

Vi ferdigstilte rapporten på tirsdag, leste igjennom den og rettet på ting underveis. Vi reformaterte rapporten sa oss ferdige og leverte den.

Kilder

As, P. N. (2018). "Om PayEx." Retrieved 21 May, 2018, from <https://payex.no/om-payex/>.

Brown, K. (2016). "What is GitHub, and what is it used for?". Retrieved 21 May, 2018, from <https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/>.

Foundation, T. A. S. (2018). "Introduction to the POM

". Retrieved 21 May, 2018, from <https://maven.apache.org/guides/introduction/introduction-to-the-pom.html>.

Foundation, T. A. S. (2018). "Welcome to Apache Maven." Retrieved 21 May, 2018, from <https://maven.apache.org>.

KC, A. (2017). "Understanding spring security and filter chain." Retrieved 21 May, 2018, from <http://anilkc.me/understanding-spring-security-filter-chain/>.

Rouse, M. (2014). "Back-end." Retrieved 21 May, 2018, from <https://searchdatacenter.techtarget.com/definition/back-end>.

Software, J. (2018) "Choose a workflow, or make your own." Retrieved 21 May 2018, from <https://www.atlassian.com/software/jira>.

Software, M. G. (2018) "Product Owner." Retrieved 21 May, 2018, from <https://www.mountaingoatsoftware.com/agile/scrum/roles/product-owner>.

Software, P. (2018). "Main Projects." Retrieved 21 May, 2018, from <https://spring.io/projects>.

Software, P. (2018). "The Security Filter Chain." Retrieved 21 May, 2018, from <https://docs.spring.io/spring-security/site/docs/3.0.x/reference/security-filter-chain.html>.

Software, P. (2018). "Spring Boot." Retrieved 21 May, 2018, from <https://projects.spring.io/spring-boot/>.

Software, P. (2018). "Spring Security." Retrieved 21 May, 2018, from <https://projects.spring.io/spring-security/>.

Software, P. (2018). "Testing." Retrieved 21 May, 2018, from <https://docs.spring.io/spring/docs/current/spring-framework-reference/testing.html>.

Svorstøl, S.-O. (2015). "En rask introduksjon til UML." Retrieved 21 May, 2018, from <https://www.ntnu.no/wiki/display/ttd4140/En+rask+introduksjon+til+UML>.

Tutorialspoint (2018). "Spring Framework- Architecture." Retrieved 21 May, 2018, from https://www.tutorialspoint.com/spring/spring_architecture.htm.

W3School (2018). "JSON-introduction." Retrieved 21 May, 2018, from https://www.w3schools.com/js/js_json_intro.asp.

Wikipedia (2017). "JSON." Retrieved 21 May, 2018, from <https://no.wikipedia.org/wiki/JSON>.

Wikipedia (2017). "Scrum." Retrieved 21 May, 2018, from <https://no.wikipedia.org/wiki/Scrum>.

Wikipedia (2018). "Apache Maven." Retrieved 21 May, 2018, from https://no.wikipedia.org/wiki/Apache_Maven.

Wikipedia (2018). "Spring Framework." Retrieved 21 May, 2018, from https://no.wikipedia.org/wiki/Spring_framework.