**Описание объекта тестирования**

Объектом тестирования является web приложение для сдачи (публикации) и аренды (обработки) апартаментов в различных городах России. Приложение реализовано на языке PHP с использованием сторонних библиотек jQuery (для реализации клиентской логики), Compass (для реализации клиентского интерфейса) и Yii2 (для реализации серверной логики). Приложение спроектировано и реализовано для функционирования на серверах под управлением ОС Windows и Unix.

Основные функции

Основные функции объекта тестирования, предназначенные для использования конечным пользователем:

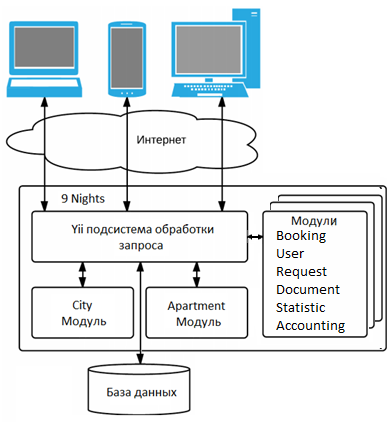
1. публикация апартаментов в общем хранилище для хранения и последующей сдачи другим пользователям;
2. аренда уже опубликованных апартаментов пользователями системы;
3. модерация от лица администратора системы опубликованных пользователями апартаментов;
4. регистрация новых пользователей системы;

Ограничения

Перечисленные выше функции должны быть доступны всему семейству браузеров на движке WebKit наименьшей версии за 2016 год.

**Архитектура системы**

Высокоуровневая архитектура



* **Интернет** - среда передачи данных
* **Yii подсистема** – обеспечивает обработку запроса, роутинг запроса на требуемый модуль и взаимодействие модулей с ресурсами системы «9 Nights»
* **City Module** – реализует функционал для работы с городами в системе «9 Nights»
* **Apartment Module** – реализует функционал для работы с апартаментами в системе «9 Nights»
* **Booking Module** – реализует функционал для работы с бронированием в системе «9 Nights»
* **User Module** – реализует функционал для работы с аккаунтами пользователей в системе «9 Nights»
* **Request Module** – реализует функционал для работы с различными заявками от пользователей и партнеров в системе «9 Nights»
* **Document Module** – реализует функционал для работы с различного вида документами в системе «9 Nights»
* **Statistic Module** – реализует функционал для ведения статистики системы «9 Nights»
* **Accounting Module** – реализует функционал для ведения бизнес-учета партнеров в системе «9 Nights»

Схема взаимодействия подсистем

Запрос

Yii подсистема

Router

City

Модуль

Apartment

Модуль

База данных

Data transfer

Подсистемы

Далее рассматриваются функции, которые выполняют отдельные подсистемы.

* **Yii подсистема** - выполняет функции обработки запросов и составления ответов для клиентов. Состоит из набора PHP сценариев, каждый сценарий соответствует определенному API запросу (html, json, xml). Является основным модулем, обеспечивающим все функциональные требования системы. Не будет подвержен тестированию, так как тестирование проводилось много раз и его результаты можно изучить на сайте разработчиков.
* **City модуль** – реализует функционал обработки данных и действий по отношению к городам в системе «9 Nights». Будет подвержен тестированию, так как обеспечивает основной функционал и входит в рамки рассматриваемого задания.
* **Apartment модуль** – реализует функционал обработки данных и действий по отношению к апартаментам в системе «9 Nights». Не будет подвержен тестированию так как не входит в рамки рассматриваемой области и не взаимодействует с другими подсистемами.

Взаимодействие подсистем

Далее рассматривается характер взаимодействия между отдельными модулями.

* **Взаимодействие подсистемы Yii и City модуля** - происходит обмен данными городов. Клиент через Yii подсистему передает данные для запроса требуемого города. City модуль, получив эти данные, обрабатывает их, производя поиск требуемого города и его данных и в ответ, через подсистему Yii, отправляет клиенту данные в виде html страницы. Передача происходит через внутренние структуры данных (array, object, variables).
* **Взаимодействие подсистемы Yii и Apartment модуля** - происходит обмен данными апартаментов. Клиент через Yii подсистему передает данные для запроса требуемых апартаментов или данные для изменения уже существующих. Apartment модуль, получив эти данные, обрабатывает их, производя требуемое действие с апартаментами. В ответ, через подсистему Yii, отправляет клиенту данные в виде html страницы. Передача происходит через внутренние структуры данных (array, object, variables).
* **Взаимодействие подсистемы Yii и Базы данных** - происходит обмен данными через подсистему Yii. Табличные данные, в ходе прохождения через подсистему Yii, преобразуются в объект модели данных PHP, что позволяет более гибко их обрабатывать. Изменяя объект данных, yii автоматически изменяет их копию в базе данных, что упраздняет заботу разработчика о самой базе данных. Передача происходит через внутренние структуры данных (object).

Список функций City модуля

Ниже приведен список функций для взаимодействия с подсистемой городов. Все указанные функции реализованы в модуле City.

* string **tableName** (void) – название таблицы баз данных для привязки объектов данных.
* array **rules** (void) – список правил валидации полей объекта данных при взаимодействии с базой данных.
* array **attributeLabels** (void) – список полей объекта данных и их синонимы в таблице базы данных.
* array **getPublic** (int $cityId, array $order) – получение всех общедоступных апартаментов в виде массива объектов.
  + *cityId* – Положительное целое. Идентификатор города.
  + *order* – Ассоциативный массив сортировки, где ключами служат имена столбов таблицы базы данных, а значением – направление сортировки семейства констант SORT\_\*
* array **getLogoImages** (bool $withoutDefault, int $forceId, bool $withoutTemps) – получение фотографий апартаментов в виде массива ссылок на изображения.
  + *withoutDefault* – false для выборки со стандартным изображением, иначе true
  + *forceId* – Положительное целое. Идентификатор апартаментов или null для использования текущего идентификатора.
  + *withoutTemps* - false для выборки со временными изображениями, иначе true
* array **getSocialUrl** (void) – получение полных http ссылок на социальные сети заданного города в виде массива строк.
* array **getApartmentCount** (bool $onlyPublic) – получение количества всех принадлежащих городу апартаментов в виде массива объектов.
  + *onlyPublic* – true для выборки общедоступных апартаментов, false для выборки всех
* array **getUsers** (int $role) – получение списка пользователей, управляющих заданным городом в виде массива объектов.
  + *role* – Целочисленная константа. Роль группы пользователей семейства USER\_ROLE\_\*
* bool **isActive** (bool $ignoreChilds) – получение состояния активности (доступности) города для пользователей в виде булевого значения.
  + *ignoreChilds* – true для игнорирования дочерних сущностей, иначе false
* bool **deleteFull** (bool $ignoreParent) – полное и безвозвратное удаление данных и ресурсов города и его апартаментов из системы.
  + *ignoreParent* – true для игнорирования родительских сущностей, иначе false
* bool **fieldIsChanged** (string $fieldName) – проверка изменения поля объекта данных и таблицы базы данных.
  + *fieldName* – Строка. Название поля для проверки состояния измененности с текущим значением.

Основные структуры данных

* **Array** – массив данных в php формате. Вида array (1 => ‘value’, …)
* **Object** – объект данных php (обычно модель данных базы данных) хранящий в себе различные методы и типы данных. Может быть как статической структурой, так и динамическим экземпляром одного класса.
* **String** и **Integer** – скалярное представление данных php.
* **Boolean** – булевое представление данных, может принимать всего два значения: true (истина) и false (ложь).
* **Null** – особый тип данных php, означающих пустое множество или «ничего», синоним Void в СИ подобных языках.
* **Запрос к системе** – набор данных, передаваемых от клиента на сервер через http протокол в формате GET или POST.
* **Ответ системы** – набор данных, передаваемых от сервера к клиенту через http протокол в формате текстовых (html, js, css), бинарных (изображения, звуки, мультимедиа) или json (ajax ответы) данных.

Коды статуса запроса

Для сообщения состояния запроса используются стандартные http коды

* **200** – Запрос обработан успешно
* **500** – На сервере произошла ошибка
* **404** – Ресурс не найден
* **403** – Нет доступа
* **502** – Таймаут обработки запроса

**План тестирования**

Процедура тестирования

Основные положения процедуры проведения тестирования:

1. В рамках курса «Верификация ПО», тестирование будет проведено один раз, то есть без повторного проведения после исправления найденных ошибок;
2. Процедура тестирования будет остановлена, только в случае нахождения блокирующих ошибок;
3. Тестирование будет проводится автором данного отчета, без привлечения других лиц;
4. Процедура тестирования будет признана выполненной успешно, если в ходе проведения не будет найдено блокирующих или критических ошибок.

Блочное тестирование

Первый вид тестирования, которому будет подвержена система - блочное тестирование. Данный вид тестирования будет применен к некоторым функциям из модуля City. Так как весь исходный код модулей написан на языке PHP, то для блочного и интеграционного тестирования будет использоваться инструмент PHPUnit в связке PhpStorm IDE v2017.1.

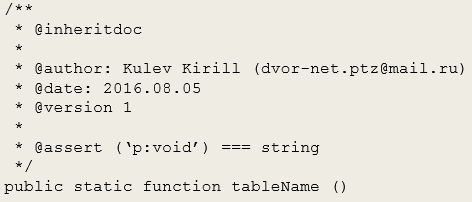
Список функций для тестирования:

* tableName
* rules
* attributeLabels
* getPublic
* getLogoImages
* getSocialUrl
* getApartmentCount
* getUsers
* isActive
* deleteFull
* fieldIsChanged

Детальное описание функций можно найти в разделе 1.

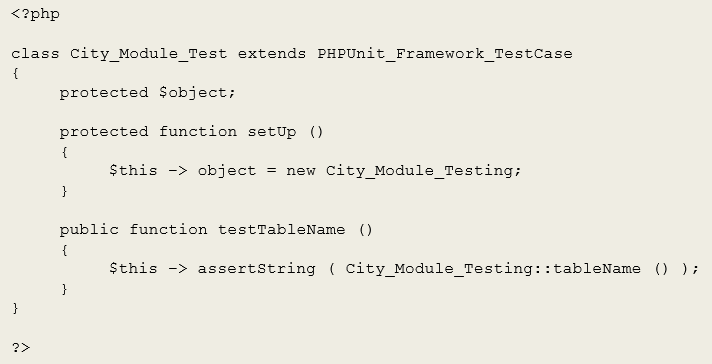
Для того, чтобы реализовать такой тест в PHPUnit с помощью PhpStorm IDE нужно в заголовочном комментарии к тестируемой функции добавить ключевое слово @asset и указать параметры теста.

Рассмотрим пример для функции tableName:



В данном случае функция будет тестироваться с входным параметром p:void, а ожидаемый результат выполнения функции будет string. Далее, нужно запустить генерацию тестов на основе исходно кода.

Для описанного выше способа сгенерируется следующий блочный тест PHPUnit:



Интеграционное тестирование

Второй этап - интеграционное тестирование. Для проведения интеграционного тестирования необходимо определить все возможные входные данные, соответствующие им ожидаемые результаты.

Будет проверяться интеграция модуля City и ядра системы Yii, протестированных блочно, на предыдущем этапе. Интегрироваться модуль будет по принципу использования функций из модуля City через вызов ядром Yii. Схематично порядок прохождения теста можно изобразить сл. схемой:

Конец теста

Вызов метода City

Вызов метода Yii

Начало теста

Аттестационное тестирование

Аттестационное тестирование будет проводится методом «Живого человека». В роли такого человека выступает сам автор тестирования.

Тестирующий человек, по заранее заданным инструкциям (Test Cases), производит требуемые действия и сверяется с заранее заданными результатами. Тест считается пройденным если в результате полученные такие же данные, которые описаны в инструкции. В противном случае тест считается негативным.

Аттестация системы будет производится по следующим высокоуровневым функциям:

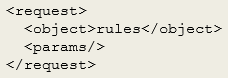
* Получение выборки набора общедоступных городов;
* Получение выборки набора пользователей, закрепленных за выбранным городом;
* Получение количества общедоступных апартаментов у заданного города;
* Получение списка ссылок на социальные сети выбранного города;
* Получение списка изображений (фотографий) выбранного города;
* Полное удаление выбранного города;

Нагрузочное тестирование

Четвертый этап - нагрузочное тестирование. Данный вид тестирования будет проводится на системе, развернутой на отдельном web-сервере. Для того, чтобы провести нагрузочные тесты нужно, определить модель нагрузки.

Для нагрузочных тестов будет использоваться инструмент Tsung.

Для того, чтобы создать нагрузочный тест в Tsung нужно, создать HTTP Request с необходимыми параметрами и, сопутствующими запросу, данными.



В поле «object» указывается имя тестируемой функции, в поле «params» набор входных параметров в виде подмассива.

Модель нагрузки:

1. Ответ от системы, для любого запроса, не должен превышать 45 секунд.
2. Ответ от системы не должен возвращать 502 ошибку (ошибка конца времени ожидания)

Тестируемые функции:

* tableName
* rules
* attributeLabels
* getPublic
* getLogoImages
* getSocialUrl
* getApartmentCount
* getUsers
* isActive
* deleteFull
* fieldIsChanged

Критерий прохождения тестов

Тест считается успешно пройденным, если ожидаемый и фактический результаты

совпадают. Если тест завершается неудачей, то перед принятием решения

целесообразно проверить правильность самого теста. Если тест завершился неудачей

и тест реализован правильно, то производится заключение о найденной ошибке.

Тестирование считается пройденным, если во время его прохождения не выявлено

критических ошибок, а процент непройденных тестов меньше 1% от общего

количества.

Критерий приостановки тестов

Тестирование должно быть приостановлено, если количество непройденных тестов

превысит 10% от их общего количества. Тестирование должно быть приостановлено

при обнаружении критических ошибок.

Критерий возобновления тестирования

Тестирование возобновляется после исправления ошибок, выявленных при

предыдущем тестировании.

Программное обеспечение для проведения тестирования

Для проведения тестирования используется настольный компьютер с установленной

операционной системой Windows Server 2016 Standard Edition

Описание тестов

* П – простой
* С – специальный
* Н – негативный

Блочные тесты

|  |  |
| --- | --- |
| **Тест** | 1-П |
| **Описание** | Получение имени таблицы модуля City |
| **Функция** | string **tableName** (void) |
| **Входные параметры** | [] |
| **Ожидаемый результат** | (string) ‘city’ |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 2-П |
| **Описание** | Получение списка правил валидации полей объекта данных модуля City при взаимодействии с базой данных |
| **Функция** | array **rules** (void) |
| **Входные параметры** | [] |
| **Ожидаемый результат** | (array) [  [['url', 'name', 'manager', 'phone1', 'email'], 'required'],  [['parent\_id', 'viber', 'whatsapp', 'skype', 'map\_zoom', 'wubook\_synchronization', 'wubook\_booking'], 'integer'],  [['latitude', 'longitude'], 'number'],  [['title', 'description', 'google\_statistic', 'jivosite', 'yandex\_metrika'], 'string'],  [['wubook\_login'], 'string', 'max' => 10],  [['phone1', 'phone2', 'wubook\_lcode'], 'string', 'max' => 20],  [['url', 'name', 'wubook\_password'], 'string', 'max' => 30],  [['manager', 'address', 'email', 'wubook\_pcode'], 'string', 'max' => 50],  [['vkontakte', 'facebook', 'youtube'], 'string', 'max' => 100],  [['parent\_id'], 'exist', 'skipOnError' => true, 'targetClass' => City::className(), 'targetAttribute' => ['parent\_id' => 'id']],  ] |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 3-П |
| **Описание** | Получение списка полей объекта данных и их синонимов в таблице базы данных |
| **Функция** | array **attributeLabels** (void) |
| **Входные параметры** | [] |
| **Ожидаемый результат** | (array) [  'id' => \Yii::t ( 'models/city', 'Entry ID' ),  'parent\_id' => \Yii::t ( 'models/city', 'Parent ID' ),  'url' => \Yii::t ( 'models/city', 'Url' ),  'name' => \Yii::t ( 'models/city', 'Name' ),  'title' => \Yii::t ( 'models/city', 'Title' ),  'description' => \Yii::t ( 'models/city', 'Description' ),  'manager' => \Yii::t ( 'models/city', 'Manager' ),  'address' => \Yii::t ( 'models/city', 'Address' ),  'phone1' => \Yii::t ( 'models/city', 'Phone1' ),  'phone2' => \Yii::t ( 'models/city', 'Phone2' ),  'email' => \Yii::t ( 'models/city', 'Email' ),  'viber' => \Yii::t ( 'models/city', 'Viber' ),  'whatsapp' => \Yii::t ( 'models/city', 'Whatsapp' ),  'skype' => \Yii::t ( 'models/city', 'Skype' ),  'vkontakte' => \Yii::t ( 'models/city', 'Vkontakte' ),  'facebook' => \Yii::t ( 'models/city', 'Facebook' ),  'youtube' => \Yii::t ( 'models/city', 'Youtube' ),  'jivosite' => \Yii::t ( 'models/city', 'Jivosite' ),  'yandex\_metrika' => \Yii::t ( 'models/city', 'Yandex Metrika' ),  'google\_statistic' => \Yii::t ( 'models/city', 'Google statistic' ),  'wubook\_login' => \Yii::t ( 'models/city', 'Wubook login' ),  'wubook\_password' => \Yii::t ( 'models/city', 'Wubook password' ),  'wubook\_lcode' => \Yii::t ( 'models/city', 'Wubook code (L)' ),  'wubook\_pcode' => \Yii::t ( 'models/city', 'Wubook code (P)' ),  'latitude' => \Yii::t ( 'models/city', 'Latitude' ),  'longitude' => \Yii::t ( 'models/city', 'Longitude' ),  'map\_zoom' => \Yii::t ( 'models/city', 'Zoom' ),] |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 4-П |
| **Описание** | Получение всех общедоступных апартаментов в виде массива объектов |
| **Функция** | array **getPublic** (int $cityId, array $order) |
| **Входные параметры** | [1, [‘name’=>SORT\_ASC]] |
| **Ожидаемый результат** | (array) [{}, …] |
| **Результат** | Запусков: 30 (10)  Итераций: 3  Ошибок: 2  Исправлений: 2  Успешность: 80% (100%) |
| **Тест** | 4-Н |
| **Описание** | Получение всех общедоступных апартаментов в виде массива объектов |
| **Функция** | array **getPublic** (int $cityId, array $order) |
| **Входные параметры** | [100500, [‘name’=>SORT\_ERROR]] |
| **Ожидаемый результат** | \Exception |
| **Результат** | Запусков: 10  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 5-П |
| **Описание** | Получение фотографий апартаментов в виде массива ссылок на изображения |
| **Функция** | array **getLogoImages** (bool $withoutDefault, int $forceId, bool $withoutTemps) |
| **Входные параметры** | [false, null, true] |
| **Ожидаемый результат** | [(string), …] |
| **Результат** | Запусков: 30 (15)  Итераций: 2  Ошибок: 1  Исправлений: 1  Успешность: 94% (100%) |
| **Тест** | 5-C |
| **Описание** | Получение фотографий апартаментов в виде массива ссылок на изображения |
| **Функция** | array **getLogoImages** (bool $withoutDefault, int $forceId, bool $withoutTemps) |
| **Входные параметры** | [true, 1, false] |
| **Ожидаемый результат** | [(string), …] |
| **Результат** | Запусков: 15  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 6-П |
| **Описание** | Получение полных http ссылок на социальные сети заданного города в виде массива строк |
| **Функция** | array **getSocialUrl** (void) |
| **Входные параметры** | [] |
| **Ожидаемый результат** | [(string), …] |
| **Результат** | Запусков: 10 (5)  Итераций: 2  Ошибок: 1  Исправлений: 1  Успешность: 80% (100%) |
| **Тест** | 7-П |
| **Описание** | Получение количества всех принадлежащих городу апартаментов в виде массива объектов |
| **Функция** | array **getApartmentCount** (bool $onlyPublic) |
| **Входные параметры** | [false] |
| **Ожидаемый результат** | [{}, …] |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 7-С |
| **Описание** | Получение количества всех принадлежащих городу апартаментов в виде массива объектов |
| **Функция** | array **getApartmentCount** (bool $onlyPublic) |
| **Входные параметры** | [true] |
| **Ожидаемый результат** | [{}, …] |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 8-П |
| **Описание** | Получение списка пользователей, управляющих заданным городом в виде массива объектов |
| **Функция** | array **getUsers** (int $role) |
| **Входные параметры** | [1] |
| **Ожидаемый результат** | [{}, …] |
| **Результат** | Запусков: 10 (5)  Итераций: 2  Ошибок: 1  Исправлений: 1  Успешность: 80% (100%) |
| **Тест** | 9-П |
| **Описание** | Получение состояния активности (доступности) города для пользователей в виде булевого значения |
| **Функция** | bool **isActive** (bool $ignoreChilds) |
| **Входные параметры** | [false] |
| **Ожидаемый результат** | (bool) true |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 9-С |
| **Описание** | Получение состояния активности (доступности) города для пользователей в виде булевого значения |
| **Функция** | bool **isActive** (bool $ignoreChilds) |
| **Входные параметры** | [true] |
| **Ожидаемый результат** | (bool) true |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 10-П |
| **Описание** | Полное и безвозвратное удаление данных и ресурсов города и его апартаментов из системы |
| **Функция** | bool **deleteFull** (bool $ignoreParent) |
| **Входные параметры** | [false] |
| **Ожидаемый результат** | (bool) true |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 11-П |
| **Описание** | Проверка изменения поля объекта данных и таблицы базы данных |
| **Функция** | bool **fieldIsChanged** (string $fieldName) |
| **Входные параметры** | (string) ‘…’ |
| **Ожидаемый результат** | (bool) false |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |

Интеграционные тесты

|  |  |
| --- | --- |
| **Тест** | 1-С |
| **Описание** | Получение состояния активности (доступности) города для пользователей в виде булевого значения + проверка положительного кол-ва доступных апартаментов. |
| **Функция** | bool **isActive** (bool $ignoreChilds) |
| **Взаимодействие** | array **getApartmentCount** (bool $onlyPublic) |
| **Входные параметры** | [false] |
| **Ожидаемый результат** | (bool) true |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 2-С |
| **Описание** | Получение всех общедоступных апартаментов в виде массива объектов + проверка модели города на доступность пользователям. |
| **Функция** | array **getPublic** (int $cityId, array $order) |
| **Взаимодействие** | bool **isActive** (bool $ignoreChilds) |
| **Входные параметры** | [1, [‘name’=>SORT\_ASC]] |
| **Ожидаемый результат** | [{}, …] |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 3-С |
| **Описание** | Получение всех общедоступных апартаментов в виде массива объектов + проверка модели города на доступность пользователям. |
| **Функция** | array **getSocialUrl** (void) |
| **Взаимодействие** | bool **isActive** (bool $ignoreChilds)  array **getPublic** (int $cityId, array $order) |
| **Входные параметры** | [] |
| **Ожидаемый результат** | [(string), …] |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 4-С |
| **Описание** | Получение всех общедоступных апартаментов в виде массива объектов + проверка модели города на доступность пользователям. |
| **Функция** | bool **deleteFull** (bool $ignoreParent) |
| **Взаимодействие** | array **getUsers** (int $role)  array **getLogoImages** (bool $withoutDefault, int $forceId, bool $withoutTemps) |
| **Входные параметры** | [false] |
| **Ожидаемый результат** | (bool) true |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |

Аттестационные тесты

|  |  |
| --- | --- |
| **Тест** | 1-П |
| **Описание** | Авторизация в системе через форму авторизации |
| **Входные параметры** | Заполненные поля «email», «пароль» |
| **Ожидаемый результат** | Сообщение об успешной авторизации, переход на главную страницу от лица авторизованного пользователя |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 2-П |
| **Описание** | Выход из системы |
| **Входные параметры** | Нажатие кнопки «выход» в шапке сайта |
| **Ожидаемый результат** | Сообщение об успешном выходе, переход на главную страницу от лица гостя |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 3-П |
| **Описание** | Получение списка доступных городов |
| **Входные параметры** | Переход на главную страницу |
| **Ожидаемый результат** | Выгрузка списка городов в автоматическом режиме в виде фото-плиток |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 4-П |
| **Описание** | Получение списка доступных апартаментов города |
| **Входные параметры** | Клик по фото-плитке города, переход на страницу «Выкладка апартаментов», заполнение фильтра цен и кол-ва комнат в апартаментах |
| **Ожидаемый результат** | Выгрузка списка апартаментов в автоматическом режиме в виде фото-каруселей с информацией о цене и отображением на карте |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 5-П |
| **Описание** | Получение данных определенных апартаментов |
| **Входные параметры** | Клик по фото-карусели апартаментов, переход на страницу «Информация об апартаментах» |
| **Ожидаемый результат** | Выгрузка подробной информации об апартаментах (включая цену, контакты менеджеров и фотографии) |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |
| **Тест** | 6-П |
| **Описание** | Поиск апартаментов через форму поиска |
| **Входные параметры** | Заполнение поля поиска на главной странице и клик по кнопке «Найти» |
| **Ожидаемый результат** | Выгрузка списка апартаментов соответствующих заданным критериям поиска в виде списка фото-каруселей с информацией о цене и отображением на карте |
| **Результат** | Запусков: 5  Итераций: 1  Ошибок: 0  Исправлений: 0  Успешность: 100% |

Нагрузочное тестирование

Результаты нагрузочного тестирования представлены в виде графика и таблицы. Видно, что имеет место экспоненциальная зависимость между временем ожидания обработки запроса и количеством посылаемых запросов. Это объясняется архитектурой веб стека (в один момент серверный скрипт может обрабатывать только 1 запрос) и архитектурой веб-приложения (отсутствуют параллельные потоки).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Метод** | **Запросов** | **Ср. в/о (мс)** | **Мин. в/о (мс)** | **Мак. в/о (мс)** | **Отключение** |
| tableName | 1000 | 500 | 389 | 617 | - |
| rules | 1000 | 503 | 374 | 647 | - |
| attributeLabels | 1000 | 621 | 588 | 968 | - |
| getPublic | 1000 | 1019 | 981 | 1692 | 830 |
| getLogoImages | 1000 | 537 | 414 | 593 | - |
| getSocialUrl | 1000 | 918 | 793 | 971 | - |
| getApartmentCount | 1000 | 611 | 535 | 672 | - |
| getUsers | 1000 | 995 | 917 | 1334 | 972 |
| isActive | 1000 | 601 | 513 | 623 | - |
| deleteFull | 10 | 618 | 526 | 631 | - |
| fieldIsChanged | 1000 | 583 | 519 | 592 | - |

График обработки запросов системой:



**Отчеты об ошибках**

Отчет №1

Краткое описание: ​Ошибка при выполнении теста 4-П. Проверка выбрасывания

ошибки при отрицательном значении идентификатора города.

Ожидаемый результат: ​Возврат ошибки выполнения функции.

Фактический результат: ​Функция получает пустую выборку из БД и возвращает пустой набор. Ошибка не возвращается.

Отчет №2

Краткое описание: ​Ошибка при выполнении теста 4-П. Проверка выбрасывания

ошибки при неправильном направлении сортировки результата.

Ожидаемый результат: ​Возврат ошибки выполнения функции.

Фактический результат: ​Функция не производит никакой сортировки, однако возвращает несортированную коллекцию. Ошибка не возвращается.

Отчет №3

Краткое описание: ​Ошибка при выполнении теста 5-П. Проверка выбрасывания

ошибки при NULL значении булевых параметров.

Ожидаемый результат: ​Возврат ошибки выполнения функции.

Фактический результат: ​Функция приводит NULL тип к FALSE значению и благополучно отрабатывает. Ошибка не возвращается.

Отчет №4

Краткое описание: ​Ошибка при выполнении теста 6-П. Тест проверяет работу функции при обрыве интернет-соединения.

Ожидаемый результат: Возврат ошибки выполнения функции.

Фактический результат: ​После обрыва и восстановления интернет-соединения,

тестируемый объект не возвращает пустой набор вместо ошибки выполнения. Ошибка не возвращается.

Отчет №5

Краткое описание: ​Ошибка при выполнении теста 8-П. Тест проверяет работу функции для пустых наборов пользователей города.

Ожидаемый результат: Возврат пустого набора.

Фактический результат: ​При отсутствующих пользователях вместо пустого набора возвращается ошибка обработки функции.

Отчет №6

Краткое описание: ​Ошибка при выполнении нагрузочного тестирования. Тест

проверяет работу объекта при высокой интенсивности поступающих запросов.

Ожидаемый результат: ​Линейное время обработки сигнала, не превышающее 30 секунд.

Фактический результат: ​Время обработки доходит до 45 секунды и возвращается ошибка 502 (timeout).

**Тестовое покрытие**

Объемность современного программного обеспечения и инфраструктуры проекта сделали невозможной для одного человека задачу проведения тестирования со 100% тестовым покрытием. Расчет тестового покрытия относительно исполняемого кода программного обеспечения проводится по формуле:



* Tcov - тестовое покрытие
* Ltc - количество строк кода, покрытых тестами
* Lcode - общее количество строк кода

На данный момент, без учета файлов онтологии и заголовочных файлов, количество строк кода проекта составляет 58293 строки. Из них покрыто тестами: 10813 строк. Таким образом тестовое покрытие составляет 18.54%.

Стоит так же помнить, что в общее количество строк также входит фреймворк Yii2 тестирование которого проводилось разработчиками. На их сайте заявлено, что тестированию подвергался весь код фреймворка. На данный момент код фреймворка Yii2 в проекте занимает 31214 строк кода.

Для получения полного покрытия стоит произвести расчет без учета доли фреймворка Yii2: 10813 / (58293 - 31214) \* 100% = 40.04% покрытия тестами.

**Результаты**

В ходе выполнения тестов было выявлено несколько ошибок, которые представлены в отчетах тестирования.

По полученным результатам тестирования, можно судить, что объект готов к работе после

проведенных доработок. Критических ошибок, которые требовалось бы исправлять срочно, не обнаружено.

Стоит также учесть, что 502я шибка времени выполнения является нормой для любого web ресурса. Степень ее негативного влияния на продукт определяет соотношение между временем обработки и количеством одновременных запросов. В рамках данного тестирования соотношение 1 секунда на обработку при 1000 одновременных запросов является хорошим показателем и не требует вмешательства со стороны разработчиков.

**Используемое программное обеспечение**

* PHPUnit - <https://phpunit.de/>
* Tsung - <http://tsung.erlang-projects.org/>
* PHPStorm - <https://www.jetbrains.com/phpstorm/>
* Apache - <https://httpd.apache.org/>
* PHP - <http://php.net/>