1. Тестирование

1.1. Модульное тестирование

Статический класс DataMethods:

Представляет собой интерфейс бекенда данных.

Производится тестирование класса DataMethods с использованием методики тестирования – разбиение на уровне класса на категории по функциональности. Категория объединяет в себе методы класса, выполняющие близкую по смыслу функциональность.

Методы класса можно разбить на 3 категории по функциональности:

- методы получения данных;
- методы изменения данных;
- методы удаления данных.

Таблица 1: Методы класса DataMethodsFilter

Название метода	Примечания
get_item	param: table [str] param: value_json[dict] Возвращает все элементы таблицы с именем table, удовлетворяющие фильтру value_json
put_item	param: table [str] param: value_json [dict] Изменяет все элементы таблицы с именем table, удовлетворяющие фильтру value_json
delete_item	param: table [str] param: value_json [dict] Удаляет все элементы таблицы с именем table, удовлетворяющие фильтру value_json

Таблица 2: Категория 1 – Тестирование метода получения данных

Название теста	TestGetItemEmpty
Тестируемый метод	get_item
Описание теста	Проверка получения данных из БД, пустой таблицы trait
Ожидаемый результат	Словарь с пустым списком в ключе “result”
Степень важности	Фатальная
Результат теста	Тест пройден

Таблица 3: Категория 1 – Тестирование метода получения данных

Название теста	TestGetItemGeneral
Тестируемый метод	get_item
Описание теста	Проверка изменения данных в БД, в таблице trait, предварительно наполненной записями name = “test_name”, version = “1.0” name = “test_name”, version = “2.0”, по фильтру {“name”:“test_name”}
Ожидаемый результат	Словарь {“result”:[{“name”:“test_name”, “version”:“1.0”}, {“name”:“test_name”,} “version”:“2.0”}]}
Степень важности	Фатальная
Результат теста	Тест пройден

Таблица 4: Категория 2 – Тестирование метода изменения данных

Название теста	TestPutItem
Тестируемый метод	put_item
Описание теста	Проверка изменения данных в БД, в таблице trait, предварительно наполненной записями name = “test_name”, version = “1.0” name = “test_name”, version = “2.0”, по фильтру {“name”:“test_name”,“changes”: {“version”:“3.0”}}
Ожидаемый результат	Словарь {“count”: 2}, отражающий наличие двух произведенных изменений в БД
Степень важности	Фатальная
Результат теста	Тест пройден

Таблица 5: Категория 3 – Тестирование метода удаления данных

Название теста	TestDeleteItem
Тестируемый метод	delete_item
Описание теста	Проверка удаления данных в БД, в таблице trait, предварительно наполненной записями name = “test_name”, version = “1.0” name = “test_name”, version = “2.0”, по фильтру {“name”:“test_name”}
Ожидаемый результат	Словарь {“count”: 2}, отражающий наличие двух произведенных удалений в БД
Степень важности	Фатальная
Результат теста	Тест пройден

Вывод по результатам тестирования

Все тесты пройдены успешно, класс готов к использованию.

1.2. Системное тестирование

Производится системное тестирования файлового сервера (стратегия чёрного ящика). Тест покрывает все прецеденты взаимодействия с файловым сервером.

При формировании тестовых наборов использовалась методика эквивалентного разбиения для входных данных.

Классы эквивалентности для входных данных:

Параметр	Допустимые классы эквивалентности	Недопустимые классы эквивалентности
Имена файлов	Строки, не содержащие символы &,$,/,:,*,? и другие спецсимволы	Строки, содержащие запрещённые символы
Адрес сервера	Строки вида 'http://address:port'	Строки, не подходящее под описание
Удалённый ресурс	Такие же строки, как и допустимые для имён файлов	Такие же строки, как и недопустимые для имён файлов
Запрос-файл	Запрос с содержимым файла в теле и с хедером 'Content-type':'multipart/form-data'	Запрос без необходимого заголовка либо испорченным телом

Тесты:

№	Описание теста	Входные данные	Ожидаемый результат	Полученный результат
1	Проверка возможности сохранения файла	Файл-запрос	Сообщение об успешном сохранении файла	Сообщение протокола HTTP 200 OK, пустой JSON-объект
2	Проверка доступа к существующему файлу	Запрос по адресу файла	Содержимое файла	Сообщение протокола HTTP 200 OK, содержимое искомого файла
3	Проверка удаления существующего файла	Запрос на удаление файла	Сообщение об успешном удалении файла	Сообщение протокола HTTP 200 OK, пустой JSON-объект
4	Проверка удаления несуществующего файла	Запрос на удаление файла по несуществующему пути	Сообщение об ошибке	Сообщение протокола HTTP 404 NOT FOUND, JSON-объект с сообщением об ошибке
5	Проверка доступа к несуществующему файлу	Запрос по адресу несуществующего файла	Сообщение об ошибке	Сообщение протокола HTTP 404 NOT FOUND, JSON-объект с сообщением об ошибке

Подробное описание тестов:

Тестирование надёжности и доступности серверов

№	Описание теста	Входные данные	Ожидаемый результат	Полученный результат
1	Провести тест, в котором несколько пользователей одновременно загружают файл на сервер	Пользовательские файлы, их имена	Все файлы загружены по верным адресам	Каждый из пользователей успешно загрузил файл на сервер
2	Провести тест, в котором происходит разрыв соединения в ходе загрузки файла	Пользовательский файл	Загрузка возобновляется по восстановлении соединения	Возникла ошибка. Необходима повторная загрузка файла.

Отчёт об обнаруженных ошибках:

Сервис: файловый сервер	
Степень важности: средняя	
Надёжность	Сервис некорректно отрабатывает сценарий разрыва соединения

Выводы по результатам системного тестирования

Сервис может быть использован только в средах, где сеть можно считать достаточно надёжной. Для использования в рамках сетей где часты разрывы необходимо производить доработку сервиса.

Код теста

```
#!/bin/bash

echo "#include <iostream>" > test.cpp
echo "int main(){std::cout << \"HELLO WORLD!\" << std::endl;}" >> test.cpp
g++ test.cpp -o a.out

set -v

```

```
curl @"localhost:50002/static"                    -X POST -F file=@a.out; echo
curl @"localhost:50002/static?path=1\2\3\4" -X POST -F file=@a.out; echo

curl @"localhost:50002/static/a.out"              -X GET > a__.out   ; echo
chmod +x a__.out                                                     ; echo
./a__.out                                                            ; echo

curl @"localhost:50002/static/1\2\3\4\a.out" -X GET > a_.out     ; echo
chmod +x a_.out                                                  ; echo
./a_.out                                                         ; echo

curl @"localhost:50002/static/a.out"              -X DELETE          ; echo
curl @"localhost:50002/static/1\2\3\4\a.out" -X DELETE           ; echo

curl @"localhost:50002/static/a.out"              -X DELETE          ; echo
curl @"localhost:50002/static/1\2\3\4\a.out" -X DELETE           ; echo

curl @"localhost:50002/static/a.out"              -X GET             ; echo
curl @"localhost:50002/static/1\2\3\4\a.out" -X GET              ; echo

rm a*.out test.cpp
```

1.3. Интеграционное тестирование

Производится интеграционное тестирование подсистем работы с файлами, данными и подсистемы мониторинга. Целью тестирования являеется проверка корректности регистрации подсистем на маяке, играющем ключевую роль в работе распределенной системы.

Интерфейс подсистемы мониторинга(категория – beacon, сокращенно – маяк):

- GET /services/<service_group>
- PUT /services/<service_group>/<service_host>:<service_port>
- POST /services

Методы подсистемы работы с файлами(категория – filesystem):

- beacon_setter
- beacon_getter

Методы подсистемы работы с данными(категория – database):

- beacon_setter
- beacon_getter

Таблица 6: Тестирование регистрации бекендов на не запущенном маяке

Название файла	TestNoBeaconGetter.sh
Взаимодействующие подсистемы	database, filesystem, beacon
Описание теста	Подсистемы filesystem и database выполняют метод beacon_setter при запуске
Начальные условия Ожидаемый результат	beacon не запущен подсистемы не могут подключиться к beacon и сообщают об этом пользователю; повторный поиск происходит регулярно, с интервалом в 10 секунд
Степень важности	Фатальная
Результат теста	Тест пройден

Таблица 7: Тестирование регистрации бекендов на запущенном маяке

Название файла	TestBeaconPoster.sh
Взаимодействующие подсистемы	database, filesystem, beacon
Описание теста	Подсистемы filesystem и database выполняют метод beacon_setter при запуске
Начальные условия	beacon запущен
Ожидаемый результат	Подсистемы подключаются к маяку и выполняют POST-запрос на адрес /services/<категория бекенда>, передавая через JSON свою адрес Журнал маяка: POST /services/database HTTP/1.1 200 POST /services/filesystem HTTP/1.1 200
Степень важности	Фатальная
Результат теста	Тест пройден

Таблица 8: Тестирование регистрации бекендов на запущенном маяке

Название файла	TestBeaconPutter.sh
Взаимодействующие подсистемы	database, filesystem, beacon
Описание теста	Подсистемы filesystem и database выполняют метод beacon_setter повторно
Начальные условия	beacon запущен, подсистемы уже выполнили первичный POST запрос
Ожидаемый результат	Подсистемы подключаются к маяку и выполняют PUT-запрос на адрес /services/<категория бекенда>/<адрес бекенда>:<порт>. Пример журнала маяка: PUT /services/database/localhost:5000 HTTP/1.1 200 PUT /services/filesystem/localhost:5001 HTTP/1.1 200
Степень важности	Фатальная
Результат теста	Тест пройден

Таблица 9: Тестирование регистрации бекендов на запущенном маяке

Название файла	TestBeaconGetter.sh
Взаимодействующие подсистемы	filesystem, beacon
Описание теста	Подсистема filesystem запрашивает адреса всех подсистем database у маяка
Начальные условия	beacon запущен, подсистемы уже выполнили первичный POST запрос
Ожидаемый результат	filesystem подключаются к маяку и выполняет GET-запрос(метод beacon_getter) на адрес /services/database/. Маяк отвечает все известные адреса бекендов, категории database в JSON-формате. Пример ответа – [{“localhost”: 5000}]
Степень важности	Фатальная
Результат теста	Тест пройден

Вывод:

Интеграционное тестирование выявило фатальную ошибку в реализации подсистемы beacon, связанную с неверно документированным поведением метода доступа к элементу по ссылке в словарях многократной вложенности в языке Python.

Ошибка была устранена с использованием альтернативной библиотченой реализации этого метода.