Projekt 1

Aplikacija za dostop do podatkov o cenah crpyto inštrumentov

Projekt boste razvijali v več fazah. Vsaka faza ima svoj cilj in zaključek. Vsaka naslednja faza nadgrajuje predhodno. Pogosto ne boste le dodajali nove kode, ampak boste staro spreminjal/reorganiziral oz. rešil povsem na novo. Temu procesu se reče refakturiranje.

Namen takega postopka je, da se počasi naučite koncepte, brez, da bi bilo preveč stvari na enkrat.

**Na koncu vsake naloge commitaj svoj napredek!** Commite tudi pushaj na github in mi redno pošlji link do repozitorija (da ne iščem vsakič) z obvestilom kaj novega si dodal.

# Osnova projekta

Piševa aplikacijo, ki bo, za začetek prikazovala, podatke o crypto trgih.

Vir podatkov bo Coinbase, saj imajo odprt API (google kaj je to API).

de

Dokumentacijo najdeš tu:  
<https://docs.cloud.coinbase.com/sign-in-with-coinbase/docs/api-prices>  
  
Ponujajo buy, sell in spot price. Predlagam da uporabljaš spot za enkrat.

Za primer uporabe imajo *examples* - uporabi tistega za “Shell” in ga nalimaj v svoj (linux) terminal. Uporablja orodje *curl* (pogooglaj).

API ti vrača podatke enkodirane (encoded) v JSON (google it). To je en izmed *machine readable* formatov (druga pogosta sta XML in CSV).

# Faze

## Faza 1

Osnovna command line interface (CLI) aplikacija. Kot je iz imena razvidno bo to aplikacijo za enkrat možno uporabljati le preko CLI (aka. terminala).

Cilj te faze je izpis trenutne cene poljubnega tokena ter deljenje kode z mano preko github.

### Naloga 1: Github projekt

1. Naredi si račun na github.com (če ga še nimaš, drugač uporabi obstoječega)
2. Ustvari nov repository
   1. Poimenj ga “ip21\_akademija\_projekt1”
   2. Naj bo public, brez gitignore in GNU v3 licenca
3. Na svojem računalniku (predlagam da imaš kakšno mapo Projects, v katero daš vse) *kloniraj* tvoj nov repository

Sedaj imaš okolje v katerem boš razvijal.

Dodaj “console.php” file, ki izpiše “Hello world”. Poganjaš z “php console.php”iz terminala. Moraš biti v pravem folderju.

Commitaj svoj napredek kot “Naloga 1, done”.

### Naloga 2: Izpis podatka o BTC

V console.php napiši PHP kodo, ki bo:

* Prebrala API:  
  <https://api.coinbase.com/v2/prices/BTC-USD/spot>  
  Hint: file\_get\_contents
* Dekodirala JSON v array
* Izpisala informacijo o tem za katere valute gre in ceno

Na pozabit commitat po vsaki nalogi (in pushat!)

### Naloga 3: Izpis poljubne valute proti USD

Kot vidiš imaš v URLju od API podtek o “price pair” (i.e. “BTC-USD”).

* Nauči se kako posredovat argumente CLI aplikaciji v php. Google “php how to accept CLI arguments”.
* Cilj je, da kličeš skripto: “php console.php ETH”

### Naloga 3: Izpis za poljubni pair

* Dodaj še drug argument (ki v “BTC-USD” nadomesti USD) in omogči preverjanje cene poljubnega dvojca.
* Izpist podatkov naj bo uporabniku prijazen (i.e. povsem v human readable formi vse pomembne informacije)
* Če pokličeš console.php brez katerega koli argumenta oz. če je prvi argument “help” izpiši “help text” (i.e. navodila za uporabo)

### Naloga 4: Odpornost na napačne argumente

Tu pridemo v pravo polje programiranja - use cases and edge cases. Kaj so vse možne situacije v katere se lahko spravi user. Vedno pričakuj da bo uporabnik naredil nekaj, kar je bolj “neumno” in nepričakovano kot si ti lahko predstavljaš. Point dobrega programerja je, da pokrije vse te situacije.

* Preveri da je vsak argument smiselne dolžine (najbrž ne le 3, saj so nekateri tokeni daljši. Ampak najbrž pa ne 15 znakov).
* Poskusi kaj se zgodi če zahtevaš token oz. token pair, ki ne obstaja. Zaznaj to napako, ki ti jo vrže API in uporabniku izpiši smiseln error message

### Naloga 5: White-listing argumentov

Na tem API endpontu:  
<https://docs.cloud.coinbase.com/sign-in-with-coinbase/docs/api-currencies>  
  
Imaš seznam vseh valut, ki obstajajo.

* Oba argumenta preveri proti seznamu valut, ki jih coinbase ponuja in pokliči endpoint za cene, le če najdeš oba. Če ne, vrni primeren error.
* Kaj se zgodi če nek pair ne obstaja, kljub temu da sta obe valuti veljavni?

### Naloga 6: Error handeling

Kaj se zgodi, če API nič ne vrne? Recimo 500 error? Ali timeout?

* Preveri ali $json, ki ga dobiš nazaj, sploh ima vsebino
* Preveri ali se je $json pravilno dekodiral (poglej kaj vrne *json\_decode* če dobi invalid json kot input)
* Vrni error če nisi dobil podatkov

Simuliraš tako da “Pokvariš” API klic (i.e.malo ga spremeni, bo API vrnil error).

### Naloga 7: Nov feature - seznam valut

Sedaj začneva širit nabor featurjov. Pogosto se CLI komande uporablja tako da imaš najprej ime komande *console.php* kateremu sledi komanda (npr: “list”, “show”, “add”, … ) tej pa sledijo argumenti.

* Reorganizairaj argumente tako da imaš tri in sicer tako: `console.php price BTC USD`
* Nato dodaj še eno komando - console.php list - ta naj izpiše vse valute v neki berljivi, smiselni obliki.

Malo že razmišljaj kako organizirati kodo.

## Faza 2

Pred začetkom te faze pošlji kodo v review in implementiraj vse popravke.

Namen te faze je, da malo organiziraš kodo na grobo skladno z [MVC](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller).

### Naloga 2.1 Ustvari model

V mapi, v kateri imaš console.php ustvari mapo “lib”, to je dokaj tipična okrajšava za “library”.

Nato ustvari file “lib/model.php” datoteko.

Vanjo prenesi (skopiraj) vse funkcije, ki se direktno ukvarjajo z pridobivanjem podatkov (i.e. funkcije, ki pridobijo seznam podatkov). Notri prenesi tudi getApiData funkcijo na katere te funkcije temeljijo.

Ta file nato vključi v console.php file z requre\_once(). Kako to deluje si preberi v dokumentaciji/online (v tečajih je to najbrž poglavje including files).

### Naloga 2.2 Pravilen model

Model se mora in **sme** ukvarjati le z pridobivanjem in obdelavo podatkov.   
(!)Ugotavljanje kakšne podatke mora pridobit (seznam podprtih ali podatki o ceni) je naloga controllerja.

(!) Izpis podatkov je naloga view-ja (i.e. model ne sme imeti NOBENEGA echa).  
  
Zato preglej svoje funkcije in poskrbi da le vračajo podatke in nikoli nič ne izpisujejo direktno.   
  
Če morajo javit error naj le vrnejo false, controller pa naj nato “Error occurred”.  
  
Naloga 2.3 Ustvari view

View je načeloma edini del kode, ki sme izpisovati karkoli uporabniku. Včasih se za errorje dovoli da te izpiše z controller. Ampak vse ostalo pa mora nujno izpisati view.

Zanjo ustvari novo mapo “libs/views” in file consoleView.php. Razlog za tako poimenovanje se skriva v tem, da bomo kasneje dodali še kakšen drug view (npr.: website[html], email[html] etc.)

V ta file skopiraj funkcijo in file dodaj v console.php z require\_once():

Na tej točki je console.php efektivno postal tvoj controller.

## Faza 3: Testiranje

Ta kratka faza je namenjena temu, da na hitro pokrijemo koncept testiranja.

Cilj faze je priprava fajla “user\_stories.md” (.md je [markdown](https://www.markdownguide.org/basic-syntax/) file, to je enostaven plain text file z posebno sintakso).

Ustvari ta file. Vanj bomo pisali [user stories](https://en.wikipedia.org/wiki/User_story).

User storjiov ni treba komplicirat - to je le seznam stavkov tip “Kot uporabnik želim X, zato napišem Y in dobim Z”.  
  
Več tu: https://www.atlassian.com/agile/project-management/user-stories

### Naloga 3.1

V file napiši vse “user stories” za tvojo aplikacijo. To naj vključuje vsaj vse *pravilne* načine uporabe.

Ko jih popišeš jih tudi vse stestiraj in potrdi, da dobiš pričakovan odziv.

### Naloga 3.2

V file dopiši vse *nepravilne* načine uporabe tvoje aplikacije (manjkajoči ali napačni parametri, parametri v narobnem vrstnem redu itd.).

Ko jih popišeš jih tudi vse stestiraj in potrdi, da dobiš pričakovan odziv.

### Naloga 3.3.

Sedaj pa popiši še vse *absurdne* načine uporabe (npr.: parameter za search ima 1000 znakov itd.).

Ko jih popišeš jih tudi vse stestiraj in potrdi, da dobiš pričakovan odziv.

## Faza 4: Boljša organizacija kode

Trenutno je tvoja kode organizirana le z funkcijami. To je ok, dokler je codebase majhen, vendar hitro postane problematična. Ločevanje po fajlih je dober začetek, vendar je le kozmetičen - ločitve vidi le developer, kakšnega pametnega vpliva na kodo (i.e. omejevanja “hodenja v zelje”) ni.

Tu pride v igro OOP - object orientend programming. Klasično ga učijo kot način za modeliranje naravnega sveta (i.e. Class Animals, katerega deduje class Dogs). Vendar se tako OOP le redko uporablja. Tudi dedovanje je IZREDNO redko.

Večino časa se classi uporabljajo le za [enkapsulacijo kode](https://en.wikipedia.org/wiki/Encapsulation_(computer_programming)) - to je fizično ločitev kode med sabo, zato da preprečiš nepredvideno interakcijo.

### Naloga 4.1

V svojem fajlu za view (i.e. lib/views/consoleView.php) ustvari class ConsoleView.

Funkcije, ki jih uporabljša za izpistovanje prestavi v class kot *public* funkcije.

Nato v console.php na vrhu najprej dodaj file z require\_once, nato pa še instantiraj (“to create a new instance”) ta ConsoleView class v variable $view.

Tam kjer si sedaj klicala funkcijo printList() pa sedaj pokliči $view->printList

### Naloga 4.2

Enako naredi še za model. callApi funkcija naj bo *private* (saj jo smeš klicati le iz drugih funkcij v classu, ne pa tudi direktno).

### Naloga 4.3: Izboljšaj model

Trentuno imaš funckijo *areTheEnterdTagsOnList*, kateri moraš podat list. Kar pomeni da moraš kot uporabnik modela (če si predstavljaš da ga nisi ti pisal) vedeti da moraš najprej list poklicati.

To je nepotrebno, ker to je, po definiciji, “stvar modela”.

* Zato to funkcijo prestavi v model, ima pa naj le en parameter (currency)
* Funkcija naj na začetku pokliče $this->getList() in dobi seznam

Ampak to pomeni, da boš seznam pridobival 2x, kajne? (ker to funkcijo kličeš 2x). Res je!  
  
Caching to the rescue! Tu bova uporabila najbolj primitivno obliko cachinga, s čimer pa bova tudi videla kako pridejo clasi prav:

* Na model class dodaj PRIVATE property (variable) po imenu “listOfCurrencies” in mu defaultuno nastavi vrednost null.
* Nato spremeni funkcijo getList(), da najprej preveri to variablo (listOfCurrencies) in če NI NULL vrne to kar je v spremenljivki, če ne pa pokliče API, rezultat shrani v variablo nato ga pa še vrne.

Tako si spet poneostavil controller (i.e. ne rabipš vrstice 41, klici v vrstici 42 so enostavnejši, le spremenit jih moraš) in dodal caching da se API za seznam nikoli ne pokliče več kot 1x.

### Naloga 4.4 Typing funkcij

Ker smo že v PHP8 land je čas da se naučit tudi typinga funcij.

<https://www.amitmerchant.com/php-type-declarations/>

* Pojdi čez vse funkcije in določi type parametrov in return type

## Faza 5: PhpTheRightWay

PhpTheRightWay je odličen vir informacij o tem kako stvari “bi morale biti”:  
<https://phptherightway.com/>  
  
Predlagam da prebereš poglavja:

* Code Style Guide
* Language Highlists (Preskoči Xdebug)
* Dependency management (preskči PEAR)
* Coding Practices (LE TheBasics in Date and Time)
* Templating (mi2 bova uporabljala twig)

## Faza 6: Prvi dependency

Za začetek si preberi kaj je to “dependency managment”.

<https://phptherightway.com/#dependency_management> (zanima te le composer, PEAR lahko preskočiš, ker je passe)

V PHP uporabljamo orodje, ki se mu reče composer:  
  
<https://getcomposer.org/doc/00-intro.md>  
  
Predlagam da prebereš to stran in “Basic usage”.

### Naloga 6.1: Naloži twig

Naloži twig, znajdi se sam (i.e. sledi navodilom):

[Twig](https://twig.symfony.com/)

Za začetek: Preberi navodila do konca “Basic API Usage” in nato implmentiraj prvi primer (z ArrayLoaderjom). Lahko to narediš v ločenem fajlu. Samo poigraj se, da vidiš če ti rata vzpostavit twig samostojno.

Če ti rata, lahko greš naprej in probaš vzpostavit drugi način (z uporabo template fajlov).

### Naloga 6.2 Vzpostavi template file

To bova lahko narediva skupaj. Ampak lahko poskusiš sam.   
  
Cilj je postavit arhitekturo za templating.   
  
1. Implementiraj uporabo twiga z uporabo template fajlov.

2. Prepiši celoten ConsoleView (ki bo šel nato v smeti) v serijo twig fajlov, ki jih uporabiš v controllerju. Vsaka metoda v ConsoleView sedaj postane svoj file.

3. Twig fajle daj v /views/console/ folder

## Faza 6: Osnovni web prikaz

Prava moč koncepta MVC (model-view-controller) se pokaže takrat, ko imaš več kot en view za enake podatke.

Zato bova sedaj enake podatke (za enkrat le enostavni seznam vseh) izpisala tudi na spletni strani.

### Naloga 6.1: Hello world

Naredi index.php in le naredi echo “Hello world”.

V terminalu (ločenem! Saj bo zaseden!) poskrbi da si v mapi projekta in poženi “*php -S localhost:8000”.*

To postavi webserver (google kaj je to) na port 8000 (če je IP naslov enakovreden “Slovenska ulica 1” je port enakovreden “Nabiralnik Novak”) in gostuje mapo v kateri si ga izvedla. “Index.php” je standardno ime za “če ne veš kateri file v mapi odpret, odpri index”.

Če v brskalniku odpreš <http://localhost:8000> bi se ti moral odpret tvoj Hello world.

### Naloga 6.2: Izpiši podatke

Iz console.php smiselno skopiraj stvari tako, da lahko z var\_dump pokažeš array, ki ga vrne model funkcija, ki vrača seznam vseh valut.

Skopiraj LE in IZKLJUČNO tisto kar nujno potrebuješ! Nič več kot to!

### Naloga 6.3: Dodaj HTML view

Dodaj nov template file (list.html.twig) in vanj dodanj osnovno HTML strukturo strani in izpiši prej pridobljene podatke (uporabi “unoredered list” html element). Lahko dodaš tudi malo CSS da centriraš ipd.

To je to za enkrat. Na web se bova vrnila kasneje.

### Naloga 6.4: Dodaj osnovni CSS

V svoj projekt dodaj style.css file in ga dodaj v svoj webpage (preko HTML meta tagga).

V CSS file dodaj ukaze, ki izbojlšajo izgled:

* HTML prikaz naj ima
  + Naslov (h1)
  + Urejen seznam, vizalno CENTRIRAN.
    - To dosežeš tako, da <*ul>* (unoredered list) tag omejiš na max ~600px, nato pa ta element centriraš (uporabi google da ugotoviš kako).
  + Seznam grafično nadgradi (mogoče boldaj ime?)