

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

УТВЕРЖДАЮ

Зав.кафедрой,

доцент, к. ф.-м. н.

_____ С. В. Миронов

ОТЧЕТ О ПРАКТИКЕ

студента 3 курса 351 группы факультета КНиИТ

Устюшина Богдана Антоновича

вид практики: производственная

кафедра: математической кибернетики и компьютерных наук

курс: 3

семестр: 6

продолжительность: 4 нед., с 01.08.2024 г. по 28.08.2024 г.

Руководитель практики от университета,

ст. преп., к. ф.-м. н.

М. И. Сафрончик

Руководитель практики от организации (учреждения, предприятия),

доцент, к. ф.-м. н.

С. В. Миронов

Тема практики: «Курс разработки Backend Web-приложения с DevOps принципами»

ВВЕДЕНИЕ

Практика проходила на базе предприятия ООО «ПрофСофт» и заключалась в прохождении курса по Backend- и DevOps-разработке.

В рамках курса велась над учебным проектом «Анекдоты», в рамках которого показывались лучшие практики в разработке серверной части приложений, а также техники по поддержке и настройке CI/CD на проекте.

В процессе разработки использовались язык программирования PHP и объектно-реляционная система управления базами данных PostgreSQL. Также в рамках работы над DevOps задачами активно использовалась YAML-нотация для написания настроечных файлов для pipeline на GitLab и написания выполняемых в рамках автоматизации работ.

Целью практики была разработка backend-части приложения, а также автоматизация его деплоя на GitLab.

В рамках производственной практики должны были быть решены следующие задачи:

1. Построение схемы базы данных
2. Создание базовых CRUD для backend-приложения
3. Создание авторизации на основе концепции access- и refresh-токенов
4. Создание docker-compose и gitlab-ci файлов для настройки контейнеризации и автоматизации запуска приложения

В процессе выполнения данных задач будет рассмотрено использование лучших практик разработки веб-приложений, а также современные подходы к организации рабочего процесса с акцентом на безопасность и масштабируемость приложения.

1 Описание проекта

Современная веб-разработка является одной из самых динамично развивающихся областей информационных технологий. С каждым годом всё большее количество компаний и организаций переносит свои сервисы и системы в онлайн-среду, что требует создания надежных, масштабируемых и безопасных веб-приложений. Особое внимание уделяется разработке серверной части (Backend), которая обеспечивает связь с базами данных, обработку запросов пользователей, а также взаимодействие с различными внешними системами. Веб-приложения, использующие язык программирования PHP и фреймворк Symfony, предоставляют широкие возможности для создания мощных и гибких систем, особенно в сочетании с такими современными технологиями, как контейнеризация с Docker и использование веб-сервера Nginx. Это делает тему разработки серверной части веб-приложений крайне актуальной в контексте повышения производительности, гибкости и безопасности веб-систем.

В данной работе будет рассматриваться процесс разработки серверной части веб-приложения для обмена анекдотами на базе PHP и фреймворка Symfony. В процессе работы будет рассмотрено, как с помощью Docker создать контейнеризированное окружение для разработки и развертывания приложения, как настроить веб-сервер Nginx для обработки запросов, а также как тестировать API с помощью Postman и документировать его с использованием Swagger. Также будут освещены основные принципы работы с архитектурой MVC на Symfony и взаимодействие с базами данных через ORM Doctrine. Особое внимание будет уделено выбору и настройке средств разработки, таких как PHPStorm, а также сравнению PHP с другими популярными языками для создания Backend-приложений. Важной частью работы станет анализ производительности и безопасности разрабатываемого приложения в контейнеризированной среде.

Целью производственной практики является приобретение навыков разработки серверной части веб-приложений с использованием современных инструментов и технологий, таких как PHP, Symfony, Docker, Nginx, Postman и Swagger. Практика направлена на углубленное изучение архитектуры веб-приложений, принципов работы серверной части и интеграции различных инструментов для разработки, тестирования и развертывания приложений.

Кроме того, значительное внимание будет уделено изучению

особенностей контейнеризации и управления окружением разработки с помощью Docker, что позволит улучшить навыки работы с современными DevOps-технологиями.

2 Выполненные в рамках проекта задачи

2.1 Построение схемы базы данных

2.1.1 Постановка задачи

Построить схему базы данных для данной предметной области (анекдоты) и представить её в виде ER-диаграммы.

2.1.2 Решение

Для начала опишем структуру таблиц, которые использовались для построения изображения, а затем предоставим SQL-запросы для их создания.

Структура таблиц:

- Анекдот — сущность использовалась для хранения информации об анекдоте. Сущность включала следующие поля:
 1. id (Primary Key): идентификатор.
 2. title: заголовок.
 3. text: текст анекдота.
 4. category: категория.
 5. author_id (Foreign Key): ссылка на пользователя.
- User — сущность, использующаяся для хранения информации о пользователе. Сущность включала следующие поля:
 1. id (Primary Key): идентификатор пользователя.
 2. surname: фамилия.
 3. name: имя.
 4. patronym: отчество.
 5. email: электронная почта.
- Mark — сущность, использующаяся для хранения информации об оценке, которую пользователь мог поставить анекдоту. Сущность включала следующие поля:
 1. user_id (Primary Key, Foreign Key): ссылка на пользователя.
 2. anec_id (Primary Key, Foreign Key): Ссылка на анекдот.
 3. value: Оценка.
- Code — сущность, использующаяся для хранения информации о коде подтверждения email пользователя при регистрации.
 1. id (Primary Key): идентификатор.
 2. code: код.

3. user_id (Foreign Key): ссылка на пользователя.

4. expired_at: время истечения срока действия.

```
1  -- Таблица User
2  CREATE TABLE User (
3      id SERIAL PRIMARY KEY,
4      sur VARCHAR(100) NOT NULL,
5      name VARCHAR(100) NOT NULL,
6      patr VARCHAR(100),
7      email VARCHAR(255) UNIQUE NOT NULL
8  );
9
10 -- Таблица Anecdote
11 CREATE TABLE Anecdote (
12     id SERIAL PRIMARY KEY,
13     title VARCHAR(255) NOT NULL,
14     text TEXT NOT NULL,
15     category VARCHAR(100),
16     author_id INT REFERENCES User(id) ON DELETE CASCADE
17 );
18
19 -- Таблица Mark
20 CREATE TABLE Mark (
21     user_id INT REFERENCES User(id) ON DELETE CASCADE,
22     anec_id INT REFERENCES Anecdote(id) ON DELETE CASCADE,
23     value INT NOT NULL CHECK (value >= 1 AND value <= 5),
24     PRIMARY KEY (user_id, anec_id)
25 );
26
27 -- Таблица Code
28 CREATE TABLE Code (
29     id SERIAL PRIMARY KEY,
30     code VARCHAR(100) NOT NULL,
31     user_id INT REFERENCES User(id) ON DELETE CASCADE,
32     expired_at TIMESTAMP NOT NULL
```

33);

Для создания схемы базы данных использовался как язык SQL, так и ORM-модели.

ORM (Object-Relational Mapping) — это технология, которая позволяет разработчикам работать с базами данных, используя объектно-ориентированный подход. ORM автоматически преобразует данные из базы данных (реляционные таблицы) в объекты языка программирования и наоборот.

ORM-модель — это объектно-ориентированное представление таблицы базы данных. Каждая таблица в базе данных соответствует классу в коде, а строки таблицы представлены как экземпляры этого класса.

В нашем случае Doctrine сгенерировала следующую миграцию, которую впоследствии применили к БД:

```
1 public function getDescription(): string
2 {
3     return 'Creating first database schema';
4 }
5
6 public function up(Schema $schema): void
7 {
8     // this up() migration is auto-generated, please modify it to
    ↳ your needs
9     $this->addSql('CREATE SEQUENCE anecdote_id_seq INCREMENT BY 1
    ↳ MINVALUE 1 START 1');
10    $this->addSql('CREATE SEQUENCE "user_id_seq" INCREMENT BY 1
    ↳ MINVALUE 1 START 1');
11    $this->addSql('CREATE TABLE anecdote (id INT NOT NULL,
    ↳ author_id INT NOT NULL, title VARCHAR(255) NOT NULL, text
    ↳ VARCHAR(255) NOT NULL, category VARCHAR(127) NOT NULL, PRIMARY
    ↳ KEY(id))');
12    $this->addSql('CREATE INDEX IDX_A5051EEC69CCBE9A ON anecdote
    ↳ (author_id)');
```



```

13      $this->addSql('CREATE TABLE code (id INT NOT NULL, code
    ↳ VARCHAR(255) NOT NULL, user_id_id INT NOT NULL, expired_at
    ↳ TIMESTAMP(0) WITHOUT TIME ZONE NOT NULL, PRIMARY KEY(id,
    ↳ code))');
14      $this->addSql('CREATE INDEX IDX_771530989D86650F ON code
    ↳ (user_id_id)');
15      $this->addSql('COMMENT ON COLUMN code.expired_at IS
    ↳ \'(DC2Type:datetime_immutable)\')');
16      $this->addSql('CREATE TABLE mark (user_id_id INT NOT NULL,
    ↳ anecdote_id_id INT NOT NULL, value INT NOT NULL, PRIMARY
    ↳ KEY(user_id_id, anecdote_id_id))');
17      $this->addSql('CREATE INDEX IDX_6674F2719D86650F ON mark
    ↳ (user_id_id)');
18      $this->addSql('CREATE INDEX IDX_6674F271A347EF68 ON mark
    ↳ (anecdote_id_id)');
19      $this->addSql('CREATE TABLE "user" (id INT NOT NULL, surname
    ↳ VARCHAR(255) NOT NULL, name VARCHAR(255) NOT NULL, patronymic
    ↳ VARCHAR(255) DEFAULT NULL, email VARCHAR(255) NOT NULL,
    ↳ PRIMARY KEY(id))');
20      $this->addSql('ALTER TABLE anecdote ADD CONSTRAINT
    ↳ FK_A5051EEC69CCBE9A FOREIGN KEY (author_id_id) REFERENCES
    ↳ "user" (id) NOT DEFERRABLE INITIALLY IMMEDIATE');
21      $this->addSql('ALTER TABLE code ADD CONSTRAINT
    ↳ FK_771530989D86650F FOREIGN KEY (user_id_id) REFERENCES "user"
    ↳ (id) NOT DEFERRABLE INITIALLY IMMEDIATE');
22      $this->addSql('ALTER TABLE mark ADD CONSTRAINT
    ↳ FK_6674F2719D86650F FOREIGN KEY (user_id_id) REFERENCES "user"
    ↳ (id) NOT DEFERRABLE INITIALLY IMMEDIATE');
23      $this->addSql('ALTER TABLE mark ADD CONSTRAINT
    ↳ FK_6674F271A347EF68 FOREIGN KEY (anecdote_id_id) REFERENCES
    ↳ anecdote (id) NOT DEFERRABLE INITIALLY IMMEDIATE');
24  }

```

Таким образом, была создана схема базы данных.

2.2 Создание базовых CRUD для Backend-приложения

2.2.1 Постановка задачи

Построить CRUD (API) PHP-приложения на фреймворке Symfony для функционирования Backend-приложения.

2.2.2 Решение

Для решения данной задачи требовалось:

1. Создать DTO с помощью инструментов PHP-фреймворка Symfony
2. Создать класс-контроллер для сущности Anecdote
3. Создать бизнес-логику для указанных endpoint из вышеописанного контроллера

Рассмотрим пример на одной из сущностей (анекдоте): остальные реализовались по аналогичному сценарию.

Создание DTO для запроса:

```
1 class AnecdoteBaseRequestDTO
2 {
3     public function __construct(
4         #[Assert\NotNull(groups: ['register'])]
5         #[Assert\Type(type: 'string', groups: ['register',
6 ↪ 'edit'])]
7         #[Assert\Length(max: 127, groups: ['register', 'edit'])]
8         public ?string $title = null,
9
10        #[Assert\NotNull(groups: ['register'])]
11        #[Assert\Type(type: 'string', groups: ['register',
12 ↪ 'edit'])]
13        public ?string $text = null,
14
15        #[Assert\NotNull(groups: ['register'])]
16        #[Assert\Type(type: 'string', groups: ['register',
17 ↪ 'edit'])]
18        public ?bool $category = null,
19    ) { }
20 }
```

Создание DTO для ответа:

```
1 class AnecdoteBaseResponseDTO
2 {
3     public int $id;
4     public string $title;
5     public string $text;
6     public string $category;
7
8     public function __construct(Anecdote $anecdote)
9     {
10         $this->id = $anecdote->getId();
11         $this->title = $anecdote->getTitle();
12         $this->text = $anecdote->getText();
13         $this->category = $anecdote->getCategory();
14     }
15 }
```

Создание контроллера для сущности anecdote с относительным URI anecdote:

```
1 #[Route(path: '/anecdote')]
2 class AnecdoteController extends AbstractController
3 {
4     public function __construct(
5         private readonly ValidatorService $validator,
6         private readonly SerializerInterface $serializer,
7     ) { }
8     #[Route(path: '', name: 'apiGetAnecdoteList', methods:
9     ↪ Request::METHOD_GET)]
10     public function getAnecdoteList(AnecdoteService
11     ↪ $anecdoteService): JsonResponse
12     {
13         return $this->json(
14             data: $anecdoteService->getAnecdoteList(),
15             status: Response::HTTP_OK,
```

```

14         );
15     }
16     #[Route(path: '', name: 'apiCreateAnecdote', methods:
↪ Request::METHOD_POST)]
17     public function createUser(Request $request, AnecdoteService
↪ $anecdoteService): JsonResponse
18     {
19         $data =
↪ $this->serializer->deserialize($request->getContent(),
↪ AnecdoteBaseRequestDTO::class, 'json');
20         $this->validator->validate(body: $data, groupsBody:
↪ ['register']);
21         return $this->json(
22             data: $anecdoteService->createAnecdote($data),
23             status: Response::HTTP_CREATED,
24         );
25     }
26     #[Route(path: '/{id<\d+>}', name: 'apiEditAnecdote', methods:
↪ Request::METHOD_PATCH)]
27     public function editAnecdote(
28         Anecdote $id,
29         Request $request,
30         AnecdoteService $anecdoteService,
31     ): JsonResponse
32     {
33         $data =
↪ $this->serializer->deserialize($request->getContent(),
↪ AnecdoteBaseRequestDTO::class, 'json');
34         $this->validator->validate(body: $data, groupsBody:
↪ ['edit']);
35
36         return $this->json(
37             data: $anecdoteService->editAnecdote($id, $data),
38             status: Response::HTTP_CREATED,

```

```

39         );
40     }
41     #[Route(path: '/{id<\d+>}', name: 'apiDeleteAnecdote',
↪ methods: Request::METHOD_DELETE)]
42     public function deleteUser(Anecdote $id, AnecdoteService
↪ $anecdoteService): JsonResponse
43     {
44         $anecdoteService->deleteAnecdote($id);
45
46         return $this->json(
47             data: [],
48             status: Response::HTTP_NO_CONTENT,
49         );
50     }
51 }

```

В данном коде представлены основные операции с сущностью Anecdote: создание, обновление, чтение, удаление. Это реализуется с помощью вызова у переменной \$anecdoteService методов create, edit и прочих.

Сама же переменная \$anecdoteService является объектом класса AnecdoteService, в котором хранится вся бизнес-логика методов, связанных с сущностью Anecdote.

Рассмотрим один из методов: их логика достаточно тривиальна:

```

1 public function editAnecdote(Anecdote $anecdote,
↪ AnecdoteBaseRequestDTO $DTO): AnecdoteBaseResponseDTO
2 {
3     if ($title = $DTO->title) {
4         $anecdote->setTitle($title);
5     }
6     if ($text = $DTO->text) {
7         $anecdote->setText($text);
8     }
9     if ($category = $DTO->category) {
10        $anecdote->setCategory($category);

```

```
11     }
12     $this->entityManager->flush();
13     return new AnecdoteBaseResponseDTO($anecdote);
14 }
```

Таким образом, при последующем использовании кода были выявлены следующие преимущества данного подхода к решению задачи:

1. DTO позволяет отделить внутренние модели приложения от структуры данных, передаваемых клиенту, что упрощает форматирование ответов и защиту чувствительных данных.
2. Контроллеры обеспечивают чистую организацию кода, выступая посредниками между бизнес-логикой и клиентами API, обрабатывая запросы, вызовы сервисов и формирование ответов.

2.3 Создание авторизации на основе концепции access- и refresh-токенов

2.3.1 Постановка задачи

Создать механизм авторизации с помощью access- и refresh-токенов.

2.3.2 access- и refresh-токены

Использование access и refresh токенов — это подход для безопасной и удобной аутентификации пользователей, часто применяемый в системах с использованием JWT (JSON Web Tokens). Этот метод позволяет минимизировать риски, связанные с компрометацией токенов, и улучшить пользовательский опыт за счёт автоматического обновления сессии.

Access-токен — короткоживущий токен, содержащий информацию об аутентификации пользователя (например, ID, роли и права доступа). Используется для выполнения запросов к защищённым ресурсам API. Имеет короткий срок действия (например, 15 минут), что снижает последствия его утечки.

Refresh-токен — долгоживущий токен, используемый только для получения нового access-токена. Не используется напрямую для доступа к API. Хранится в более защищённом месте (например, в HTTP-only cookies), чтобы минимизировать риск его утечки.

Алгоритм работы системы:

1. Пользователь аутентифицируется (например, с помощью логина и пароля).
2. Сервер выдаёт:
 - а) Access-токен для доступа к ресурсам API.
 - б) Refresh-токен для продления действия сессии.
3. Клиент отправляет access-токен в каждом запросе к API (обычно в заголовке Authorization: Bearer <token>).
4. Если access-токен истёк, клиент использует refresh-токен, чтобы получить новый access-токен через специальный API-эндпоинт.
5. Если refresh-токен истёк, пользователь должен пройти повторную аутентификацию.

2.3.3 Решение

Для выполнения задачи требуется выполнить следующее:

1. Создать сущность Device и добавить её в базу данных. Указать В ней два токена - access и refresh, у каждого своё время жизни.
2. Реализовать проверку на время жизни токена в ApiAuthenticator.
3. Реализовать метод обновления времени жизни access токена.
4. Добавить функционал отправки письма на почту при регистрации с помощью MailerService

2.4 Создание docker-compose и gitlab-ci файлов для настройки контейнеризации и автоматизации запуска приложения

2.4.1 Постановка задачи

2.4.2 Решение

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы были изучены современные инструменты и технологии для разработки серверной части веб-приложений. В частности, были рассмотрены особенности работы с языком PHP и фреймворком Symfony, а также разработана архитектура приложения на основе MVC.

Настроено контейнеризированное окружение с использованием Docker и веб-сервера Nginx для обеспечения эффективной и стабильной работы приложения. Особое внимание уделялось тестированию RESTful API с помощью Postman и документированию его с использованием Swagger. Также проведен

анализ преимуществ и недостатков применяемых технологий в контексте разработки масштабируемых и безопасных веб-приложений.

Перспективы применения результатов данной работы достаточно широки.

Во-первых, разработанное приложение может быть использовано как основа для дальнейшего развития и расширения функциональности, например, для добавления новых типов контента или внедрения системы рекомендаций на основе предпочтений пользователей.

Во-вторых, контейнеризация с использованием Docker позволяет легко масштабировать приложение и развертывать его в различных окружениях, что важно для гибкости и мобильности современных веб-сервисов.

Применение Nginx как веб-сервера повышает производительность и надежность работы приложения при обработке большого количества запросов, что открывает возможности для использования данного решения в высоконагруженных системах. Кроме того, навыки тестирования и документирования API с Postman и Swagger могут быть применены в разработке других проектов, требующих четкой и структурированной документации интерфейсов.

Таким образом, созданный проект достигнул поставленных задач школы ProfSoft, цель производственной практики была достигнута, а все поставленные в ходе практики задачи решены [1].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 *Bayer, M.* Alembic: A database migrations tool for sqlalchemy [Электронный ресурс]. — <https://alembic.sqlalchemy.org/>. — (Дата обращения 26.09.2024) Загл. с экр. Яз. англ.

ПРИЛОЖЕНИЕ А

Создание авторизации

```
1  class SecurityService
2  {
3      private const string SUBJECT = 'Код авторизации';
4
5      private const string ACCESS_TOKEN_LIFETIME = '+10
↪ minutes';
6      private const string REFRESH_TOKEN_LIFETIME = '+90 days';
7
8      public function __construct(
9          #[Autowire(service: YandexMailerService::class)]
10         private MailerServiceInterface $mailer,
11         protected EntityManagerInterface $entityManager,
12     ) { }
13     public function sendCode(LoginDTO $DTO): void
14     {
15         $code = new Code();
16         if
↪ (! $this->entityManager->getRepository(User::class)->findOneByEmail($
↪ instanceof User) {
17             throw new ApiException(
18                 message: "Пользователь по указанному email не
↪ найден",
19                 status: Response::HTTP_NOT_FOUND,
20             );
21         }
22         $code
23             ->setEmail($DTO->email);
24
25         $this->entityManager->persist($code);
26         $this->entityManager->flush();
27         $this->mailer->send(self::SUBJECT, $code->getCode(),
↪ (array)$DTO->email);
```

```

28     }
29
30     public function verifyCode(LoginDTO $DTO): array
31     {
32         $code =
↪     $this->entityManager->getRepository(Code::class)->findOneBy([
33             'code' => $DTO->code,
34             'email' => $DTO->email,
35             'status' => CodeStatus::ACTIVE->value,
36         ]);
37
38         if (!$code instanceof Code) {
39             throw new ApiException(
40                 'Неверный код авторизации',
41                 status: Response::HTTP_UNAUTHORIZED,
42             );
43         }
44         if ($code->getExpiredAt() < new \DateTime()) {
45             $code
46                 ->setStatus(CodeStatus::EXPIRED->value);
47             $this->entityManager->flush();
48             throw new ApiException(
49                 'Код авторизации истек',
50                 status: Response::HTTP_UNAUTHORIZED,
51             );
52         }
53         $owner =
↪     $this->entityManager->getRepository(User::class)->findOneBy([
54             'email' => $DTO->email,
55         ]);
56
57         $device = (new Device())
58             ->setOwner($owner)

```

```

59         ->setTokenExpiresAt((new
↪ \DateTime())->modify(self::ACCESS_TOKEN_LIFETIME))
60         ->setRefreshTokenExpiresAt((new
↪ \DateTime())->modify(self::REFRESH_TOKEN_LIFETIME));
61
62         $code
63         ->setStatus(CodeStatus::INACTIVE->value);
64
65         $this->entityManager->persist($device);
66         $this->entityManager->flush();
67
68         return [
69             'token' => $device->getToken(),
70             'refreshToken' => $device->getRefreshToken(),
71         ];
72     }
73
74     public function logout(string $apikey): void
75     {
76         $device =
↪ $this->entityManager->getRepository(Device::class)->findOneBy([
77             'apikey' => $apikey,
78         ]);
79         $device->setStatus(DeviceStatus::INACTIVE->value);
80         $this->entityManager->flush();
81     }
82
83     public function refresh(?string $refreshToken): array {
84         $device =
↪ $this->entityManager->getRepository(Device::class)->findOneBy([
85             'refreshToken' => $refreshToken,
86             'status' => DeviceStatus::ACTIVE->value,
87         ]);
88

```

```

89         if (!$device instanceof Device) {
90             throw new ApiException(
91                 message: 'Некорректный refresh токен',
92                 status: Response::HTTP_UNAUTHORIZED,
93             );
94         }
95
96         if ($device->getRefreshTokenExpiresAt() < new
↵ \DateTime()) {
97             $device->setStatus(DeviceStatus::EXPIRED->value);
98             $this->entityManager->flush();
99
100            throw new ApiException(
101                message: 'Refresh токен истёк',
102                status: Response::HTTP_UNAUTHORIZED,
103            );
104        }
105
106        $device
107            ->setToken($this->generateToken())
108            ->setTokenExpiresAt((new
↵ \DateTime())->modify(self::ACCESS_TOKEN_LIFETIME))
109            ->setRefreshToken($this->generateToken());
110
111        $this->entityManager->flush();
112
113        return [
114            'token' => $device->getToken(),
115            'refreshToken' => $device->getRefreshToken(),
116        ];
117    }
118
119    public function generateToken(): string {
120        return md5(random_int(100000, 999999) . microtime());
121    }

```

