

DDD на практике

Предметно-ориентированное проектирование.

Погружение без фанатизма

Тестовое задание

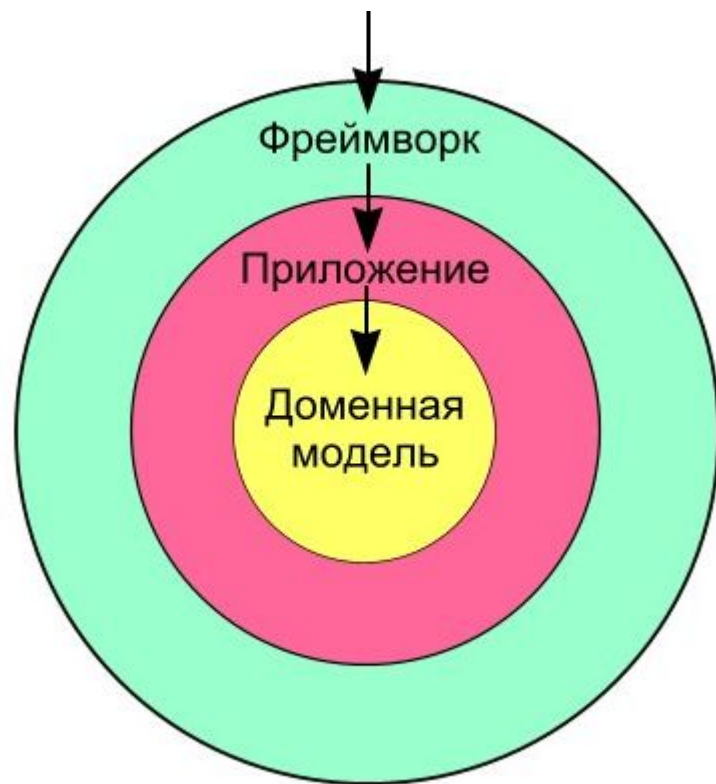
1. Разработать систему позволяющую добавить учетную запись указав логин, ключ апи, секретный ключ. Получить можно на сайте <https://localbitcoins.net>
2. Создать автоматическое обновление баланса аккаунта раз в 60 сек. и обновлять курс валюты раз в 10сек, bitcoin/usd используя формулу расчета $\max(\text{bitstampusd_avg}, \text{bitfinexusd_avg}) * \text{usd_in_rub}$, используя запросы к REST Api. См.документацию апи.
3. Вывести на главной странице, в таблице, список аккаунтов и курсы валют.

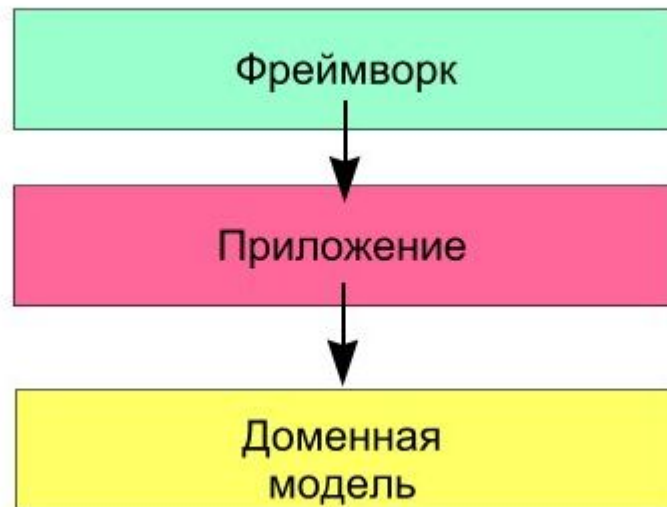
В промышленных приложениях DDD использует ряд шаблонов

1. Value Object
2. Entity
3. Service
4. Module
5. Aggregate
6. Repository
7. Specification
8. Object-relational mapping(ORM)
9. Inversion of Control(IoC)
10. Factories

Буду использовать

1. Value Object
2. Entity
3. Service
4. Module
5. Aggregate
6. Repository
7. Specification
8. Object-relational mapping(ORM)
9. Inversion of Control(IoC)
10. Factories
11. Collection
12. DTO





Этап 1. Определение доменной модели

1. Разработать систему позволяющую **добавить учетную запись указав логин, ключ апи, секретный ключ**. Получить можно на сайте <https://localbitcoins.net>
2. Создать автоматическое **обновление баланса аккаунта** раз в 60 сек. и обновлять **курс валюты** раз в 10сек, bitcoin/usd используя формулу расчета $\max(\text{bitstamp_avg}, \text{bitfinex_avg}) * \text{usd_in_rub}$, используя запросы к REST Api См.документацию апи.
3. Вывести на главной странице в таблице, список **аккаунтов** и историю **курсов валют**.

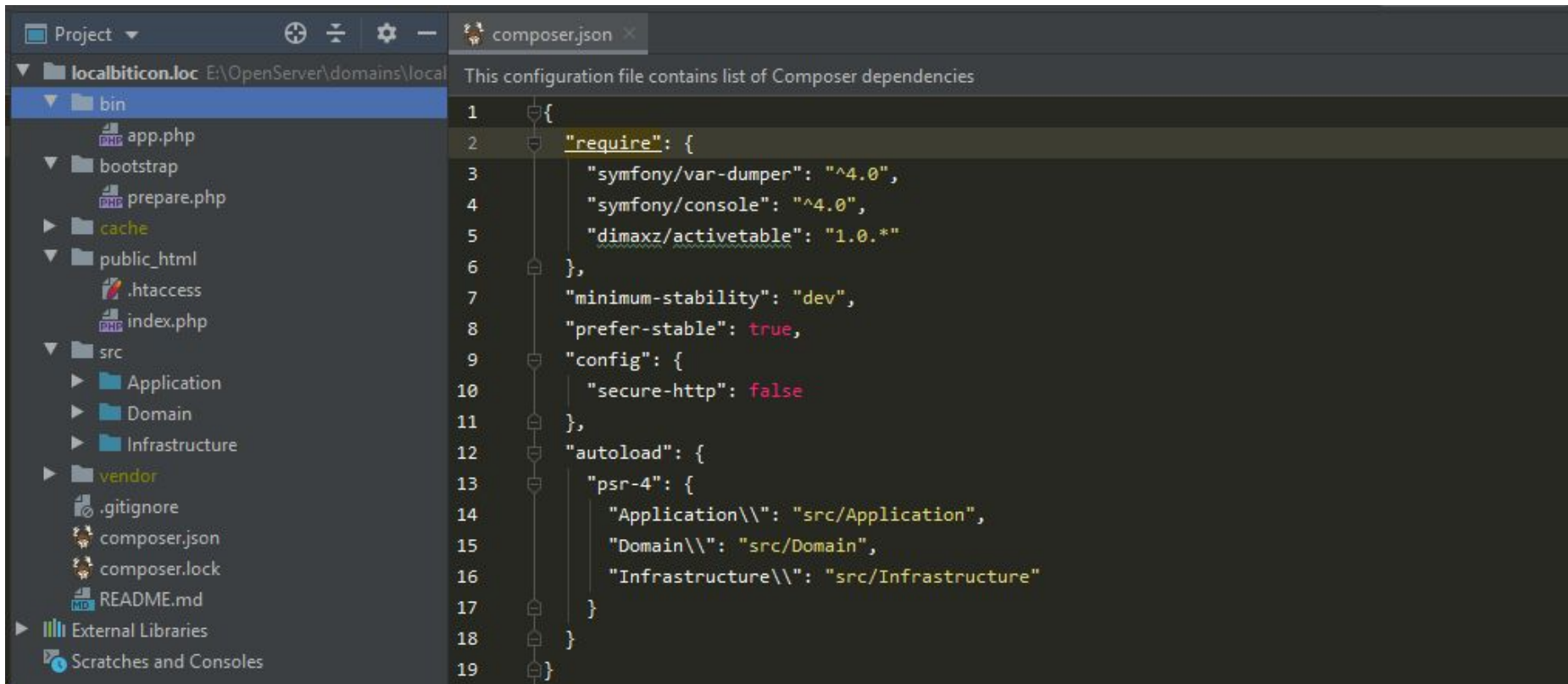
Определение элементов приложения

1. Разработать систему позволяющую добавить учетную запись указав логин, ключ апи, секретный ключ. Получить можно на сайте <https://localbitcoins.net>
2. Создать автоматическое обновление баланса аккаунта раз в 60 сек. и обновлять курс валюты раз в 10сек, bitcoin/usd используя формулу расчета $\max(\text{bitstamp_usd_avg}, \text{bitfinex_usd_avg}) * \text{usd_in_rub}$, используя запросы к REST Api. См.документацию апи.
3. Вывести на главной странице в таблице, список аккаунтов и историю курсов валют.

Определение элементов инфраструктуры

1. Разработать систему позволяющую добавить учетную запись указав логин, ключ апи, секретный ключ. Получить можно на сайте <https://localbitcoins.net>
2. Создать автоматическое обновление баланса аккаунта раз в 60 сек. и обновлять курс валюты раз в 10сек, bitcoin/usd используя формулу расчета $\max(\text{bitstamp_usd_avg}, \text{bitfinex_usd_avg}) * \text{usd_in_rub}$, используя запросы к REST Api. См.документацию апи.
3. Вывести на главной странице в таблице, список аккаунтов и историю курсов валют.

Этап 2. Подготовка. Создадим каркас



The screenshot shows an IDE interface with a project structure on the left and a code editor on the right. The project structure is for a local development environment, with a 'bin' directory highlighted. The code editor displays the 'composer.json' file, which contains a list of Composer dependencies. The dependencies include 'symfony/var-dumper', 'symfony/console', and 'dimaxz/activetable'. The configuration also sets 'minimum-stability' to 'dev', 'prefer-stable' to 'true', and 'secure-http' to 'false'. The 'autoload' section defines the 'psr-4' standard for the project's namespaces.

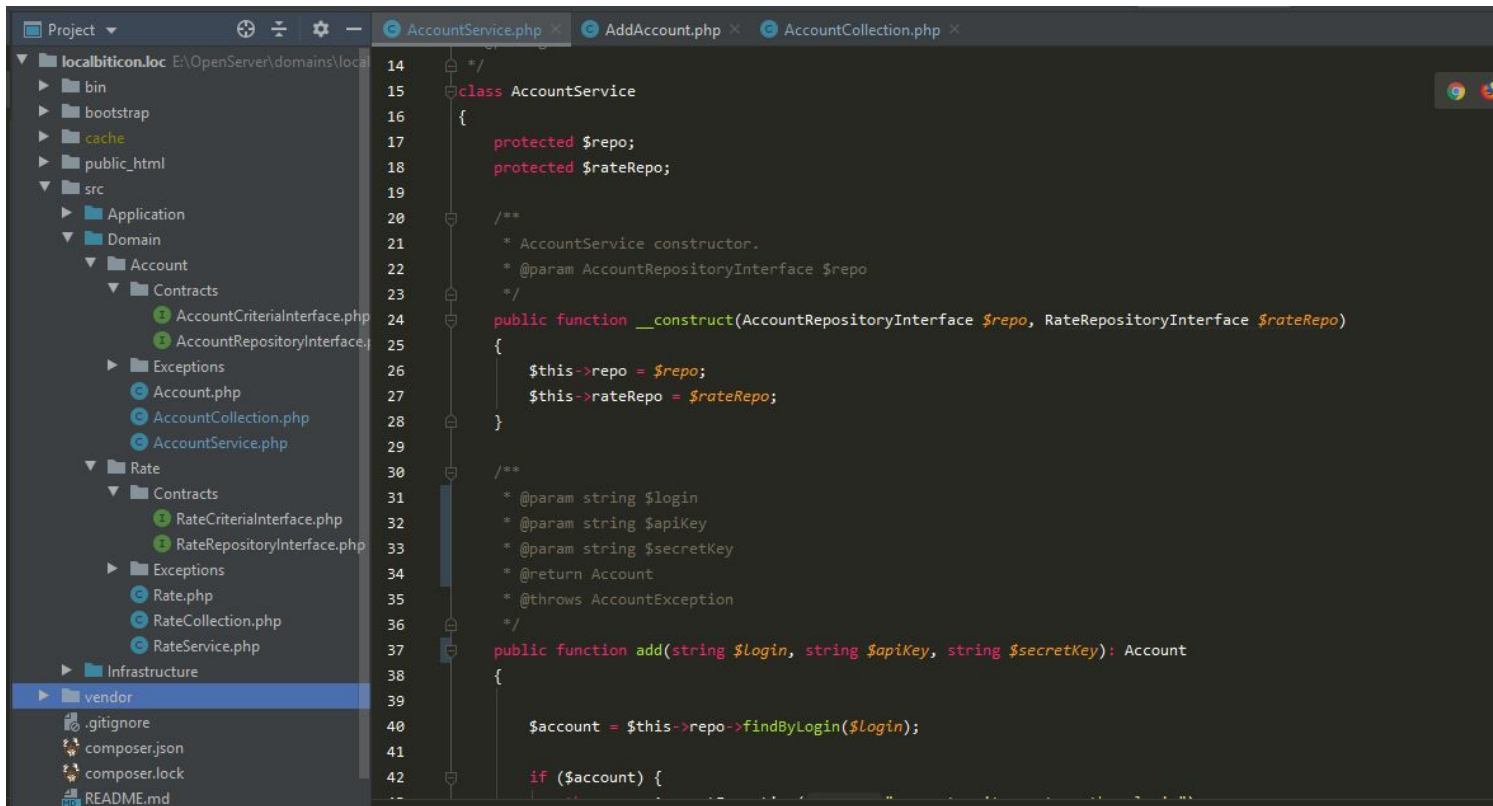
Project Structure:

- localbiticon.loc
 - bin
 - app.php
 - bootstrap
 - prepare.php
 - cache
 - public_html
 - .htaccess
 - index.php
 - src
 - Application
 - Domain
 - Infrastructure
 - vendor
 - .gitignore
 - composer.json
 - composer.lock
 - README.md
- External Libraries
- Scratches and Consoles

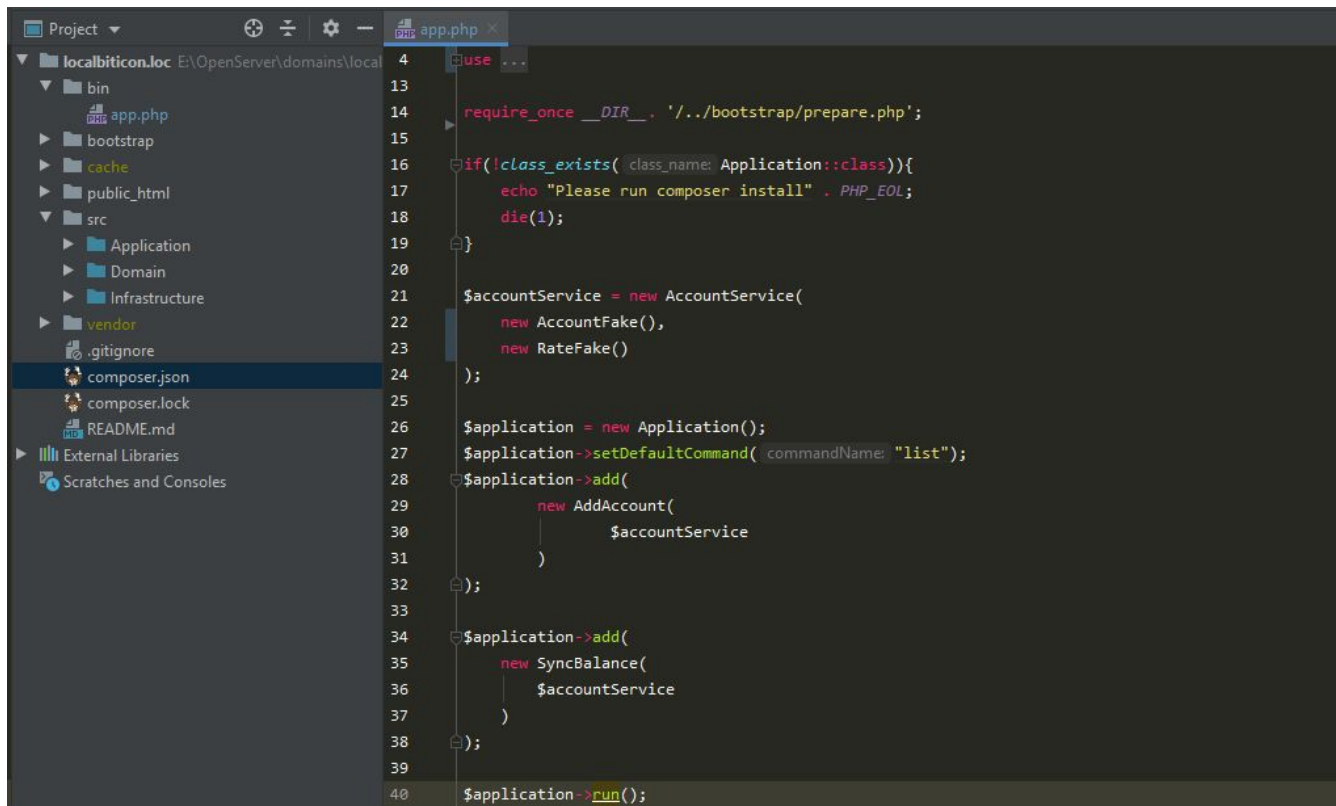
composer.json

```
1 {
2     "require": {
3         "symfony/var-dumper": "^4.0",
4         "symfony/console": "^4.0",
5         "dimaxz/activetable": "1.0.*"
6     },
7     "minimum-stability": "dev",
8     "prefer-stable": true,
9     "config": {
10         "secure-http": false
11     },
12     "autoload": {
13         "psr-4": {
14             "Application\\": "src/Application",
15             "Domain\\": "src/Domain",
16             "Infrastructure\\": "src/Infrastructure"
17         }
18     }
19 }
```

Этап 3. Реализация доменной модели



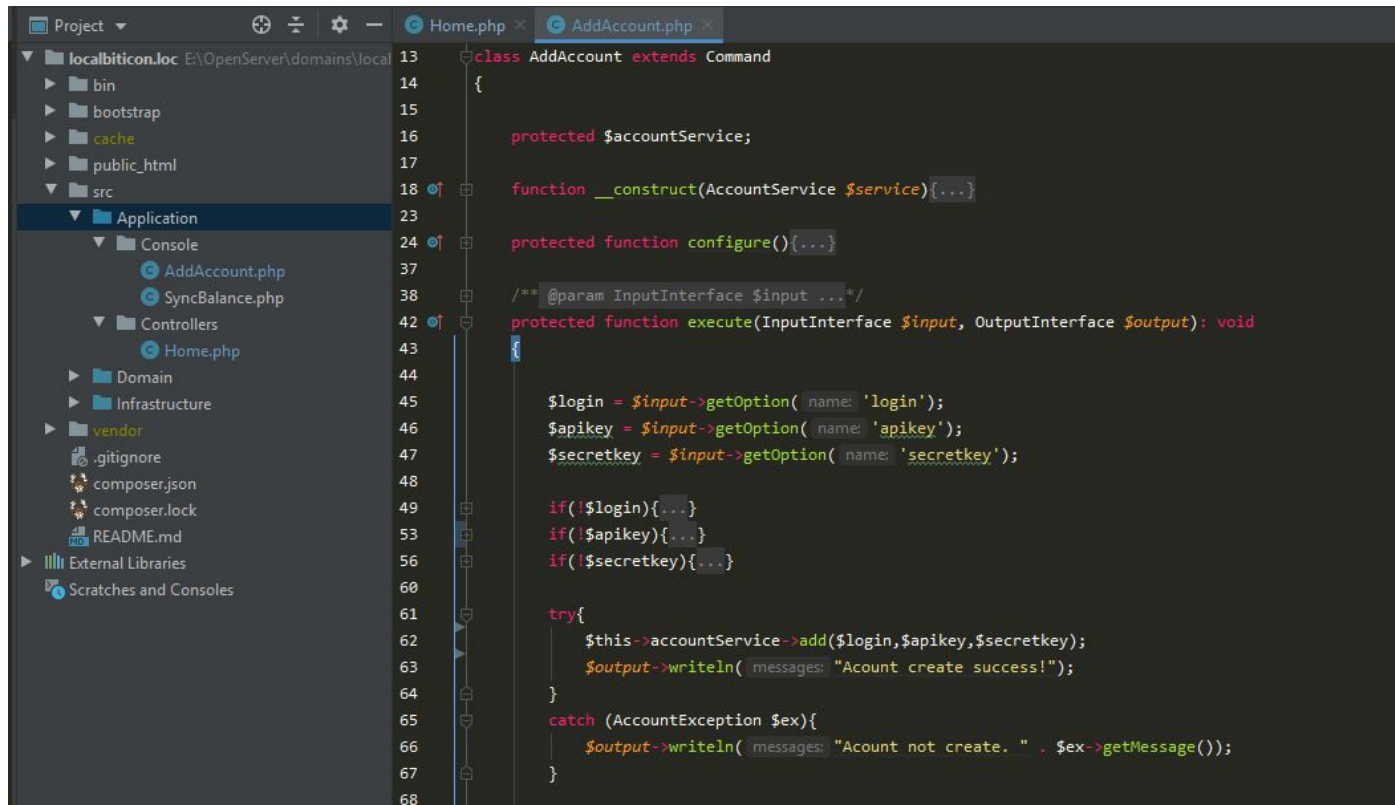
Этап 4. Реализация приложения без инфраструктуры



The screenshot shows an IDE with a project named 'localbiticon.loc' at 'E:\OpenServer\domains\local'. The project structure includes folders like 'bin', 'bootstrap', 'cache', 'public_html', 'src' (containing 'Application', 'Domain', 'Infrastructure'), 'vendor', and files like 'composer.json', 'composer.lock', and 'README.md'. The 'src' folder is expanded, showing the 'Application' folder. The 'app.php' file is open in the editor, showing the following code:

```
4  use ...
13
14  require_once __DIR__ . '/../bootstrap/prepare.php';
15
16  if(!class_exists( class_name: Application::class)){
17      echo "Please run composer install" . PHP_EOL;
18      die(1);
19  }
20
21  $accountService = new AccountService(
22      new AccountFake(),
23      new RateFake()
24  );
25
26  $application = new Application();
27  $application->setDefaultCommand( commandName: "list");
28  $application->add(
29      new AddAccount(
30          $accountService
31      )
32  );
33
34  $application->add(
35      new SyncBalance(
36          $accountService
37      )
38  );
39
40  $application->run();
```

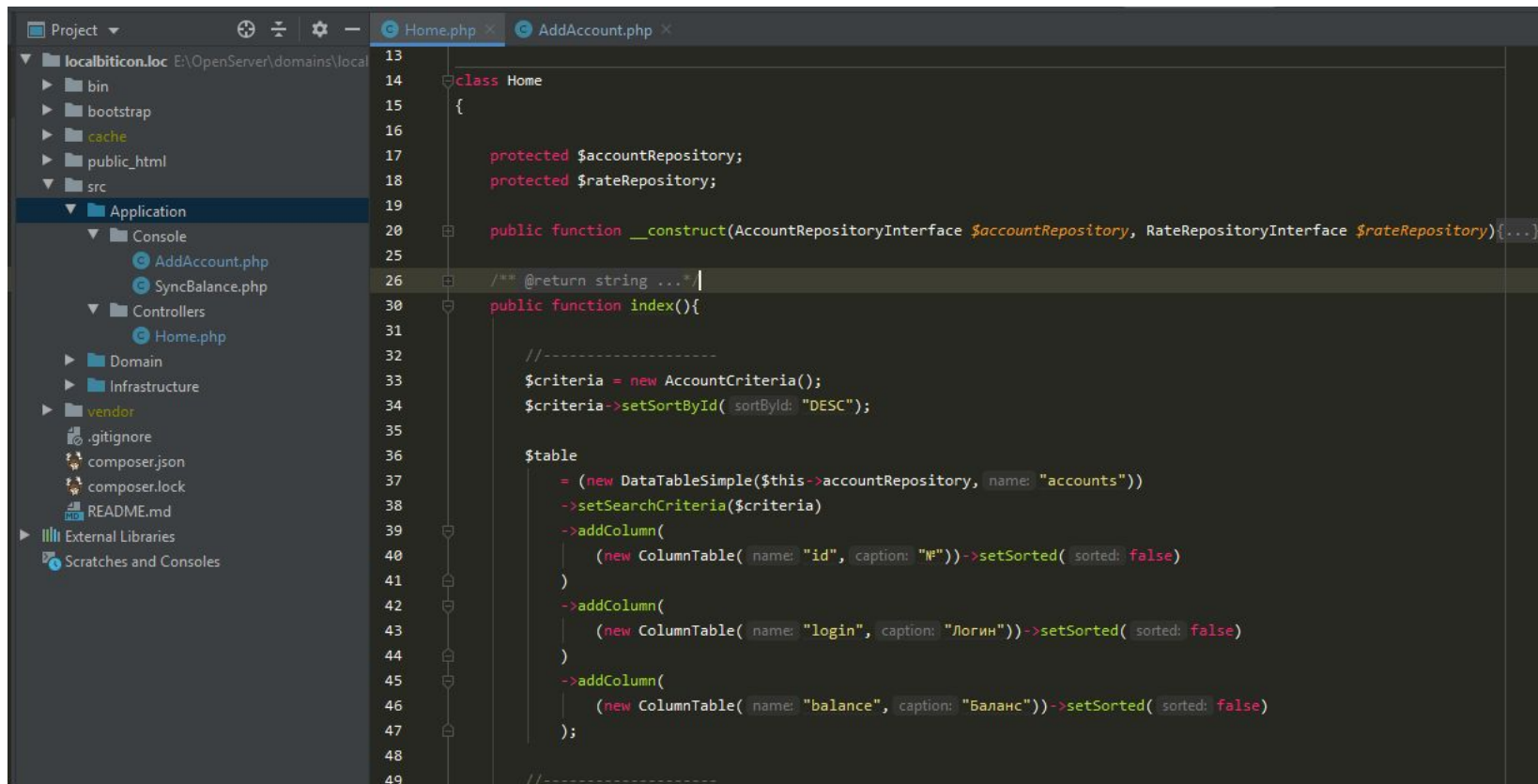
Реализация приложения, консольные команды



The screenshot displays an IDE with a project structure on the left and a code editor on the right. The project structure shows a localbiticon.loc project with folders like bin, bootstrap, cache, public_html, and src. The src folder contains an Application folder, which includes a Console folder with AddAccount.php and SyncBalance.php, and a Controllers folder with Home.php. The code editor shows the implementation of the AddAccount class, which extends the Command class. The class has a protected \$accountService property, a __construct method, a configure method, and an execute method. The execute method takes an InputInterface \$input and an OutputInterface \$output as arguments. It retrieves the login, apikey, and secretkey from the input and uses them to add a new account to the accountService. It then outputs a success message or an error message if an exception is thrown.

```
13 class AddAccount extends Command
14 {
15
16     protected $accountService;
17
18     function __construct(AccountService $service){...}
19
20     protected function configure(){...}
21
22     /** @param InputInterface $input ... */
23     protected function execute(InputInterface $input, OutputInterface $output): void
24     {
25
26         $login = $input->getOption( name: 'login');
27         $apikey = $input->getOption( name: 'apikey');
28         $secretkey = $input->getOption( name: 'secretkey');
29
30         if(!$login){...}
31         if(!$apikey){...}
32         if(!$secretkey){...}
33
34         try{
35             $this->accountService->add($login,$apikey,$secretkey);
36             $output->writeln( messages: "Account create success!");
37         }
38         catch (AccountException $ex){
39             $output->writeln( messages: "Account not create. " . $ex->getMessage());
40         }
41     }
42 }
```

Реализация приложения, контроллеры



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes a 'src' directory with an 'Application' folder, which contains 'Console' and 'Controllers' subfolders. The 'Controllers' folder contains 'Home.php' and 'AddAccount.php'. The 'Home.php' file is open in the editor, showing the following code:

```
13
14 class Home
15 {
16
17     protected $accountRepository;
18     protected $rateRepository;
19
20     public function __construct(AccountRepositoryInterface $accountRepository, RateRepositoryInterface $rateRepository){...}
25
26     /** @return string ... */
30     public function index(){
31
32         //-----
33         $criteria = new AccountCriteria();
34         $criteria->setSortById( sortBy: "DESC");
35
36         $table
37             = (new DataTableSimple($this->accountRepository, name: "accounts"))
38             ->setSearchCriteria($criteria)
39             ->addColumn(
40                 (new ColumnTable( name: "id", caption: "№"))->setSorted( sorted: false)
41             )
42             ->addColumn(
43                 (new ColumnTable( name: "login", caption: "Логин"))->setSorted( sorted: false)
44             )
45             ->addColumn(
46                 (new ColumnTable( name: "balance", caption: "Баланс"))->setSorted( sorted: false)
47             );
48
49         //-----
```

Консольные команды

```
$ ./bin/app.php add-account --help
```

Description:

Add Account

Usage:

add-account [options]

Options:

-l, --login=LOGIN

-k, --apikey=APIKEY

-s, --secretkey=SECRETKEY

-h, --help Display this help message

-q, --quiet Do not output any message

-V, --version Display this application version

--ansi Force ANSI output

--no-ansi Disable ANSI output

-n, --no-interaction Do not ask any interactive question

-v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

```
Admin@LANETZ MINGW64 /e/OpenServer/domains/localbiticon.loc (master)
```

```
$ ./bin/app.php add-account -l Vova -k 1221636 -s ghfhq5323hfdjfds866434b4jfd8
```

Account create success!

Аккаунты

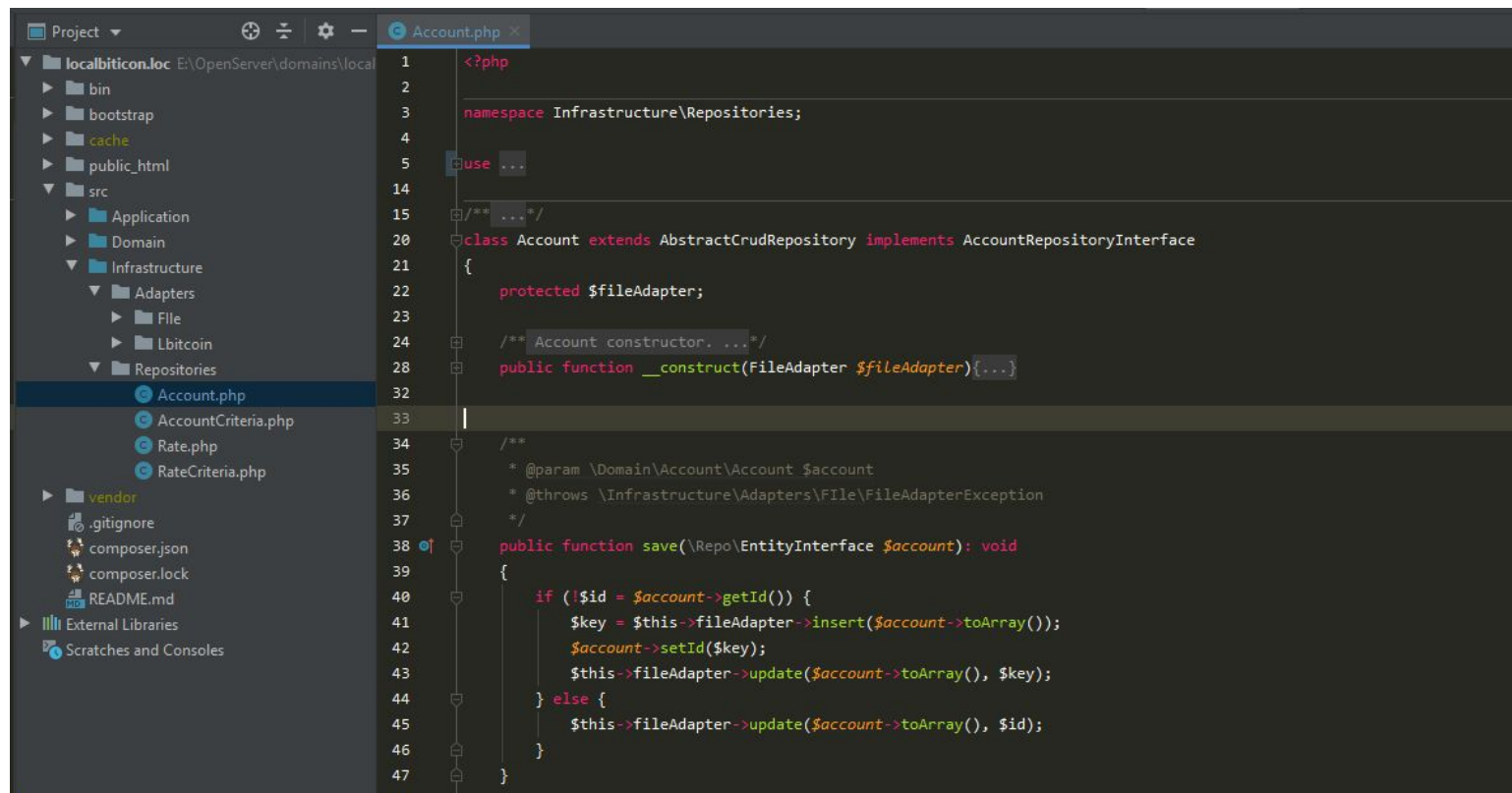
| № | Логин | Баланс |
|------------|-----------|------------|
| 1568240046 | testme123 | 0 |
| 1568236374 | testme | 0 |
| 1568227190 | dimaxz | 0.00005219 |

Курсы

| № | Дата | Логин аккаунта | Основная валюта | Валюта конвертации | Курс |
|------------|------------------|----------------|-----------------|--------------------|-----------|
| 1568238757 | 2019-09-11 21:52 | dimaxz | usd | rub | 666504.28 |
| 1568238747 | 2019-09-11 21:52 | dimaxz | usd | rub | 666504.28 |
| 1568238737 | 2019-09-11 21:52 | dimaxz | usd | rub | 666504.28 |
| 1568238727 | 2019-09-11 21:52 | dimaxz | usd | rub | 666504.28 |
| 1568238717 | 2019-09-11 21:51 | dimaxz | usd | rub | 666504.28 |
| 1568238707 | 2019-09-11 21:51 | dimaxz | usd | rub | 666504.28 |
| 1568238697 | 2019-09-11 21:51 | dimaxz | usd | rub | 666504.28 |
| 1568238687 | 2019-09-11 21:51 | dimaxz | usd | rub | 666504.28 |
| 1568238677 | 2019-09-11 21:51 | dimaxz | usd | rub | 666504.28 |
| 1568238667 | 2019-09-11 21:51 | dimaxz | usd | rub | 666504.28 |
| 1568238657 | 2019-09-11 21:50 | dimaxz | usd | rub | 666504.28 |

Этап 5.
Реализация
интерфейса
И тестов без
инфраструктур
ы.
Демонстрация
заказчику.

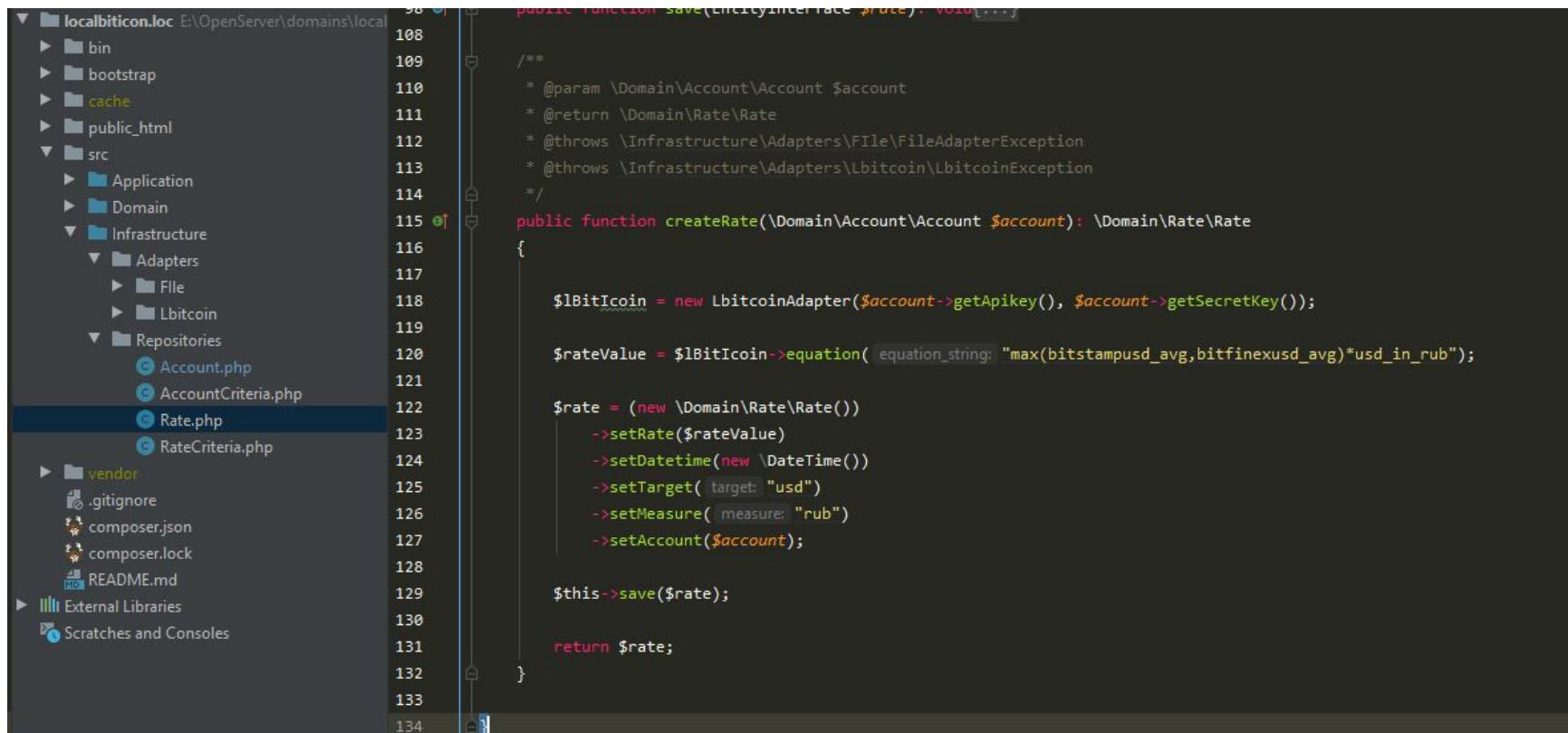
Этап 6. Реализация инфраструктуры, работа с внешними данными



The screenshot displays a code editor with a project explorer on the left and a code editor on the right. The project explorer shows a directory structure for a local project named 'localbiticon.loc'. The 'src' directory contains 'Application', 'Domain', 'Infrastructure', and 'vendor'. The 'Infrastructure' directory contains 'Adapters', 'File', 'Lbitcoin', and 'Repositories'. The 'Repositories' directory contains 'Account.php', 'AccountCriteria.php', 'Rate.php', and 'RateCriteria.php'. The 'Account.php' file is selected, and its code is displayed in the editor. The code implements the 'AccountRepository' class, which extends 'AbstractCrudRepository' and implements 'AccountRepositoryInterface'. It includes a protected property '\$fileAdapter', a constructor, and a 'save' method that uses the 'fileAdapter' to insert or update account data.

```
1 <?php
2
3 namespace Infrastructure\Repositories;
4
5 use ...
6
7
8
9
10
11
12
13
14
15 /** ... */
16
17 class Account extends AbstractCrudRepository implements AccountRepositoryInterface
18 {
19     protected $fileAdapter;
20
21     /** Account constructor. ... */
22     public function __construct(FileAdapter $fileAdapter){...}
23
24
25
26
27
28
29
30
31
32
33
34 /**
35  * @param \Domain\Account\Account $account
36  * @throws \Infrastructure\Adapters\File\FileAdapterException
37  */
38 public function save(\Repo\EntityInterface $account): void
39 {
40     if (!$id = $account->getId()) {
41         $key = $this->fileAdapter->insert($account->toArray());
42         $account->setId($key);
43         $this->fileAdapter->update($account->toArray(), $key);
44     } else {
45         $this->fileAdapter->update($account->toArray(), $id);
46     }
47 }
```

Реализация инфраструктуры. Инфраструктура вторична



```
108 public function save(EntityInterface $rate): void {...}
109
110 /**
111  * @param \Domain\Account\Account $account
112  * @return \Domain\Rate\Rate
113  * @throws \Infrastructure\Adapters\File\FileAdapterException
114  * @throws \Infrastructure\Adapters\Lbitcoin\LbitcoinException
115  */
116 public function createRate(\Domain\Account\Account $account): \Domain\Rate\Rate
117 {
118     $lBitcoin = new LbitcoinAdapter($account->getApiKey(), $account->getSecretKey());
119
120     $rateValue = $lBitcoin->equation( equation_string: "max(bitstampusd_avg,bitfinexusd_avg)*usd_in_rub");
121
122     $rate = (new \Domain\Rate\Rate())
123         ->setRate($rateValue)
124         ->setDatetime(new \DateTime())
125         ->setTarget( target: "usd")
126         ->setMeasure( measure: "rub")
127         ->setAccount($account);
128
129     $this->save($rate);
130
131     return $rate;
132 }
133
134
```

Итог. Чего мы должны избегать в DDD

1. Строить приложение только вокруг данных

Итог. Чего мы должны избегать в DDD

1. Строить приложение только вокруг данных
2. Говорить о деталях реализации

Итог. Чего мы должны избегать в DDD

1. Строить приложение только вокруг данных
2. Говорить о деталях реализации
3. Использовать технический шум

ССЫЛКИ

<https://github.com/dimaxz/localbitcoins-ddd>

<https://www.google.ru/search?q=DDD>