Sovellusprojekti Kepler, 2. lähdekoodin katselmointi

Aika Maanantai 11.5.2015 klo 16:15 – 17:20
Paikka Jyväskylän yliopisto, Agora, Sovellusprojektien kokoushuone C226.1

Läsnä
Projektiryhmä
Anu Koskela
Henrik Paananen
Atte Räty
Joonas Konki, sihteeri
Ohjaajat
Petri Partanen
Jukka-Pekka Santanen

Muistio

Muistio laadittu: 18.5.2015
Muokattu: —

1 Katselmoinnin yleiset huomiot

Katselmoinnissa käytiin läpi Partasen kommentteja Kepler-sovelluksen lähdekoodista ja toteutusratkaisuista. Partanen totesi, että ohjelmakoodia on tullut paljon lisää ja sen syvällinen läpikäynti muutamassa tunnissa on mahdotonta. Tämä katselmointi on siis vain pintaraapaisu mahdollisiin ongelmiin.

Varausten lisätietoihin annettavan viestin pituudelle tulisi asettaa yläraja. Käyttöliittymän tulisi rajoittaa tekstikentän kirjainmäärää. Palvelinpuolen sovelluksen tulisi tallentaa vain rajallinen määrä tekstiä, esimerkiksi 1000 kirjainta. Santanen huomautti, että käyttöliittymän tulisi ilmoittaa liian pitkästä viestistä tai kertoa viestin maksimikoko. Lisätietokenttiin täytyy toteuttaa XSS-sanitointi.

Sovelluksessa on useita väärinkäyttömahdollisuuksia. On mahdollista varata sama vuoro monta kertaa ja vielä kaiken lisäksi toisille henkilöille. On myös mahdollista varata menneitä vuoroja toisille henkilöille. Esimerkiksi Lisa voi varata Annelle vuoroja suoraan konsolikomennolla

```
kepler.addReservation({ time_slot_id: 12, unit_id: 10,
unit_group_id: 2, user_group_id: 3, note: ”Assari on ääliö!”
}, function(){ });
```

Toisille henkilöille on mahdollista luoda mittausryhmiä. Esimerkiksi Lisa voi luoda pääkäyttäjälle ryhmän konsolikomennolla

```
kepler.addUserGroup({ member_id_list: [1,2], group_owner_id:
1, name: ”apinat” }, function(){ });
```

Lisäksi on mahdollista poistaa oma henkilökohtainen ryhmä ja useita muita mittausryhmiä kerralla komennolla

```
for (var i = 0; i < 20; i++) {
kepler.deleteUserGroup(group_id: i, function(){ }); }
```

Kutsu `get_units_editable` palauttaa virheellisesti kaiken tiedon myös opiskelijoille.

Jokainen API-kutsu tulisi testata väärinkäytösten varalta ainakin opiskelijan oikeuksilla!

2 Palvelinpuolen toteutukseen liittyvät huomiot

Kommentointia ja erityisesti docstringejä on lisättävä lähdekoodiin, myös tiedostojen alkuun. Kommentteja kananttaa lisätä myös koodin sekaan, jos koodi ei ole itsestään selvää. Esim. jotkut for-silmukat olisi hyvä selittää, mitä niissä tehdään. API dokumentaation osalta ei tarvitse tehdä toistamiseen Sphinx-kommentteja kunhan liittää itse tuotetut dokumentaatioon mukaan. TODO-kommentit kannattaa käydä vielä huolella läpi.

Pitkiä funktioita kannattaa jakaa pienempiin kokonaisuuksiin. Esimerkiksi `api/reservation.py`: `add_reservation` on jo aika pitkä funktio. Kannattaa määritellä uusia pienempiä funktioita, vaikka sitä kutsuttaisiin vain kerran. Pienempiin palasiin pilkkominen auttaa luettavuutta, sillä funktion nimi kertoo paljon.

`api/user_group.py`: Integer-vertailut on tehtävä aina '`==`'-operaattorilla, ei `is`-operaattorilla, koska se vertaa objekteja toisiinsa, esim. `10000 is (10001 - 1) == false`.

`api/user_group.py`: Sisältää vaarallisen näköisiä ehtoja, jotka eivät ehkä tee sitä mitä halutaan. Lauseke `if params.name is not None or ""` on sama kuin `if (params.name is not None) or ""`. Lisätietoa löytyy esim. `https://docs.python.org/3.4/reference/expressions.html#operator-precedence`

Tyhjän listan ja merkkijonon olemassaolon voi tarkistaa `not`-operaattorilla. `len(lista) == 0` vs. `not lista` tai `teksti == ""` vs. `not teksti`.

Lähdekoodin tyyliä ja virheitä kannattaa analysoida pylint-ohjelmalla. Ohjelman ja ulkopuolisten komponenttien lisenssit täytyy lisätä lähdekoodin mukaan.

3 Käyttöliittymän toteutukseen liittyvät huomiot

Partanen kommentoi yleisesti JavaScript-lähdekoodin olevan pääosin mallikasta.

JavaScript-lähdekoodista puuttuu joitain puolipisteitä lauseiden lopuista. Käytetyt .html() käskyt tulisi korjata, sillä niihin voi helposti syöttää JavaScript-koodia. If-ehtolauseissa kannattaa aina käyttää hakasulkeita. Tarpeettomat varoitukset ja tulostukset konsoliin on poistettava. Lähdekoodin minimisaation käyttö (`.min.js`) tuo myös tietoturvaa (poistaa kommentit), mutta silloin on muistettava puolipisteet.

JavaScript-lähdekoodin tyylin ja virheiden analysointiin kannattaa käyttää JSHint-ohjelmaa ja HTML-sivuille esimerkiksi sivun `https://html5.validator.nu/` HTML5 Validator -ohjelmaa. Tyhjiä rivejä voisi lisätä koodin luettavuuden parantamiseksi. Erityisesti `unit_group.js` kannattaa lisätä tyhjiä rivejä parantamaan luettavuutta.

Muuttujien määrittelyt voi usein yhdistää käyttämällä pilkkua listauksissa. Esimerkiksi `var eka; var toka;` sijaan voisi kirjoittaa `var eka,toka;`.

Kommentointia ja erityisesti docstringejä tulee lisätä (JSDoc, `http://usejsdoc.org`). Kommentteihin kannattaa kirjata myös mahdollisia tiedostettuja huonoja toteutusratkaisuja jatkokehitystä varten.

`kepler.js` sisältää paljon saman toiminnallisuuden toistoa funktioissa. Koodin toistamisen sijaan kannattaa tehdä funktioita. Esimerkiksi callback-ominaisuuden voisi upottaa suoraan `makeCall`-metodiin.

Globaalien muuttujien määrittelystä keskusteltiin. Ne voisi sijoittaa johonkin isompaan globaalin objektin sisään, mikä helpottaisi koodin luettavuutta funktioiden sisällä käytettyjen muuttujien osalta. Tätä ei kannata kuitenkaan enää toteuttaa projektin tässä vaiheessa.

Tietokannan transaktioiden käytöstä käytiin keskustelua. Esim. `add_resourcessa` tulisi huomioida virhetilanteet, joissa jokin yksittäinen transaktio epäonnistuu. Tietokantakäskyt tulisi koota yhteen transaktioon. Käyttäjälle tulee ilmoittaa epäonnistuneesta yrityksestä vaikka yksinkertaisella viestillä ja lokata virheilmoitus jonnekin.