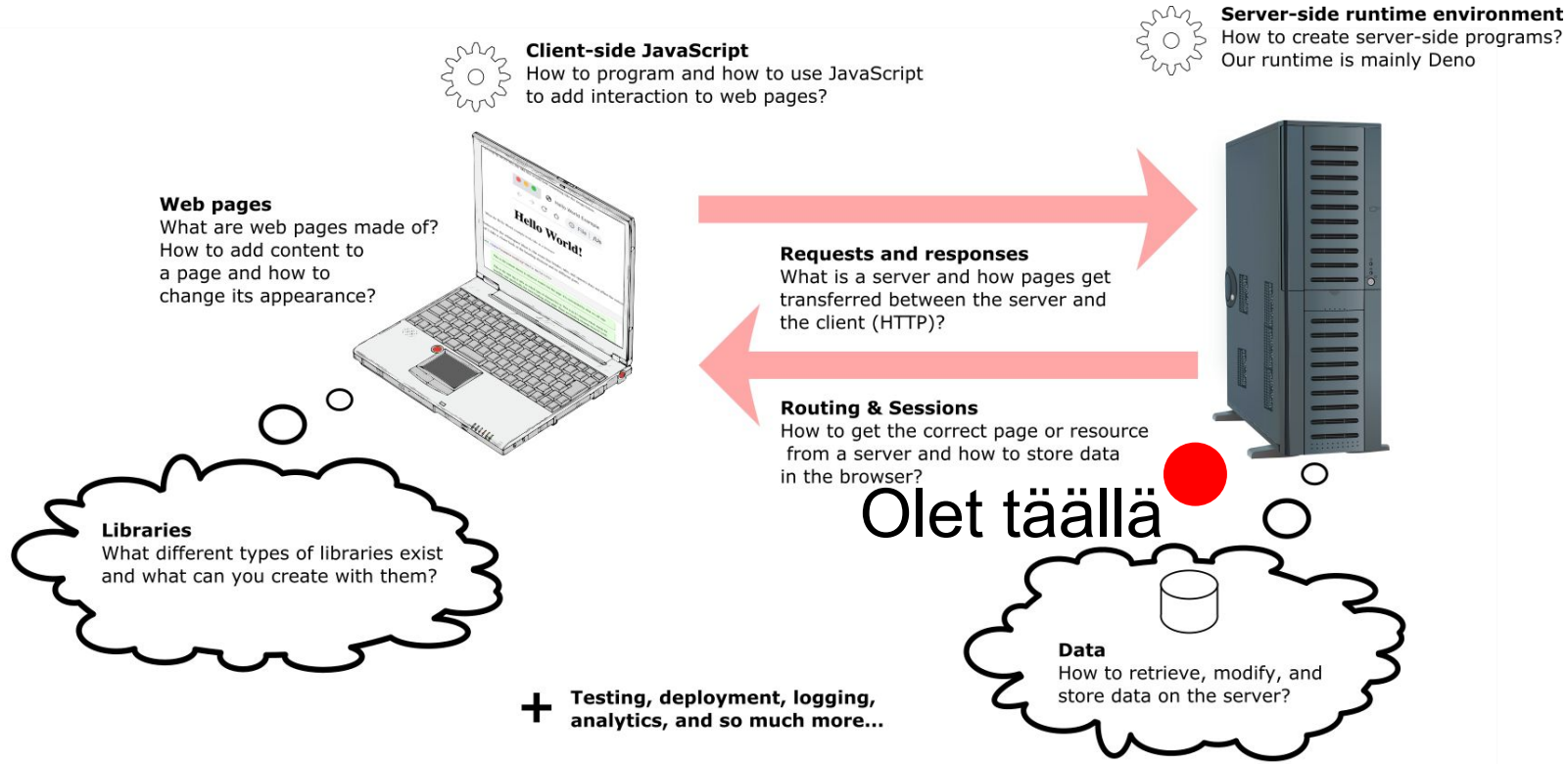
Päivä 5 - Tietokannat sovelluksissa

Web-sovellukset korkealla tasolla



Päivä 5

- 9-12 Aamupäivä
  - Yhteenveto poluista ja HTTP-metodeista
  - Kahvitauko
  - Sivutus ja polkuparametrit
  - Yhteenveto
- 12:00 - 13:00 Lounas
- 13:00 - 16:00 Iltapäivä
  - Muutama sana ORMeista
  - Kerrosarkkitehtuuri, “uusi sovellus lähes tyhjästä”
  - Kahvitauko
  - Kerrosarkkitehtuuri, kaksi erilaista tapaa muodostaa näkymä
  - Yhteenveto

Polut ja metodit

- HTTP-pyynnöt tehdään aina johonkin palvelimen polkuun
  - Polkuun voidaan määritellä parametreja, esim “/todos/:id”, jolloin parametrin arvo haetaan pyynnön yhteydessä
- HTTP-pyynnöt tehdään aina jollain metodilla, joista yleisimmät ovat GET ja POST
  - GET-pyyntöä käytetään resurssin hakemiseen
  - POST-pyyntöä käytetään resurssin muokkaamiseen
- Kutsuttava funktio valitaan HTTP-pyynnön polun ja metodin perusteella
  - Valinta tapahtuu määritellyistä reiteista, esim “routes.get(“/todos/:id”, todoController.findOne)”
  - Funktiossa määritellään mitä tehdään ja minkälaista tietoa palvelin palauttaa
- Polut liittyvät resursseihin → kannattaa nimetä polut resurssien mukaan
  - Käytäntönä usein monikko, esim “items”, “collections”, …

Hands-on - “project-05-todos”

- Lisää mahdollisuus tehtävän (todo) poistamiseen kokoelmasta
  - Tarkastele tiedostoa collection.eta ja määrittele reitti routes.js-tiedostoon
  - Luo sopivaan *Controller.js-tiedostoon funktio, jota reitistä päädytään kutsumaan
  - Luo sopivaan *Service.js-tiedostoon funktio, jota *Controller.js-tiedostosta päädytään kutsumaan
  - Toteuta tehtävän poistamiseen tarkoitettu toiminnallisuus: “DELETE FROM …”
- Lisää tehtävälistaan (/todos) tieto siitä, onko tehtävä merkitty tehdyksi
  - Tietokannassa tehtävällä jo kenttä “completed”
- Lisää tehtävälistaan (/todos) mahdollisuus merkitä tehtävä tehdyksi
  - (1) lomake ja nappi, (2) reitti, (3) kontrolleri, (4) service, (5) sql-kysely
- Lisää kokoelmasivulle (/collections/:id) mahdollisuus merkitä tehtävä tehdyksi
  - (1) lomake ja nappi, (2) reitti, (3) kontrolleri, (4) service, (5) sql-kysely
- Rajaa listaus siten, että tietokannasta haetaan vain tekemättömät tehtävät
  - Oleellisten SQL-kyselyjen muokkaus

Kahvitauko?

Pyynnön parametrit ja sivutus

- Polun ja metodin lisäksi pyyntöön voi lisätä parametreja
  - Esim. “/todos?task=abc” -- tässä parametrina “task”, jolla arvo “abc”
  - Esim. “/todos?page=1” -- tässä parametrina “page”, jolla arvo “1”
  - Löytyvät Oakissa context-objektin url-attribuutin searchParams-attribuutista, joka on URLSearchparams-olio

- Pyyntöparametreja käytetään mm. sivutukseen:
  - Mikäli tietokantataulussa on paljon tietoa, kaikkea ei haeta ja näytetä kerralla
  - Sen sijaan, kerrallaan näytetään “sivullinen” tietoa

- Sivutus käytössä polussa “/todos”-sivulla
  - Kokeile lisätä sivulle yli 5 tehtävää

Hands-on - “project-05-todos”

- Mahdollisuus edelliselle sivulle siirtymiseen näkyvissä vain mikäli se mahdollista
  - Linkki näkyvillä vain tietyissä tapauksissa
  - Kontrollerissa (tai jossain muualla) tarkastellaan, että palvelimelta ei pyydetä olematonta sivua → mikäli näin tapahtuu, tilanne korjataan

- Mahdollisuus seuraavalle sivulle siirtymiseen näkyvissä vain mikäli se mahdollista
  - Linkki näkyvillä vain tietyissä tapauksissa
  - Tässä tarvittaneen uusi tietokantakysely, jossa haetaan rivien määrä
    - SELECT COUNT(*) as count ...
  - Kontrollerissa (tai jossain muualla) tarkastellaan, että palvelimelta ei pyydetä olematonta sivua → mikäli näin tapahtuu, tilanne korjataan

Sivutuksen hyödyt ja haitat

- Hyötyjä
  - Haettava tietomäärä hallinnoitavissa ja navigoitavissa
  - Kun sivutus tehdään tietokannassa, tarpeetonta tietoa ei siirry palvelimelle (ja selaimelle)
  - …
- Haittoja
  - Lisää toteutuksen kompleksisuutta
  - Tiedon etsiminen mahdollisesti hitaampaa
    - Esim. 30 sivun läpikäynti vs yhden sivun “skrollaaminen”
    - Ratkaisuna mm. hakukentät muille arvoille, “infinite scrolling”

Hands-on - “project-05-todos”

- Lisää tehtäväsivulle tekstikenttä, joka mahdollistaa tehtävän hakemisen tietyn nimen perusteella
  - Lisää sivulle lomake, joka tekee GET-pyynnön palvelimen polkuun “/todos”
  - Lisää lomakkeeseen tekstikenttä, jonka nimi “name” on “task” → searchParams-olioon tulee muuttuja “task”, joka sisältää kentän arvon
  - Muokkaa hakutoiminnallisuutta siten, että tehtäviä haettaessa haetaan tehtäviä, joiden nimessä esiintyy haettu merkkijono (SELECT * FROM … WHERE task ILIKE ‘%haettava%’)
  - Mikäli aikaa jää, lisää muuta toiminnallisuutta (esim. hakutulosten sivutus)

Lounas

Muutama sana ORMeista

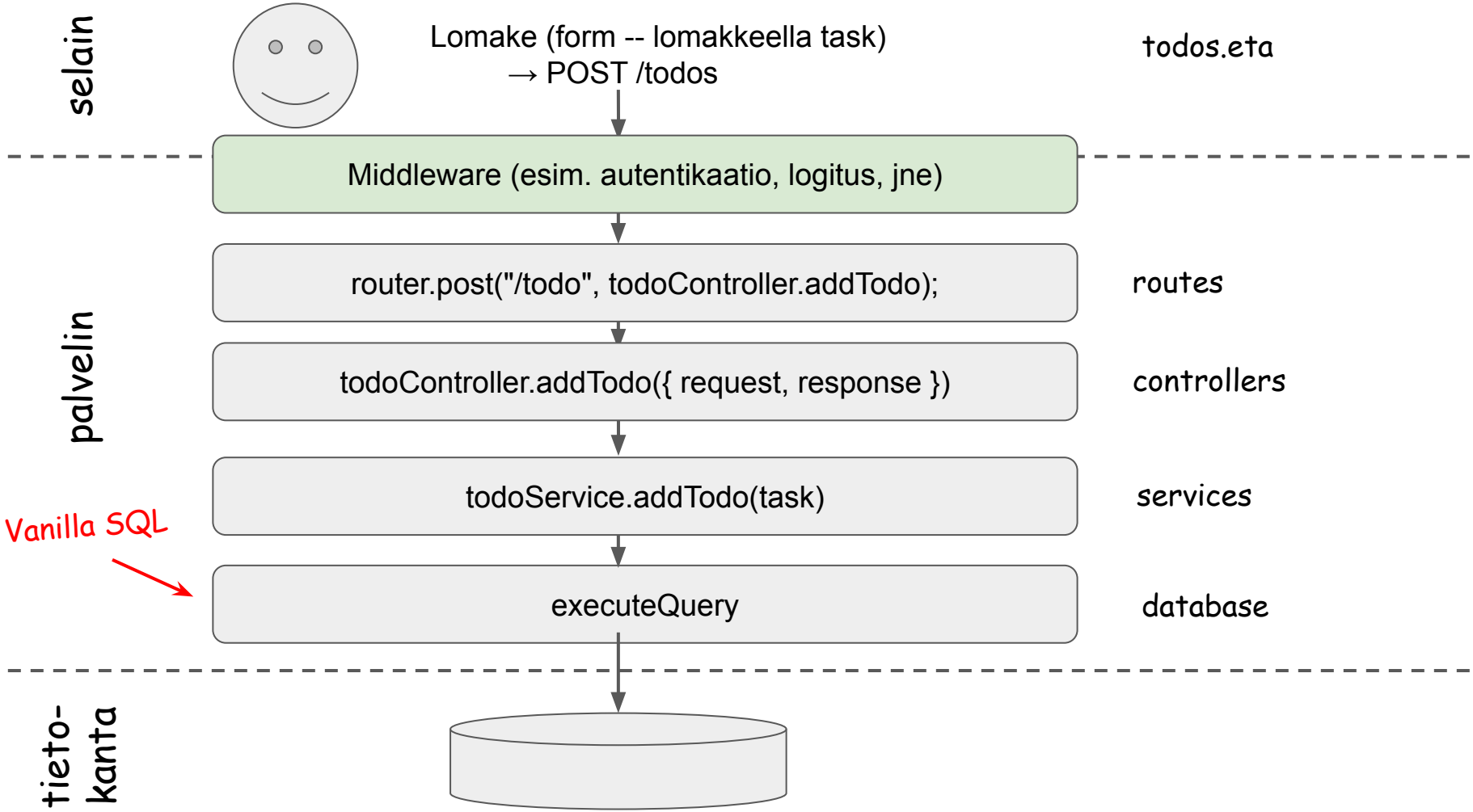
- Object Relational Mapping (ORM)
  - Mahdollistaa tietokantataulujen määrittelyn luokkina
  - Kyselyt luokkien avulla niiden metodeja käyttäen

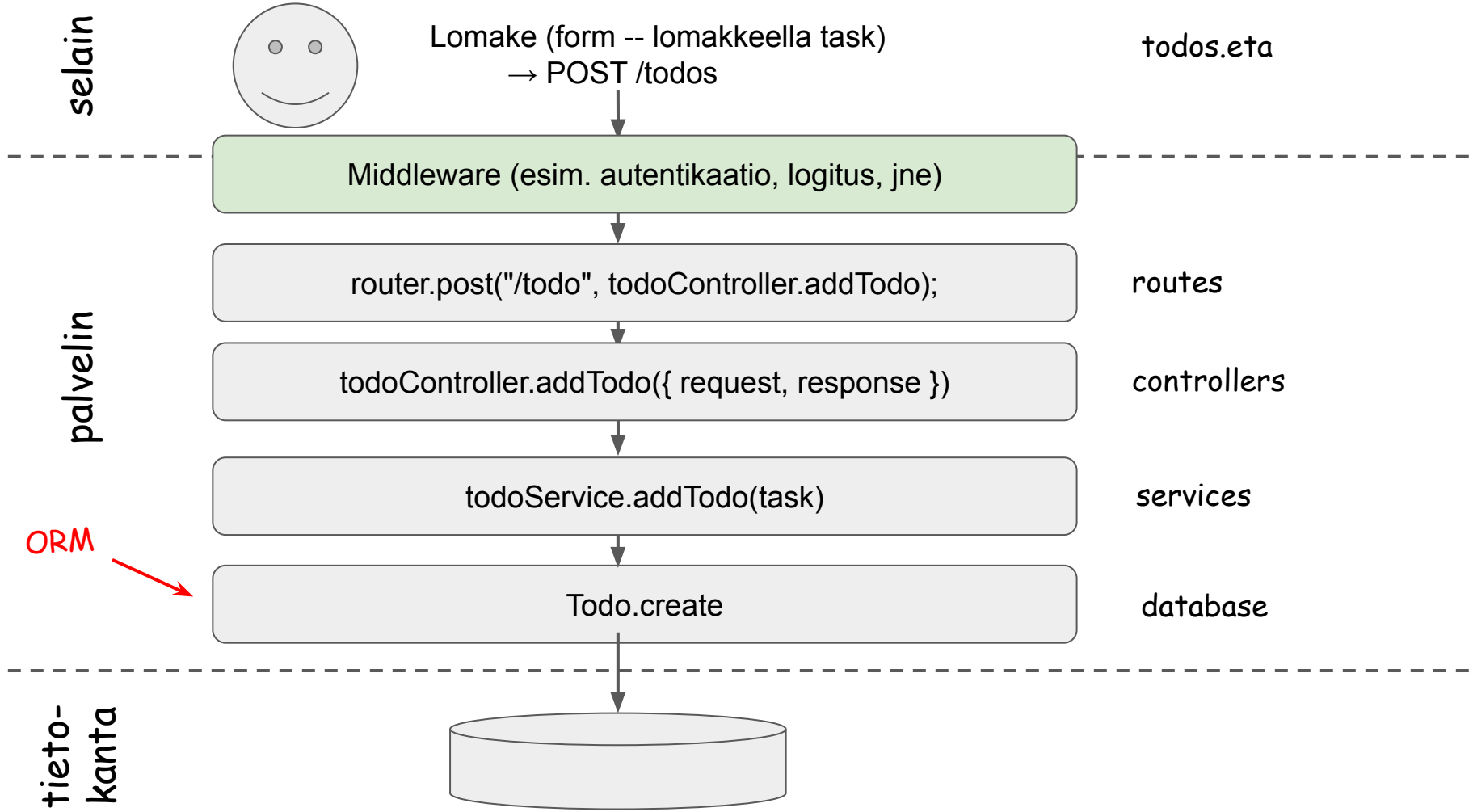
- Hyötyjä
  - Kyselyjen kirjoitus usein nopeampaa ORM-kirjastolla, mikäli kirjasto tuttu
  - Ainakin yksinkertaiset kyselyt ymmärrettävämpiä koodissa

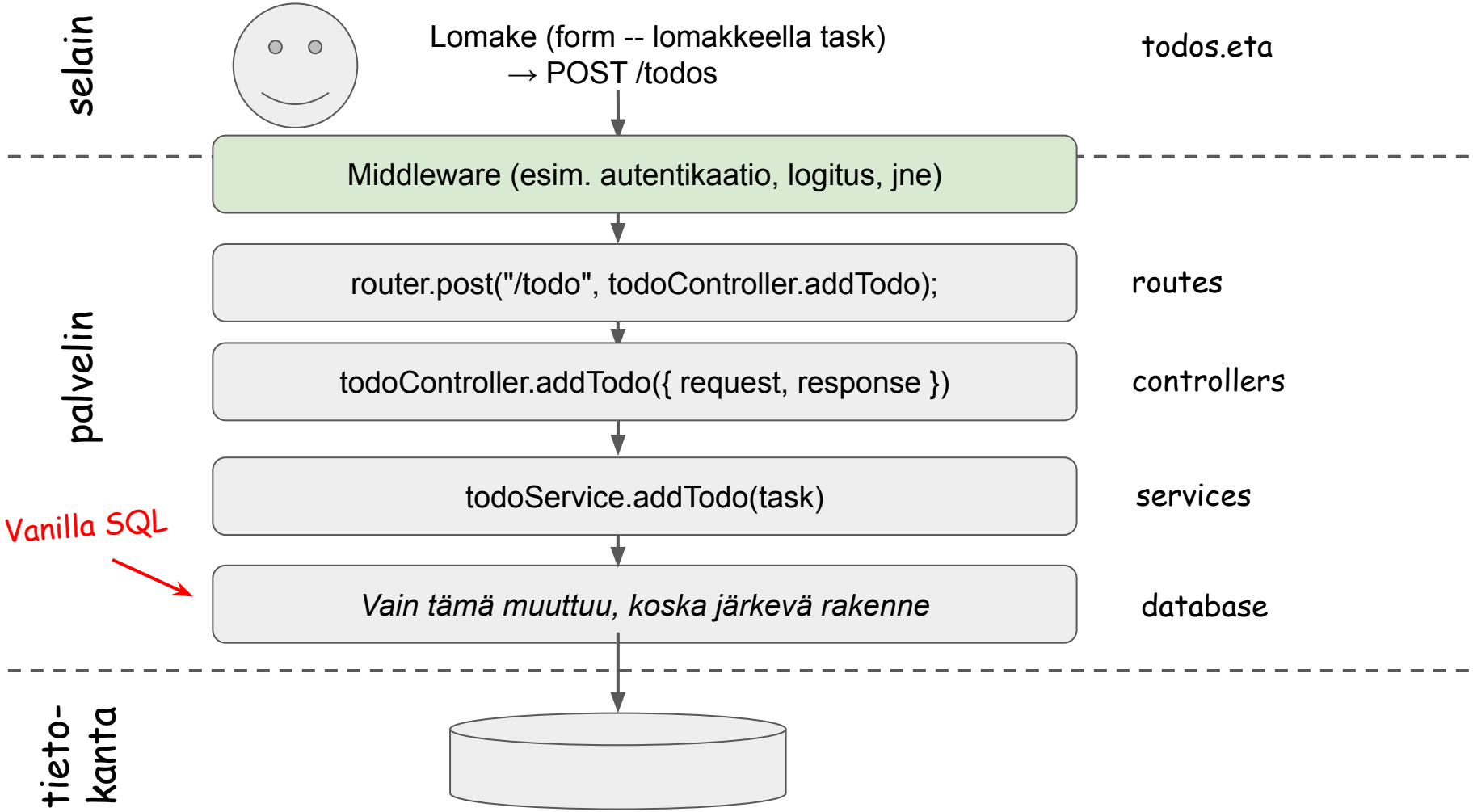
- Haittoja
  - Mahdolliset kirjastokohtaiset konventiot → pakko käyttää tietynlaista ohjelmointityyliä
  - Kyselytoiminnallisuus abstrahoitu → virheiden etsiminen mahdollisesti haastavampaa, N+1 kyselyn ongelma, kyselyiden tehokkuus?, ...

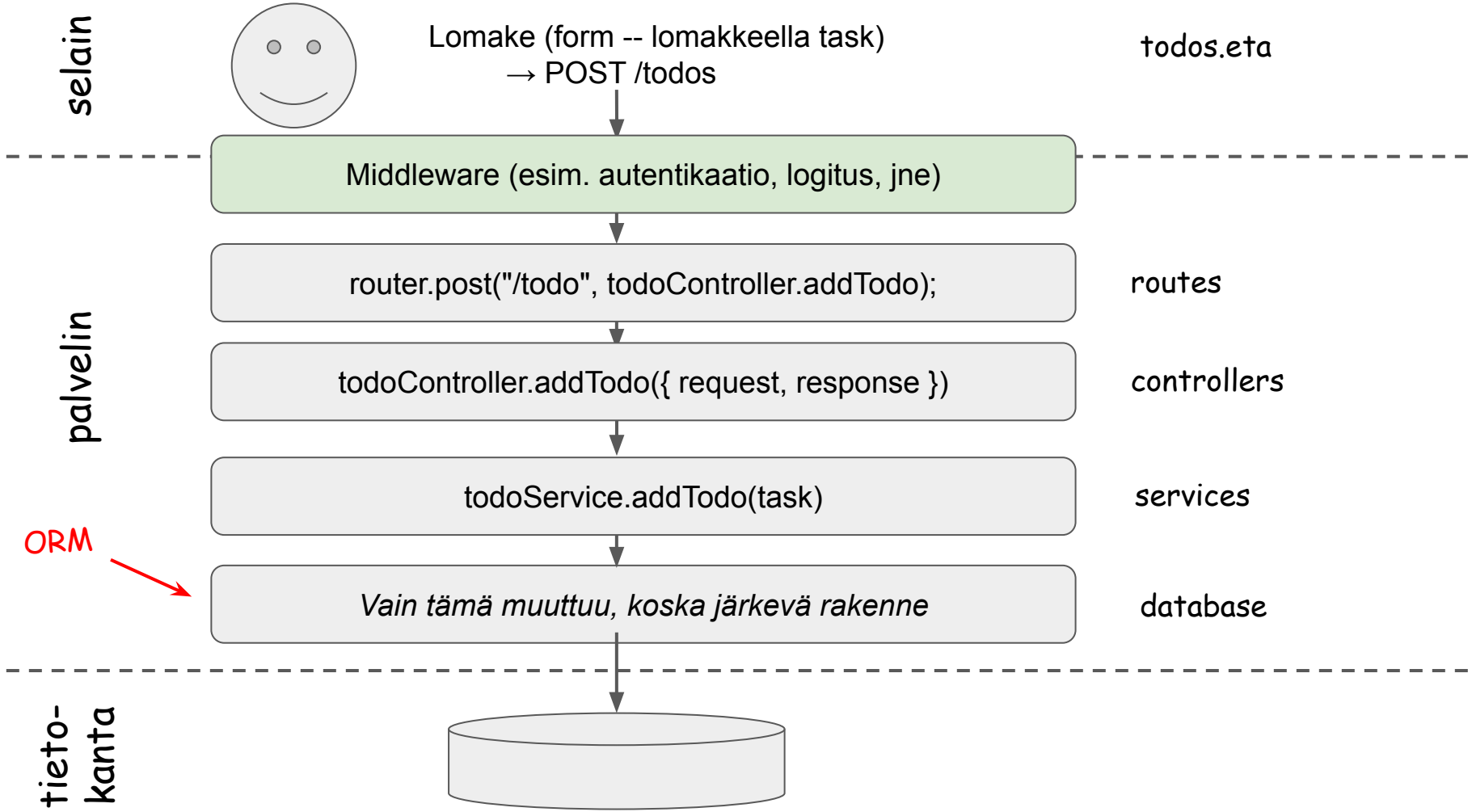
Demo & hands-on - “project-05-todos-orm”

- Muokkaa sovellusta siten, että uutta kokoelmaa lisättäessä kokoelmaan voidaan lisätään myös muistiinpano (kenttä “note” tietokannassa)
  - Muokkaa collections.eta sivua - lisää lomakkeeseen uusi kenttä
  - Muokkaa collectionController.js:stä sopivaa funktiota, ota lähetty arvo talteen
  - Muokkaa collectionService.js:stä sopivaa funktiota, lisää mahdollisuus muistiinpanon lisäämiseen
  - Muokkaa collection.eta sivua - näytä sivulla kokoelmaan liittyvä muistiinpano


selain
Lomake (form -- lomakkeella task)
→ POST /todos
todos.eta
Middleware (esim. autentikaatio, logitus, jne)
palvelin
router.post("/todo", todoController.addTodo);
routes
todoController.addTodo({ request, response })
controllers
todoService.addTodo(task)
services
Vanilla SQL
executeQuery
database
tieto-
kanta


selain
Lomake (form -- lomakkeella task)
→ POST /todos
todos.eta
Middleware (esim. autentikaatio, logitus, jne)
palvelin
router.post("/todo", todoController.addTodo);
routes
todoController.addTodo({ request, response })
controllers
todoService.addTodo(task)
services
ORM
Todo.create
database
tieto-
kanta


selain
Lomake (form -- lomakkeella task)
→ POST /todos
todos.eta
Middleware (esim. autentikaatio, logitus, jne)
palvelin
router.post("/todo", todoController.addTodo);
routes
todoController.addTodo({ request, response })
controllers
todoService.addTodo(task)
services
Vanilla SQL
Vain tämä muuttuu, koska järkevä rakenne
database
tieto-
kanta


selain
Lomake (form -- lomakkeella task)
→ POST /todos
todos.eta
Middleware (esim. autentikaatio, logitus, jne)
palvelin
router.post("/todo", todoController.addTodo);
routes
todoController.addTodo({ request, response })
controllers
todoService.addTodo(task)
services
ORM
Vain tämä muuttuu, koska järkevä rakenne
database
tieto-
kanta

Kerrosarkkitehtuuri

- Tapa jakaa sovellus loogisiin kokonaisuuksiin
  - Auttaa sovelluksen rakenteen → tuttu, toistuva rakenne
  - Auttaa sovelluksen muokkaamista → vaikka osa muuttuu, muut palat pysyvät
  - Auttaa sovelluksen testaamista → kokonaisuuksia voi testata myös erikseen

- Tyypillisesti ainakin:
  - Jonkinlainen tapa esittää näkymä (esim luotu templateista, HTML+Javascript, ...)
  - “Controllers” → Käsittelevät pyynnöt ja palauttavat vastauksen
  - “Services” → Tarjoavat pyyntöihin toiminnallisuuksia kuten tietokannan käsittelyä
  - Jonkinlainen tietokanta-abstraktio (ORMit, “executeQuery”)

Hands-on - “project-05-foods”

- Toteuta pohjaan sovellus, joka mahdollistaa ruokien lisäämisen ja listaamisen
- Sovelluksessa tulee olla lomake, jolla voi lisätä ruokia tietokantaan
  - Jokaisesta ruuasta tallennetaan ruuan nimi
- Sovelluksessa tulee olla listaus tietokannassa olevista ruuista
- Sovelluksessa tulee olla jokaisen ruuan yhteydessä nappi, jolla ruuan voi poistaa tietokannasta
- Sovelluksessa tulee noudattaa kerrosarkkitehtuuria

Kannattaa ottaa mallia aiemmista projekteista -- ei käytetä tässä ORMia.

Kahvitauko?

Kerrosarkkitehtuuri ja näkymän muodostaminen

- Demo
  - “project-05-world-explorer” -- näkymät muodostetaan palvelimella
  - “project-05-world-explorer-frontend” -- näkymät muodostetaan selaimessa

Kerrosarkkitehtuuri ja näkymän muodostaminen

- Demo
  - “project-05-world-explorer” -- näkymät muodostetaan palvelimella
  - “project-05-world-explorer-frontend” -- näkymät muodostetaan selaimessa

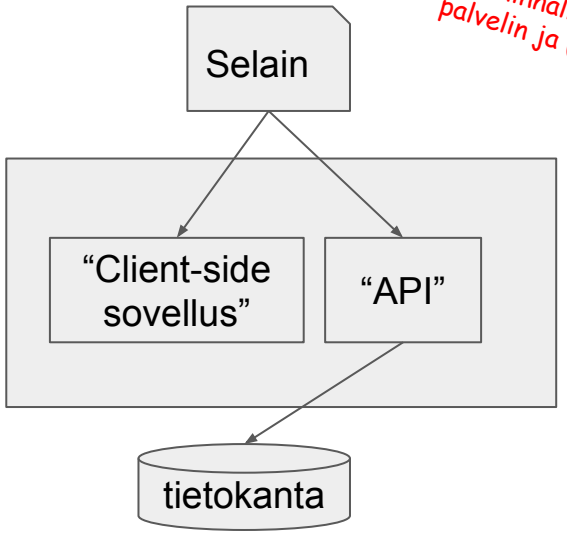
→ *kerrosarkkitehtuuri yhä läsnä,*
muutokset ensisijaisesti siinä,
miten dataa haetaan

Kerrosarkkitehtuuri ja näkymän muodostaminen

- Demo
  - “project-05-world-explorer” -- näkymät muodostetaan palvelimella
  - “project-05-world-explorer-frontend” -- näkymät muodostetaan selaimessa

→ *kerrosarkkitehtuuri yhä läsnä, muutokset ensisijaisesti siinä, miten dataa haetaan*

Toisessa projektissa kaksi palvelinta: (1) client-side -toiminnallisuuden lähettävä palvelin ja (2) API



Hands-on: “World explorer”

- Toteuta mahdollisuus tavaran lisäämiseen pelaajan nykyiseen sijaintiin
- Toteuta mahdollisuus tavaran poistamiseen pelaajan nykyisestä sijainnista
- Advanced: toteuta mahdollisuus tavaroiden kantamiseen
  - Tarvitaan uusi tietokantataulu, joka sisältää pelaajan tavarat
  - Tarvitaan logiikka, joka hakee tietokannasta pelaajan tavarat
  - Tarvitaan muokkaus näkymään, jotta pelaajan tavarat näytetään
  - Tarvitaan logiikka, joka “siirtää” tavaran sijainnista pelaajalle

Voit valita kumpaa projektia muokkaat, suosituksena palvelimella toimiva

Yhteenveto iltapäivästä

\+ *seuraava lähipäivä lyhyesti*