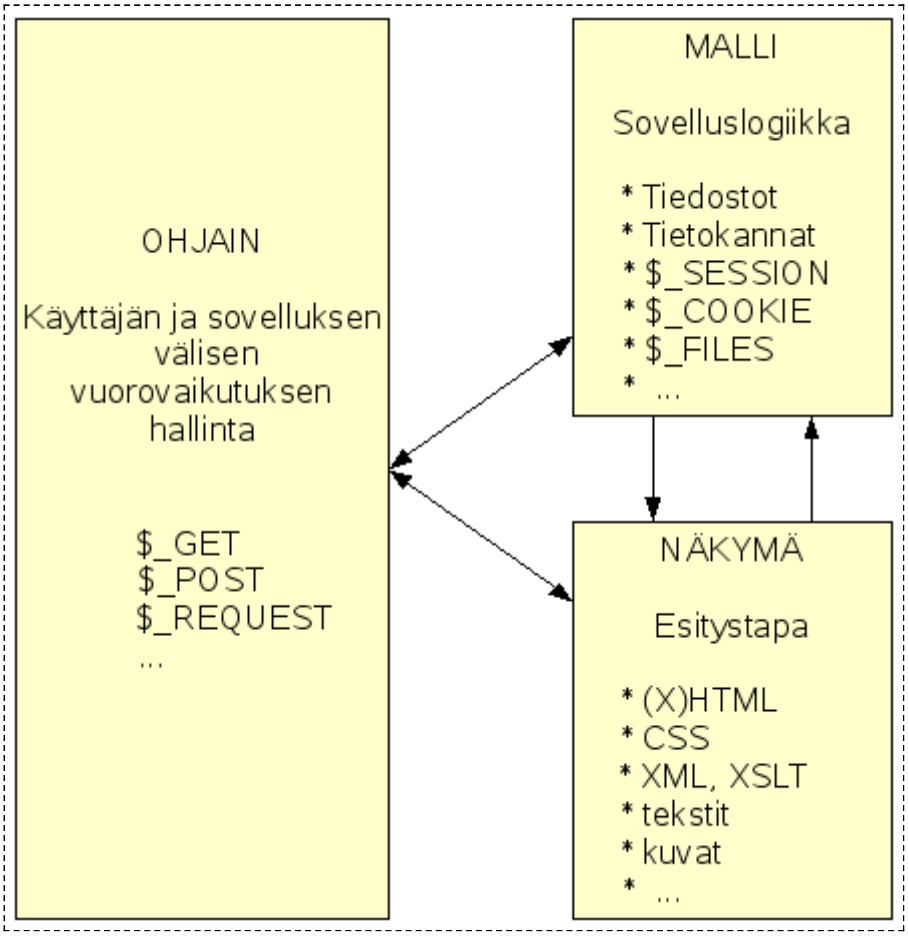
MVC: Model-View-Controller -malli

Malli-Näkymä-Ohjain

- Model-View-Controller (MVC) -malli eli Malli-Näkymä-Ohjain -ohjelmistoarkkitehtuurimalli
- MVC:n perusidea: Käyttöliittymän erottaminen sovellusaluetiedosta
- Suunnittelumalli (desing pattern) tarkoittaa yleistä ja täsmällistä tapaa jonkin usein esiintyvän ongelman ratkaisemiseksi. (Pyörää ei tarvitse keksiä joka kerta uudelleen)
- MVC on suunnittelumallia yleisempi ohjelmistoarkkitehtuurimalli
- MVC sopii sellaisiin vuorovaikutteisiin sovelluksiin, joissa tarvitaan useita erilaisia käyttöliittymän näkymiä
  => Sopii moniin Web-sovelluksiin!

MVC:n osat

MVC-arkkitehtuurissa ohjelma jaetaan kolmeen osaan: malliin, näkymään ja ohjaimeen.



Malli

- Malli on vastuussa sovelluksen sovelluslogiikan toteuttamisesta.
- Malli vastaa sovellusaluekohtaisen tilatiedon tallentamisesta, ylläpidosta ja käsittelystä.
- Ainoastaan malli on yhteydessä tietovarastoon (mm. relaatiotietokanta, tiedostot, istuntomuuttujat)
- Malli ei ole riippuvainen yhdestäkään tietystä näkymästä tai ohjaimesta
- Malli ei koskaan sisällä (X)HTML-merkkausta tai CSS-tyylejä!

Näkymä

- Näkymä on vastuussa käyttöliittymän ulkoasusta ja mallin tietojen esitystavasta käyttöliittymässä.
- Web-sovelluksissa käyttölittymä rakennetaan ensisijaisesti (X)HTML:n pohjalle.
- Näkymä ei koskaan sisällä tietovarastojen (tiedostot, tietokannat, ...) suoraa käsitelyä.

Ohjain

- Ohjain on vastuussa käyttäjän ja sovelluksen välisen vuorovaikutuksen ohjaamisesta

- Ohjain vastaanottaa kaikki käyttäjältä tulevat palvelupyynnöt ja muuttaa sovelluksen tilaa pyynnön mukaisella mallilla ja näkymällä.

MVC: Etuja/Haittoja

Edut

- Useita näkymiä samaan tietosisältöön
- Näkymät helposti muokattavissa ja lisättävissä (ei sisällä sovelluslogiikkaa)
- Mallit "helposti" muokattavissa ja lisättävissä (ei sisällä esitystapalogiikkaa)
- Ohjaimet helposti muokattavissa ja lisättävissä (ei sisällä sovelluslogiikkaa ja esitystapalogiikkaa)

Haitat

- Monimutkaisuus lisääntyy => Ei sovellu pieniin sovelluksiin

---

URI-tunnisteiden uudelleenkirjoitus

MVC-mallin käyttö Web-sovelluksissa johtaa näkymien yksilöintiin HTTP-pyynnön parametreilla. esim:

http://localhost/showuser.php?la=fi&id=havpen

Tämä voi aiheuttaa ongelmia mm.

- hakujärjestelmien (Google) käytössä (eivät indeksoi riittävän laajasti)
- käyttäjille, jotka haluavat hyödyntää suoraan URI-tunnisteita

URI-tunnisteet voidaan uudelleenkirjoittaa esim. Apache web-palvelimen `mod_rewrite`-moduulin avulla niin, että yksittäiset näkymät ovat saatavilla pelkkien hakemistopolkujen avulla, näin aiemmin mainittu URI voisi muuttua esim. muotoon:
http://localhost/users/havpen/

Neljä lähestymistapaa toteuttaa web-sovelluksia

- Kirjoitetaan kaikki ohjelmakoodi itse => Työläs, maksimaalinen joustavuus
- Höydynnetään vahvasti valmiita ohjelmakirjastoja (PasswdLib.phar, PDO, jne) => kehitystyö nopeutuu

- Käytetään sovelluskehyksiä (Laravel, Symphony, Zend, Yii, , jne) => kehitystyö nopeutuu, joustavuus vähenee
- Käytetään/räätälöidään valmissovelluksia (esim WordPress) => kehitystyö nopeutuu, joustavuus vähenee edelleen

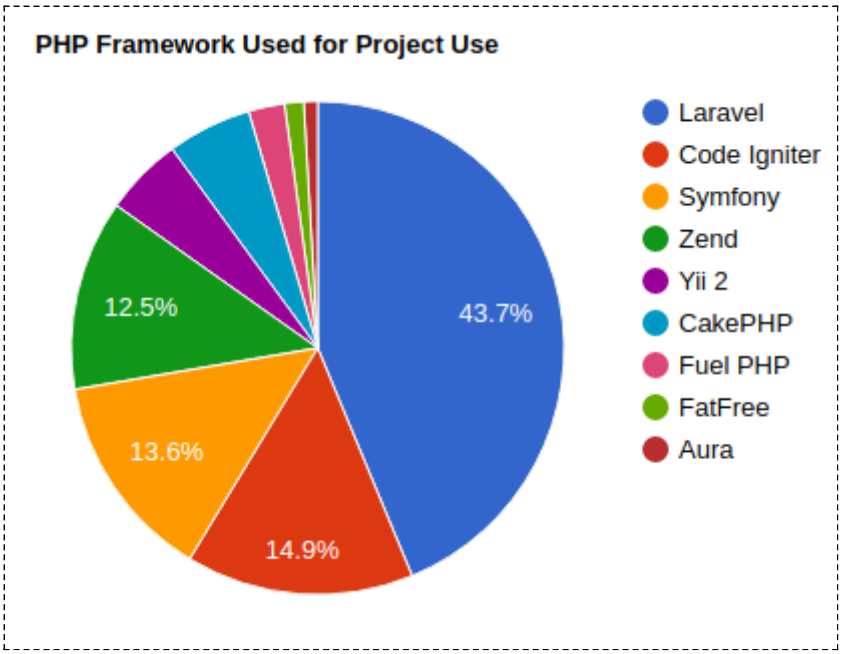
---

Sovelluskehykset (application framework)

- MVC-mallin mukaisen arkkitehtuurin toteuttaminen itse tyhjästä on työlästä (ja yleensä) turhaa
- Sovelluskehyksen (ohjelmistokehys; engl. application framework) tarkoituksena on nopeuttaa sovelluksen kehitystyötä.
- Sovelluskehys tarjoaa valmiita ohjelman osia, joita ei tarvitse kirjoittaa uudelleen ohjelmistokehityksen aikana.
- Sovelluskehystä ei voi käyttää sellaisenaan, vaan oma sovellus rakennetaan kehyksen päälle.
- Tavallisesti sovelluskehys tarjoaa enemmän testattuja ja luotettavampia ratkaisuja yleisiin ongelmiin kuin itse tehden on mahdollista ainakaan nopealla aikataululla tehdä.
- Oliopohjainen kehys on joukko luokkia, jotka on suunniteltu toimimaan yhdessä ja ratkaisemaan jokin rajatun osa-alueen ohjelmointi-ongelma. [http://zds.iki.fi/zds/tekstit/kehykset.shtml]
- Luokkakirjastosta poiketen kehykselle on määritelty myös luokkien suoritusaikaisia keskinäisiä riippuvuuksia. Kehys yleensä sisältää järkevän ja yleisen oletustoteutuksen ratkaisemaansa ongelmaan [http://zds.iki.fi/zds/tekstit/kehykset.shtml]
- Tavanomaisesta luokkakirjastosta tai komponentista sovelluskehykseksen erottaa vuorovaikutuksen ohjaus. Luokkakirjastoa käytettäessä ohjelmoija on itse vastuussa sovelluksen toiminnan ohjaamisesta. Sovelluskehys sen sijaan sisältää apuvälineiden lisäksi myös valmiin mallin vuorovaikutuksesta olioiden välillä. Sovelluskehys ohjaa itse toimintaansa ja kutsuu tarvittaessa ohjelmoijan määrittelemiä luokkia ja rakenteita [http://zds.iki.fi/zds/tekstit/kehykset.shtml]

> Sovelluskehykselle on myös ominaista, että sen soveltuvuusalue on hyvin tarkkaan rajattu, mutta sovellus voidaan kehystä käyttäen laatia hyvinkin pienellä erikoistavan työn määrällä. Siinä missä kirjasto on voitu koota löyhästikin ja sitä käytetään poimien yksittäisiä luokkia avuksi, kehys on tiukempi kokonaisuus ja samalla myös rajoitetumpi. Tämä rajoittuneisuus on kehyksien samalla suurin vahvuus mutta myös huonoin puoli [Mattsson].

PHP-kielellä on toteutettu useita MVC-malliin perustuvia PHP-MVC-sovelluskehyksiä. Mikään sovelluskehys ei ole noussut DeFacto-asemaan, vaikka Laravelin asema on juuri tällä hetkellä (26.9.2019) vahva. Tunnetuimpia ja käytetyimpiä lienevät tällä hetkellä



Lähde: https://coderseye.com/best-php-frameworks-for-web-developers/

Muita vertailuja

- https://www.valuecoders.com/blog/technology-and-apps/top-popular-php-frameworks-web-dev/
- https://raygun.com/blog/top-php-frameworks/