Lyhyesti

Yksikkötestit
- Ajettavissa komennolla mvn clean test
- Tuottaa raportin polkuun target/site/jacoco/index.html
- Testien kappalemäärä: JUnit-testejä 172 kappaletta ja Jasmine-testejä 22 kappaletta
- Rivikattavuus: JUnit 92%, JavaScriptillä vain julkiset metodit on testattu

Integraatiotestit
- Enimmäkseen hyväksymätestejä
- Ajettavissa komennolla mvn clean integration-test
- Testien kappalemäärä 20

Mutaatiotestaus
- Ajettavissa komennolla mvn org.pitest:pitest-maven:mutationCoverage
- Tuottaa raportin polkuun target/pit-reports/index.html
- Mutaatiotestien kappalemäärä 811
- Mutaatioista eliminoitiin 82%

Viimeisimmät itse ajamiemme mutaatiotestien ja hyväksymätestien raportit löytyvät projektin alakansiosta documents/testReports.

Miten sovellusta on testattu?

Testit on kirjoitettu sovellukselle lasilaatikko-menetelmällä ja ne voidaan jakaa kolmeen eri tyyppiin:

1) **Java-luokkien yksikkötestit** on toteutettu seuraavasti:
    a) JUnit-testeillä, jotka ajetaan tavallisina testitiedostoina
    b) JUnit-testeillä, jotka ajetaan MockitoJUnitRunner-luokalla ja joissa testattavan luokan riippuvuuksista on toteutettu Mockiton avulla tynkä-luokat. Tämä siksi, että riippuvuudet toimisivat ennustettavalla tavalla ja testissä voitaisiin keskittyä vain testattavan luokan toimintaan sillä oletuksella, että riippuvuuksille on omat testit, jotka testaavat, että riippuvuuden metodit toimivat oikealla tavalla niitä kutsuttaessa.
    c) JUnit-testeillä, jotka ajetaan SpringJUnit4ClassRunner-luokalla käyttäen samaa ohjelman käynnistyskonfiguraatiota kuin ohjelmaa normaalisti käynnistettäessä. Siis testejä ajettaessa Spring käynnistetään ajamalla Launcher-luokan launch-metodi. Kun tällainen testi ajetaan, käyttäytyy testi-metodi kuin ohjelma olisi käynnistetty normaalisti ja Spring olisi alustanut kaikista luokista beanit oliosäiliöönsä.
    d) JUnit-testejä, jotka on ajettu SpringJUnit4ClassRunner-luokalla, mutta käynnistyskonfiguraatio on määritelty itse erikseen. Tällaisia testejä on

käytännössä vain yksi erikoistesti, jolla testataan, että WebSocket-yhteyden kautta lähetetyt viestit päätyvät kontrollerin metodille, ja vastaavasti kontrolleri-metodi kykenee lähettämään vastauksen WebSocket-yhteyden kautta.

2) **Integraatiotestit** on toteutettu JUnit-testeinä, jotka ajetaan käyttäen WebDriverRunner-luokkaa. Integraatiotesteissä sovellusta käytetään Selenium WebDriverin avulla, eli erilaisia sivun nappeja painellaan ja annetaan erilaisia syötteitä, ja testit vertaavat WebDriverin sivulla näkemää sisältöä odotettuun sisältöön varmistaakseen, että sovellus toimii oikealla tavalla. Käyttäjätarinoiden hyväksymätestaus on toteutettu tällaisina integraatiotesteinä.

3) **JavaScriptin yksikkötestit** on toteutettu Jasminella.

Edellä mainittujen testien lisäksi olemme suorittaneet jatkuvasti black box -testaamista, kuten esimerkiksi tutkivaa testausta siten, että olemme avanneet yhteen selaimeen (esim. Google Chrome) kirjautuneen käyttäjän, ja toiseen erilaiseen selaimeen (esim. Mozilla Firefox) kirjautumattoman käyttäjän, jotta olemme voineet simuloida erilaisia asiakkaan ja hoitajan välisiä kanssakäymisiä, ja tällä tavalla tutkia ohjelman toimintaa.

Miten testit ajetaan?

Sekä Javan-luokkien, että JavaScriptillä tehtyjen metodien yksikkötestit ajetaan käyttäen Maven-surefire-pluginia esimerkiksi komennolla mvn clean test. Java-luokkien yksikkötestit löytyvät src/main/test/java/sotechat-kansion alta. JavaScript-metodien Jasmine-testit puolestaan löytyvät src/main/test/webapp-kansion alta.

Kun komento mvn clean test ajetaan, Jacoco-maven-plugin luo raportin ajettujen Java-luokkien yksikkötestien rivi- ja haaraumakattavuudesta. Raporttia voi tarkastella avaamalla selaimella tiedoston target/site/jacoco/index.html.

Yksikkötestejä voi mutaatiotestata pitest-maven-pluginilla komennolla mvn clean org.pitest:pitest-maven:mutationCoverage. Mutaatiotestauksesta luodaan raportti, jota voi tarkastella avaamalla selaimessa tiedoston target/pit-reports/index.html.

Integraatiotestit löytyvät src/main/test/java/integrationTests-kansion alta (lähinnä hyväksymätestejä). Integraatiotestit ajetaan Maven-failsafe-pluginin avulla komennolla mvn clean integration-test.

Checkstyle

Ohjelman toiminnallisuuden lisäksi olemme testanneet ohjelmakoodin ylläpidettävyyttä maven-checkstyle-pluginin avulla. Tämä plugin ajetaan komennolla mvn clean checkstyle:checkstyle ja se luo target/site-kansioon raportin tiedostoon checkstyle.html. Raportista näkee, kuinka hyvin ohjelmakoodi noudattaa

src/main/resources/Checkstyle.xml-tiedostoon määriteltyjä koodin ulkoasua koskevia sääntöjä.

Olemme määritelleet Checkstylen siten, että esimerkiksi seuraavanlaisessa koodissa Checkstyle huomauttaa, että ohjelmakoodi alkaa samalta riviltä kuin metodin nimi:

Ei siis näin:

```
  public Conversation() {this.messagesOfConversation = new
ArrayList<>();this.participantsOfConversation = new ArrayList<>();
    }
```

...vaan näin:

```
    public Conversation() {
        this.messagesOfConversation = new ArrayList<>();
        this.participantsOfConversation = new ArrayList<>();
    }
```