Szakdoga

* planning
* tools
  + github desktop
  + git reset --hard 4a155e5
  + development fallback methods
    - core library
    - STATE – commits
  + .env
  + dependency management
* django
  + why
  + how does it work
  + user management
    - CUser – email based auth.
    - custom user field
    - user manager
    - contrib.aut
    - +reg
  + referenceing foreign key values - <https://stackoverflow.com/questions/1823880/relatedmanager-object-has-no-attribute/1823965>

it means that multiple foreign key can be available. so you have to query or filter them with user.rel\_name.all() or .filter()

* + not saving data in extra storage, always accessing them via userSocialAuth , using img url as source
  + url and worktree structure, hierarchy, project-app conceptions
    - test url
  + templating
  + class-based vs. function based
  + email
    - notifcations
    - verification
    - tokens
    - base 64 encoding
  + security of mails
  + signals, recievers
    - define in signals.py, apps.py, \_\_init\_\_.py
    - usage 1: as custom signal – place dispatch in code
    - usage 2: default model signals
  + social auth
    - facebook developers
    - registering apps
    - test apps
    - limitations (domain, app validation)
    - implementation in settings.py
    - extra fields
    - „extra\_data” field – how to make it work – db field type
    - parsing json
    - extracting and using data
  + rating
    - saving querys with storing num\_ratings and current\_avg, and refreshing them with signals
    - also using many-to-many fields for storing ratings to avoid re-rateing
* admin
  + costumization
  + users and static fields
  + social users
* forntend
  + Django templating
  + file structure
  + workflow
  + automation tools
  + using svg <https://css-tricks.com/using-svg/>
  + svg optimizing
  + svg encoding
* deployment
  + hosting
    - google cloud
    - AWS
    - digital ocean
    - heroku
    - pythonanywhere
    - linode
    - domain
  + setup

**Az alkalmazás**

ezt és ezt várjuk tőle

Mindennek fényében látható, hogy az alkalmazás lelke az authetntikációs rendszer és a kialakított felhasználói profil. Ha ez jól van kialakítva, akkor onnantól arra ráépíthetünk mindenféle funkciót, amit megtervezünk az oldalhoz. Mindezeket Django appokban implementálva hozzáadhatjuk, kivehetjük szabadon, és újra fel is használhatjuk más webalkalmazásokban.

* django-allauth <https://django-allauth.readthedocs.io/en/latest/overview.html>
* default backend – what for, when necessary? Defaultként be van állítva. Ha mi a settingsben hozzáteszünk, akkor azzal felülírjuk a backendek listáját, így oda ezt a defaultot is fel kell tűntetnünk ha használni szeretnénk. Social loginnál pl. szeretnénk, ha nem CSAK social login lenne, hanem mellette a saját loginunk is.
* social login – app, test app
* allauth socialaccount - with js\_sdk or oauth2
* reverse url
* LAMP stack, stacks
* server os
* linux distributions
* container platform – open shift, kubernetes
* deployment methodology -> blue green deployment, canary <https://docs.openshift.com/container-platform/3.3/dev_guide/deployments/deployment_strategies.html>
* CICD tools - <https://dzone.com/articles/what-cicd-tool-should-i-use> közepe felé

**Szerver os**

Bár a Windows ma a legelterjedtebb operációs rendszere pc-re, a legelterjedtebb szerver OS viszont a linux. Köztudottan nagyon sok disztribúciója (változata) él, és érdemes megfontolni melyiket választjuk.

Van egy kifejezés, „LAMP stack” – ami a weboldal backendjének Linux, Apache, MySQL, PHP felépítésére utal. Ez a nagyon sok lehetséges felépítés közül egy, egy népszerű. A mi esetünkben eleve nem PHP-val dolgozunk, hanem Python a backend, viszont azért vetem fel mégis, mert ilyen klasszikus összeállítások útján olvasgatva sokat megtudhatunk, hogy a ma népszerű, jól működő stack-eknek mi az erőssége, és miket használnak gyakran és miért működnek olyan jól.

Amikor dontöttünk, hogy linux alapú szervert szeretnénk használni, akkor ki kell választanunk, milyen disztribúcióját használjuk. (Még egyszer fontos kiemelni, hogy itt szerverhez keresünk operációs rendszert, más szempontok fognak szóbajönni, mintha asztali megoldást keresnénk.) A leggyakrabban felmerülők a Debian, Ubuntu, Fedora, OpenSUSE, CentOS, Arch, Slackware. Sokan személyes preferenciából választanak, gyakran adott, hogy min kell majd dolgozni, de alapvetően az alkalmazásunk tulajdonságaihoz, rendelkezésünkre álló forrásokhoz érdemes mérni.

Az OpenSUSE (korábban SUSE Linux) könnyen telepíthető, telepítéskor az alapvető csomagokat tartalmazza csak, minden mást szabadon rendezhetünk, felkonfigurálhatunk - olyan fejlesztőknek és rendszergazdáknak lett kitalálva leginkább, akik saját szervert szeretnének üzemeltetni, és jól jön, hogy sok mindenben szabad kezet kaphatnak.

A Fedora disztribúcióban ún. rolekit tool-al menedzselhetjük a szerverünket. Szép letisztult felhasználói felület készült hozzá, tartalmaz egy erősnek számító PostgreSQL adatbázis szervert, és kialakításánál fogva az authentikációs folyamatok is igen jól kezelhetők vele. Azonban valamivel haladóbb szintű ismereteket igényel a hatékony működtetése, nem biztos, hogy ezzel érdemes kezdeni.

A mai legnépszerűbb linux szerver disztribúció a debian stable. Az ubuntu is debian alapokra épít.

Az ubuntu szerver egyik kiemelkedő tulajdonsága a felhasználói felülete. Ha mi leggyakrabban ssh kapcslaton, ftp-n, másegyeben keresztül használjuk csak, akkor ennek a tulajdonságnak nem sok hasznát vesszük. Ám ettől függetlenül is nagyon erős és népszerű szerver op. rendszer, akár média szervernek, e-mail szervernek, játék szervernek. Kifejezetten ajánlott azoknak akik újak még a linux világban.

Hasznos lehet ha a tanulásunk elején egy ilyen jól bejáratott, nagy fejlesztői közösséggel rendelkező összeállításon tanuljuk meg a weboldalak üzembehelyezésének folyamatát, ezeket ismerve egyre speciálisabb és komplexebb rendszereket tudunk már majd bátran használni.

<https://www.techradar.com/best/best-linux-server-distro>

///„which linux distribution for servers”