ФГБОУ ВО «ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
КАФЕДРА ИНФОРМАТИКИ И МАТЕМАТИЧЕСКОГО ОБЕСПЕЧЕНИЯ

Отчет по дисциплине «Верификация программного обеспечения»
Тестирование мобильного приложения для мониторинга курса акций

Выполнил:
студент 6-го курса
группы №22’608
Орлов Кирилл Евгеньевич

Преподаватель:
к.ф-м.н., доцент К. А. Кулаков

Петрозаводск – 2018

Оглавление

Объект тестирования

В рамках курса «Верификация программного обеспечения» будет рассматриваться мобильный сервис для мониторинга курса акций различных компаний. Данный сервис позволяет следить за курсом акций в разные дни, настроить список акций избранных компаний, также получить предсказание о том, как будет изменяться курс акций конкретной компании в будущем.

Сервис состоит из сервера и мобильного приложения. Будем считать, что сервер всегда работает правильно. Объектом тестирования является мобильное приложение, разработанное для операционной системы iOS версии 11.0+ и написанное на языке Swift 4.0.

Основными экранами приложения являются:

1. Экран Earnings Calendar.

Экран предназначен для просмотра курса акций компаний в разные дни месяца. Экран состоит из области календаря и области акций, где:

1) Область календаря – представляет собой интерфейс для выбора дня недели, за который будут отображаться акции компаний.

2) Область акций – это вертикально прокручивающийся список компаний с указанием текущего курса акций каждой компании.

С экрана возможен переход на экраны Watch List, Filters и Search.

2. Экран Watch List.

Экран предназначен для отображения списка избранных компаний. Состоит из прокручивающегося списка избранных компаний с указанием текущего курса акций каждой компании.

С экрана возможен переход на экраны Earnings Calendar и Search.

3. Экран Filters.

Экран предназначен для выбора фильтров (например, компании в области IT, медицины и т. д.). Список отображаемых компаний на экране Earnings Calendar отображается с учётом выбранных фильтров на экране Filters.

4. Экран Search.

Экран предназначен для просмотра списка всех компаний, поиска компаний, добавления интересующих пользователя компаний в Watch List.

Архитектура приложения представлена на рисунке 1.

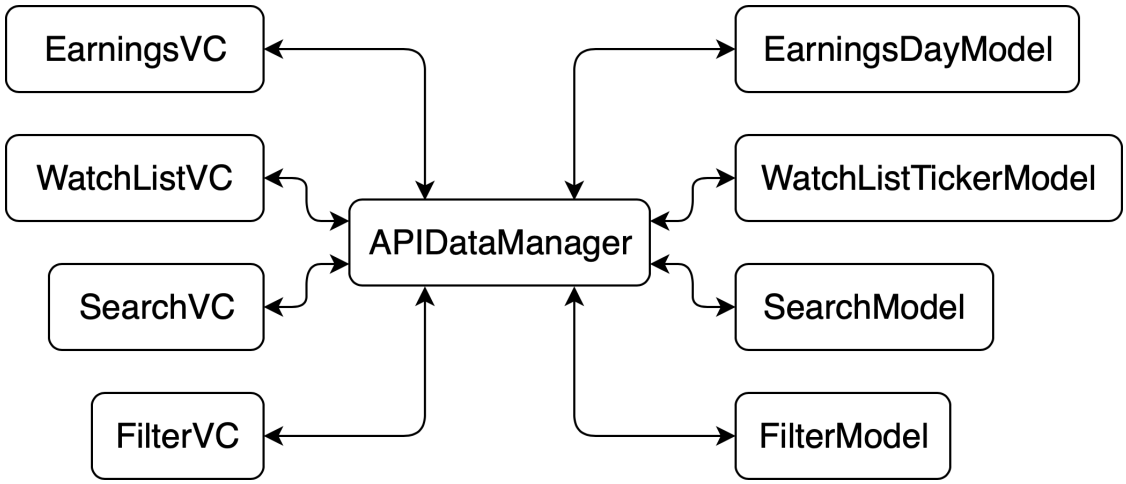

Рисунок 1 – Архитектура приложения

Функциональные требования к объекту тестирования

Тестируемое приложение должно реализовывать следующий набор функциональных требований:

1. Экран Earnings Calendar:

    1.1. Получение с сервера JSON-файла с календарём, его парсинг, отображение доступных недель и списка компаний с курсом акций.

    1.2. Переключение дней недели и обновление списка акций.

    1.3. Сортировка списка компаний по различным параметрам (название, стоимость акций и прочее).

2. Экран Watch List:

2.1. Получение с сервера JSON-файла, парсинг и отображение списка акций избранных компаний.

2.2. Удаление компании из списка избранных.

2.3. Сортировка списка компаний по различным параметрам (название, стоимость акций и прочее).

3. Экран Filters:

3.1. Выбор нужного набора фильтров.

3.2. Обновление экрана Earnings Calendar, с учётом выбранных фильтров.

4. Экран Search:

4.1. Получение с сервера JSON-файла, парсинг, отображение списка всех доступных компаний.

4.2. Изменение списка избранных компаний.

4.3. Обновление экрана Watch List с учётом обновленного списка избранных компаний.

Все ошибки, возвращаемые с сервера, обрабатываются и сопровождаются соответствующими уведомлениями пользователя путём использования всплывающих окон. Аналогично обрабатывается ситуация, когда подключение к сети Интернет отсутствует.

Описание модулей

Приложение разработано с использованием паттерна MVC (Model-View-Controller). Основными модулями приложения являются:

1. APIDataManager.

Назначение: Модуль отвечает за взаимодействие с сервером.

Функции:

- `func getSearchList(completionHandler: @escaping ([SearchModel]?, ErrorType?) -> ())` – Получает с сервера список всех доступных компаний. Возвращает список компаний или ошибку.
- `func setUserTickers(tickers: [String], completionHandler: @escaping (ErrorType?) -> ())` – Отправляет на сервер актуальный список избранных компаний пользователя. В случае ошибки возвращает ошибку.
- `func getWatchList(completionHandler: @escaping ([WatchListTickerModel]?, ErrorType?) -> ())` – Получает с сервера список избранных компаний пользователя. Возвращает список избранных компаний или ошибку.
- `func getEarningsCalendar(completionHandler: @escaping ([EarningsDayModel]?, ErrorType?) -> ())` – Получает с сервера календарь и список компаний с курсом акций в каждый день. Возвращает заполненный календарь или ошибку.

2. EarningsVC.

Назначение: Контроллер экрана Earnings Calendar.
Функции:

- `func getEarnings()` – Получает Earnings Calendar от APIDataManager и подставляет данные во View.
- `func sortEarningsCalendar(by key: SortType)` – Сортирует Earnings Calendar в зависимости от ключа сортировки.

3. WatchListVC.

Назначение: Контроллер экрана Watch List.
Функции:

- `func showWatchList()` – Получает Watch List от APIDataManager и подставляет его во View.
- `func sortWatchList(by key: SortType)` – Сортирует Watch List в зависимости от ключа сортировки.

4. SearchVC.

Назначение: Контроллер экрана Search.

Функции:

- `func showSearchList()` – Получает список всех компаний от APIDataManager и подставляет данные во View.
- `func saveUserTickers()` – Сохраняет список избранных компаний пользователя.

5. FilterVC.

Назначение: Контроллер экрана Filters.

Функции:

- `func showFilters()` – Получает список фильтров и подставляет данные во View.
- `func saveUserFilters()` – Сохраняет список фильтров пользователя.

6. EarningsDayModel.

Назначение: Модель данных для компаний на экране Earnings Calendar.

Функции:

- `func getEarningsCalendar(from data: NSDictionary?) -> [EarningsDayModel]?` – Получает Earnings Calendar в формате JSON от APIDataManager, преобразует его в EarningsDayModel и возвращает в APIDataManager.

7. WatchListTickerModel.

Назначение: Модель данных для компаний на экране Watch List.

Функции:

- `func getWatchList(from data: NSDictionary?) -> [WatchListTickerModel]?` – Получает Watch List в формате JSON от APIDataManager, преобразует его в WatchListTickerModel и возвращает в APIDataManager.

8. SearchModel.

Назначение: Модель данных для компаний на экране Search.

Функции:

- `func getSearchList(from data: NSDictionary?) -> [SearchModel]?` – Получает список всех компаний от APIDataManager, преобразует его в SearchModel и возвращает в APIDataManager.

9. FilterModel.

Назначение: Модель данных для фильтров на экране Filters.

Функции:

- `func getFilters() -> [FilterModel]?` – Возвращает список фильтров в FiltersVC для отображения.

Объект и стратегия тестирования

Общая информация

Для тестирования будет использоваться стандартный инструментарий, встроенный в среду разработки Xcode версии 10.0. Тесты будут разрабатываться на языке Swift 4.0.

Стратегия модульного тестирования

Модульное тестирование будет проводиться для network-слоя приложения, а также для моделей данных, отвечающих за парсинг JSON-файлов, получаемых с сервера.

Следующие функции подлежат модульному тестированию:

1. `func getSearchList(completionHandler: @escaping ([SearchModel]?, ErrorType?) -> ())`
2. `func setUserTickers(tickers: [String], completionHandler: @escaping (ErrorType?) -> ())`
3. `func getWatchList(completionHandler: @escaping ([WatchListTickerModel]?, ErrorType?) -> ())`
4. `func getEarningsCalendar(completionHandler: @escaping ([EarningsDayModel]?, ErrorType?) -> ())`

5. `func sortEarningsCalendar(by key: SortType)`

6. `func sortWatchList(by key: SortType)`

7. `func getEarningsCalendar(from data: NSDictionary?) -> [EarningsDayModel]?`

8. `func getWatchList(from data: NSDictionary?) -> [WatchList-TickerModel]?`

9. `func getSearchList(from data: NSDictionary?) -> [SearchModel]?`

10. `func getFilters() -> [FilterModel]?`

Функции, отвечающие за отображение данных на экране, не подлежат модульному тестированию. Работоспособность этих функций будет проверена в рамках аттестационного тестирования.

Стратегия интеграционного тестирования

Цель интеграционного тестирования – удостовериться в корректности совместной работы элементов приложения (каждый модуль не только выполняет свои функции, но и взаимодействует с другими модулями без ошибок).

В данной работе выбран принцип восходящего тестирования. Сначала будет тестироваться самый нижний уровень системы. Затем постепенно к более низким уровням будут интегрироваться более высокоуровневые модули.

Этапы интеграции

1. Earnings Calendar flow:

APIDataManager: `func getEarningsCalendar(completionHandler: @escaping ([EarningsDayModel]?, ErrorType?) -> ())`

EarningsDayModel: `func getEarningsCalendar(from data: NSDictionary?) -> [EarningsDayModel]?`

EarningsVC: `func getEarnings()`

EarningsVC: `func sortEarningsCalendar(by key: SortType)`

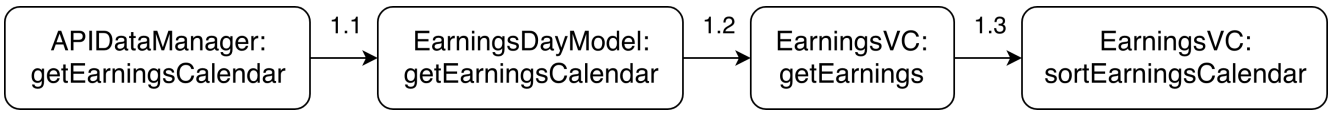

Рисунок 2 – Earnings Calendar flow

2. Watch List flow:

APIDataManager: `func getWatchList(completionHandler: @escaping ([WatchListTickerModel]?, ErrorType?) -> ())`

WatchListTickerModel: `func getWatchList(from data: NSDictionary?) -> [WatchListTickerModel]?`

WatchListVC: `func showWatchList()`

WatchListVC: `func sortWatchList(by key: SortType)`

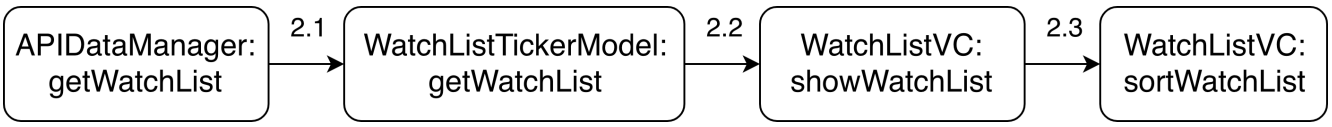

Рисунок 3 – Watch List flow

3. Search flow:

APIDataManager: `func getSearchList(completionHandler: @escaping ([SearchModel]?, ErrorType?) -> ())`

SearchModel: `func getSearchList(from data: NSDictionary?) -> [SearchModel]?`

SearchVC: `func showSearchList()`

SearchVC: `func saveUserTickers()`

APIDataManager: `func setUserTickers(tickers: [String], completion-Handler: @escaping (ErrorType?) -> ())`

Рисунок 4 – Search flow

4. Filter flow:

FilterModel: `func getFilters() -> [FilterModel]?`

FilterVC: `func showFilters()`

FilterVC: `func saveUserFilters()`

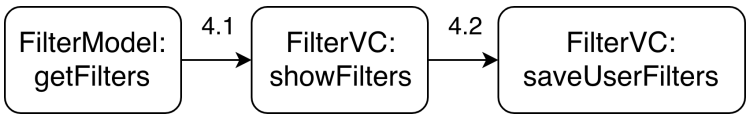

Рисунок 5 – Filter flow

Стратегия конфигурационного тестирования

Приложение будет протестировано на различных устройствах, имеющих различную версию операционной системы и разрешение экрана.

Стратегия аттестационного тестирования

В ходе аттестационное тестирования будет протестирована работоспособность приложения и его возможность осуществлять заявленный функционал. Аттестационные тесты покрывают ранее перечисленные функциональные требования. Аттестационное тестирование происходит мануальным методом.

Описание тестов

Корректность возвращаемых данных

Под корректной работой сервера подразумевается его доступность, а также отсутствие ошибок в логике работы сервера. Кроме того, сервер должен возвращать валидный JSON (см. ниже), коды ответа сервера должны быть в диапазоне 200-399.

Пример валидного JSON-файла со списком избранных компаний пользователя:

```
{
    "tickers": [
        {
            "date": "2018-11-19 16:55:09.589",
            "earning_time": "2018-11-19 16:55:09.589",
            "ticker": "AAON",
            "prediction": "5"
        },
        {
            "date": "2018-10-18 15:45:02.234",
```

```
            "earning_time": "2018-10-18 15:45:02.234",
            "ticker": "AAPL",
            "prediction": "5"
        }
    ]
}
```

Валидность JSON-файла можно проверить при помощи сервиса-валидатора (например, jsonlint.com).

Описание модульных тестов

Тест 1.

Модуль: APIDataManager.

Метод: `func getSearchList(completionHandler: @escaping ([SearchModel]?, ErrorType?) -> ())`

Тип: Негативный.

Описание: Тест проверяет, что метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Подключение к Интернет отсутствует.

Ожидаемый результат: Метод возвращает ошибку networkError.

Тест 2.

Модуль: APIDataManager.

Метод: `func getSearchList(completionHandler: @escaping ([SearchModel]?, ErrorType?) -> ())`

Тип: Негативный.

Описание: Тест проверяет, что метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Сервер возвращает ошибку 4xx-5xx.

Ожидаемый результат: Метод возвращает ошибку serverError.

Тест 3.

Модуль: APIDataManager.

Метод: `func getSearchList(completionHandler: @escaping ([SearchModel]?, ErrorType?) -> ())`

Тип: Негативный.

Описание: Тест проверяет, что метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Сервер возвращает несоответствующий формату JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает ошибку jsonError.

Тест 4.

Модуль: APIDataManager.

Метод: `func getSearchList(completionHandler: @escaping ([SearchModel]?, ErrorType?) -> ())`

Тип: Позитивный.

Описание: Тест проверяет, что метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Подключение к Интернет присутствует, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает ошибку = nil.

Тест 5.

Модуль: APIDataManager.

Метод: `func setUserTickers(tickers: [String], completionHandler: @escaping (ErrorType?) -> ())`

Тип: Негативный.

Описание: Тест проверяет, что метод принимает массив избранных компаний пользователя. Метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Метод принимает непустой массив избранных компаний пользователя. Подключение к Интернет отсутствует.

Ожидаемый результат: Метод возвращает ошибку networkError.

Тест 6.

Модуль: APIDataManager.

Метод: `func setUserTickers(tickers: [String], completionHandler: @escaping (ErrorType?) -> ())`

Тип: Негативный.

Описание: Тест проверяет, что метод принимает массив избранных компаний пользователя. Метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Метод принимает непустой массив избранных компаний пользователя. Сервер возвращает ошибку 4xx-5xx.

Ожидаемый результат: Метод возвращает ошибку serverError.

Тест 7.

Модуль: APIDataManager.

Метод: `func setUserTickers(tickers: [String], completionHandler: @escaping (ErrorType?) -> ())`

Тип: Негативный.

Описание: Тест проверяет, что метод принимает массив избранных компаний пользователя. Метод возвращает ошибку в случае отсутствия подключения к Ин-

тернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Метод принимает непустой массив избранных компаний пользователя. Сервер возвращает несоответствующий формату JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает ошибку jsonError.

Тест 8.

Модуль: APIDataManager.

Метод: `func setUserTickers(tickers: [String], completionHandler: @escaping (ErrorType?) -> ())`

Тип: Позитивный.

Описание: Тест проверяет, что метод принимает массив избранных компаний пользователя. Метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Метод принимает непустой массив избранных компаний пользователя. Подключение к Интернет присутствует, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает ошибку = nil.

Тест 9.

Модуль: APIDataManager.

Метод: `func setUserTickers(tickers: [String], completionHandler: @escaping (ErrorType?) -> ())`

Тип: Краевой.

Описание: Тест проверяет, что метод принимает массив избранных компаний пользователя. Метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Метод принимает пустой массив избранных компаний пользователя. Подключение к Интернет присутствует, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает ошибку = nil.

Тест 10.

Модуль: APIDataManager.

Метод: `func getWatchList(completionHandler: @escaping ([WatchList-TickerModel]?, ErrorType?) -> ())`

Тип: Негативный.

Описание: Тест проверяет, что метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Подключение к Интернет отсутствует.

Ожидаемый результат: Метод возвращает ошибку networkError.

Тест 11.

Модуль: APIDataManager.

Метод: `func getWatchList(completionHandler: @escaping ([WatchList-TickerModel]?, ErrorType?) -> ())`

Тип: Негативный.

Описание: Тест проверяет, что метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Сервер возвращает ошибку 4xx-5xx.

Ожидаемый результат: Метод возвращает ошибку serverError.

Тест 12.

Модуль: APIDataManager.

Метод: `func getWatchList(completionHandler: @escaping ([WatchList-TickerModel]?, ErrorType?) -> ())`

Тип: Негативный.

Описание: Тест проверяет, что метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Сервер возвращает несоответствующий формату JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает ошибку jsonError.

Тест 13.

Модуль: APIDataManager.

Метод: `func getWatchList(completionHandler: @escaping ([WatchList-TickerModel]?, ErrorType?) -> ())`

Тип: Позитивный.

Описание: Тест проверяет, что метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Подключение к Интернет присутствует, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает ошибку = nil.

Тест 14.

Модуль: APIDataManager.

Метод: `func getEarningsCalendar(completionHandler: @escaping ([EarningsDayModel]?, ErrorType?) -> ())`

Тип: Негативный.

Описание: Тест проверяет, что метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полу-

ченного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Подключение к Интернет отсутствует.

Ожидаемый результат: Метод возвращает ошибку networkError.

Тест 15.

Модуль: APIDataManager.

Метод: `func getEarningsCalendar(completionHandler: @escaping ([EarningsDayModel]?, ErrorType?) -> ())`

Тип: Негативный.

Описание: Тест проверяет, что метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Сервер возвращает ошибку 4xx-5xx.

Ожидаемый результат: Метод возвращает ошибку serverError.

Тест 16.

Модуль: APIDataManager.

Метод: `func getEarningsCalendar(completionHandler: @escaping ([EarningsDayModel]?, ErrorType?) -> ())`

Тип: Негативный.

Описание: Тест проверяет, что метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Сервер возвращает несоответствующий формату JSON-файл (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает ошибку jsonError.

Тест 17.

Модуль: APIDataManager.

Метод: `func getEarningsCalendar(completionHandler: @escaping ([EarningsDayModel]?, ErrorType?) -> ())`

Тип: Позитивный.

Описание: Тест проверяет, что метод возвращает ошибку в случае отсутствия подключения к Интернет, ошибки сервера, неверного формата JSON-файла, полученного с сервера. В случае успешного выполнения запроса, метод возвращает ошибку = nil.

Исходные данные: Подключение к Интернет присутствует, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает ошибку = nil.

Тест 18.

Модуль: EarningsVC.

Метод: `func sortEarningsCalendar(by key: SortType)`

Тип: Позитивный.

Описание: Тест проверяет, что метод сортирует таблицу со списком компаний и курсом акций в зависимости от ключа сортировки (по имени, дате, цене или предсказанию).

Алгоритм: Имеется неотсортированный массив данных и несколько отсортированных (эталонных) массив тех же данных для каждого ключа сортировки. Применяем функцию сортировки данных с одним из ключей сортировки к неотсортированному массиву данных. Сравниваем полученный массив с эталонным. Если массивы совпадают, то метод отработал корректно, иначе – некорректно.

Исходные данные: Неотсортированный массив данных, массив данных, отсортированный по имени, ключ сортировки = tickerName.

Ожидаемый результат: Массив, отсортированный в результате работы метода, совпадает с эталонным массивом для ключа tickerName.

Тест 19.

Модуль: EarningsVC.

Метод: `func sortEarningsCalendar(by key: SortType)`

Тип: Позитивный.

Описание: Тест проверяет, что метод сортирует таблицу со списком компаний и курсом акций в зависимости от ключа сортировки (по имени, дате, цене или предсказанию).

Алгоритм: Имеется неотсортированный массив данных и несколько отсортированных (эталонных) массив тех же данных для каждого ключа сортировки. Применяем функцию сортировки данных с одним из ключей сортировки к неотсортированному массиву данных. Сравниваем полученный массив с эталонным. Если массивы совпадают, то метод отработал корректно, иначе – некорректно.

Исходные данные: Неотсортированный массив данных, массив данных, отсортированный по дате, ключ сортировки = earningDate.

Ожидаемый результат: Массив, отсортированный в результате работы метода, совпадает с эталонным массивом для ключа earningDate.

Тест 20.

Модуль: EarningsVC.

Метод: `func sortEarningsCalendar(by key: SortType)`

Тип: Позитивный.

Описание: Тест проверяет, что метод сортирует таблицу со списком компаний и курсом акций в зависимости от ключа сортировки (по имени, дате, цене или предсказанию).

Алгоритм: Имеется неотсортированный массив данных и несколько отсортированных (эталонных) массив тех же данных для каждого ключа сортировки. Применяем функцию сортировки данных с одним из ключей сортировки к неотсортированному массиву данных. Сравниваем полученный массив с эталонным. Если массивы совпадают, то метод отработал корректно, иначе – некорректно.

Исходные данные: Неотсортированный массив данных, массив данных, отсортированный по цене, ключ сортировки = priceChanged.

Ожидаемый результат: Массив, отсортированный в результате работы метода, совпадает с эталонным массивом для ключа priceChanged.

Тест 21.

Модуль: EarningsVC.

Метод: `func sortEarningsCalendar(by key: SortType)`

Тип: Позитивный.

Описание: Тест проверяет, что метод сортирует таблицу со списком компаний и курсом акций в зависимости от ключа сортировки (по имени, дате, цене или предсказанию).

Алгоритм: Имеется неотсортированный массив данных и несколько отсортированных (эталонных) массив тех же данных для каждого ключа сортировки. Применяем функцию сортировки данных с одним из ключей сортировки к неотсортированному массиву данных. Сравниваем полученный массив с эталонным. Если массивы совпадают, то метод отработал корректно, иначе – некорректно.

Исходные данные: Неотсортированный массив данных, массив данных, отсортированный по предсказанию, ключ сортировки = prediction.

Ожидаемый результат: Массив, отсортированный в результате работы метода, совпадает с эталонным массивом для ключа prediction.

Тест 22.

Модуль: WatchListVC.

Метод: `func sortWatchList(by key: SortType)`

Тип: Позитивный.

Описание: Тест проверяет, что метод сортирует таблицу со списком избранных компаний пользователя и курсом акций в зависимости от ключа сортировки (по имени, дате, цене или предсказанию).

Алгоритм: Имеется неотсортированный массив данных и несколько отсортированных (эталонных) массив тех же данных для каждого ключа сортировки. Применяем функцию сортировки данных с одним из ключей сортировки к неот-

сортированному массиву данных. Сравниваем полученный массив с эталонным. Если массивы совпадают, то метод отработал корректно, иначе – некорректно.

Исходные данные: Неотсортированный массив данных, массив данных, отсортированный по имени, ключ сортировки = tickerName.

Ожидаемый результат: Массив, отсортированный в результате работы метода, совпадает с эталонным массивом для ключа tickerName.

Тест 23.

Модуль: WatchListVC.

Метод: `func sortWatchList(by key: SortType)`

Тип: Позитивный.

Описание: Тест проверяет, что метод сортирует таблицу со списком избранных компаний пользователя и курсом акций в зависимости от ключа сортировки (по имени, дате, цене или предсказанию).

Алгоритм: Имеется неотсортированный массив данных и несколько отсортированных (эталонных) массив тех же данных для каждого ключа сортировки. Применяем функцию сортировки данных с одним из ключей сортировки к неотсортированному массиву данных. Сравниваем полученный массив с эталонным. Если массивы совпадают, то метод отработал корректно, иначе – некорректно.

Исходные данные: Неотсортированный массив данных, массив данных, отсортированный по дате, ключ сортировки = earningDate.

Ожидаемый результат: Массив, отсортированный в результате работы метода, совпадает с эталонным массивом для ключа earningDate.

Тест 24.

Модуль: WatchListVC.

Метод: `func sortWatchList(by key: SortType)`

Тип: Позитивный.

Описание: Тест проверяет, что метод сортирует таблицу со списком избранных компаний пользователя и курсом акций в зависимости от ключа сортировки (по имени, дате, цене или предсказанию).

Алгоритм: Имеется неотсортированный массив данных и несколько отсортированных (эталонных) массив тех же данных для каждого ключа сортировки. Применяем функцию сортировки данных с одним из ключей сортировки к неотсортированному массиву данных. Сравниваем полученный массив с эталонным. Если массивы совпадают, то метод отработал корректно, иначе – некорректно.

Исходные данные: Неотсортированный массив данных, массив данных, отсортированный по цене, ключ сортировки = priceChanged.

Ожидаемый результат: Массив, отсортированный в результате работы метода, совпадает с эталонным массивом для ключа priceChanged.

Тест 25.

Модуль: WatchListVC.

Метод: `func sortWatchList(by key: SortType)`

Тип: Позитивный.

Описание: Тест проверяет, что метод сортирует таблицу со списком избранных компаний пользователя и курсом акций в зависимости от ключа сортировки (по имени, дате, цене или предсказанию).

Алгоритм: Имеется неотсортированный массив данных и несколько отсортированных (эталонных) массив тех же данных для каждого ключа сортировки. Применяем функцию сортировки данных с одним из ключей сортировки к неотсортированному массиву данных. Сравниваем полученный массив с эталонным. Если массивы совпадают, то метод отработал корректно, иначе – некорректно.

Исходные данные: Неотсортированный массив данных, массив данных, отсортированный по предсказанию, ключ сортировки = prediction.

Ожидаемый результат: Массив, отсортированный в результате работы метода, совпадает с эталонным массивом для ключа prediction.

Тест 26.

Модуль: EarningsDayModel.

Метод: `func getEarningsCalendar(from data: NSDictionary?) -> [EarningsDayModel]?`

Тип: Позитивный.

Описание: Тест проверяет, что метод получает Earnings Calendar в формате JSON, преобразует его в EarningsDayModel и возвращает массив [EarningsDayModel] или nil, в случае ошибки.

Алгоритм: В качестве входных данных для метода передаём JSON-файл с известным количеством элементов. Если количество элементов в возвращаемом методом массиве совпадает с количеством элементов в JSON-файле, то метод отработал корректно. Если количество элементов различается или метод возвращает nil, то метод работает некорректно.

Исходные данные: Валидный JSON-файл в нужном формате с известным количеством элементов (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает массив EarningsDayModel с количеством элементов, равным количеству элементов в JSON-файле.

Тест 27.

Модуль: EarningsDayModel.

Метод: `func getEarningsCalendar(from data: NSDictionary?) -> [EarningsDayModel]?`

Тип: Негативный.

Описание: Тест проверяет, что метод получает Earnings Calendar в формате JSON, преобразует его в EarningsDayModel и возвращает массив [EarningsDayModel] или nil, в случае ошибки.

Алгоритм: В качестве входных данных для метода передаём JSON-файл с известным количеством элементов. Если количество элементов в возвращаемом методом массиве совпадает с количеством элементов в JSON-файле, то метод отработал корректно. Если количество элементов различается или метод возвращает nil, то метод работает некорректно.

Исходные данные: Несоответствующий формату JSON-файл с известным количеством элементов (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает nil.

Тест 28.

Модуль: WatchListTickerModel.

Метод: `func getWatchList(from data: NSDictionary?) -> [WatchListTickerModel]?`

Тип: Позитивный.

Описание: Тест проверяет, что метод получает Watch List в формате JSON, преобразует его в WatchListTickerModel и возвращает массив [WatchListTickerModel] или nil, в случае ошибки.

Алгоритм: В качестве входных данных для метода передаём JSON-файл с известным количеством элементов. Если количество элементов в возвращаемом методом массиве совпадает с количеством элементов в JSON-файле, то метод отработал корректно. Если количество элементов различается или метод возвращает nil, то метод работает некорректно.

Исходные данные: Валидный JSON-файл в нужном формате с известным количеством элементов (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает массив WatchListTickerModel с количеством элементов, равным количеству элементов в JSON-файле.

Тест 29.

Модуль: WatchListTickerModel.

Метод: `func getWatchList(from data: NSDictionary?) -> [WatchListTickerModel]?`

Тип: Негативный.

Описание: Тест проверяет, что метод получает Watch List в формате JSON, преобразует его в WatchListTickerModel и возвращает массив [WatchListTickerModel] или nil, в случае ошибки.

Алгоритм: В качестве входных данных для метода передаём JSON-файл с известным количеством элементов. Если количество элементов в возвращаемом методом массиве совпадает с количеством элементов в JSON-файле, то метод отработал корректно. Если количество элементов различается или метод возвращает nil, то метод работает некорректно.

Исходные данные: Несоответствующий формату JSON-файл с известным количеством элементов (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает nil.

Тест 30.

Модуль: SearchModel.

Метод: `func getSearchList(from data: NSDictionary?) -> [SearchModel]?`

Тип: Позитивный.

Описание: Тест проверяет, что метод получает список всех компаний в формате JSON, преобразует его в SearchModel и возвращает массив [SearchModel] или nil, в случае ошибки.

Алгоритм: В качестве входных данных для метода передаём JSON-файл с известным количеством элементов. Если количество элементов в возвращаемом методом массиве совпадает с количеством элементов в JSON-файле, то метод отработал корректно. Если количество элементов различается или метод возвращает nil, то метод работает некорректно.

Исходные данные: Валидный JSON-файл в нужном формате с известным количеством элементов (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает массив SearchModel с количеством элементов, равным количеству элементов в JSON-файле.

Тест 31.

Модуль: SearchModel.

Метод: `func getSearchList(from data: NSDictionary?) -> [SearchModel]?`

Тип: Негативный.

Описание: Тест проверяет, что метод получает список всех компаний в формате JSON, преобразует его в SearchModel и возвращает массив [SearchModel] или nil, в случае ошибки.

Алгоритм: В качестве входных данных для метода передаём JSON-файл с известным количеством элементов. Если количество элементов в возвращаемом методом массиве совпадает с количеством элементов в JSON-файле, то метод отработал корректно. Если количество элементов различается или метод возвращает nil, то метод работает некорректно.

Исходные данные: Несоответствующий формату JSON-файл с известным количеством элементов (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает nil.

Тест 32.

Модуль: FilterModel.

Метод: `func getFilters() -> [FilterModel]?`

Тип: Позитивный.

Описание: Тест проверяет, что метод работает со списком фильтров в формате JSON, преобразует его в FilterModel и возвращает массив [FilterModel] или nil, в случае ошибки.

Алгоритм: Метод работает с JSON-файлом с известным количеством элементов. Если количество элементов в возвращаемом методе массиве совпадает с количеством элементов в JSON-файле, то метод отработал корректно. Если количество элементов различается или метод возвращает nil, то метод работает некорректно.

Исходные данные: Валидный JSON-файл в нужном формате с известным количеством элементов (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает массив FilterModel с количеством элементов равным количеству элементов в JSON-файле.

Тест 33.

Модуль: FilterModel.

Метод: `func getFilters() -> [FilterModel]?`

Тип: Негативный.

Описание: Тест проверяет, что метод работает со списком фильтров в формате JSON, преобразует его в FilterModel и возвращает массив [FilterModel] или nil, в случае ошибки.

Алгоритм: Метод работает с JSON-файлом с известным количеством элементов. Если количество элементов в возвращаемом методе массиве совпадает с количеством элементов в JSON-файле, то метод отработал корректно. Если количество элементов различается или метод возвращает nil, то метод работает некорректно.

Исходные данные: Несоответствующий формату JSON-файл с известным количеством элементов (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: Метод возвращает nil.

Описание интеграционных тестов

Тест 34.

Flow: Earnings Calendar.

Методы:

1. APIDataManager: `func getEarningsCalendar(completionHandler: @escaping ([EarningsDayModel]?, ErrorType?) -> ())`

2. APIDataManager: `func getEarningsCalendar(completionHandler: @escaping ([EarningsDayModel]?, ErrorType?) -> ())`

Тип: Общий.

Этап интеграции: 1.1.

Исходные данные:

- Подключение к Интернет присутствует.
- Сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Описание:

- Получение Earnings Calendar с сервера.
- Парсинг JSON в модель.

Ожидаемый результат:

- Получение Earnings Calendar с сервера.
- В APIDataManager вернулся непустой массив [EarningsDayModel].

Тест 35.

Flow: Earnings Calendar.

Методы:

1. APIDataManager: `func getEarningsCalendar(completionHandler: @escaping ([EarningsDayModel]?, ErrorType?) -> ())`
2. EarningsDayModel: `func getEarningsCalendar(from data: NSDictionary?) -> [EarningsDayModel]?`
3. EarningsVC: `func getEarnings()`

Тип: Общий.

Этап интеграции: 1.2.

Исходные данные:

- Подключение к Интернет присутствует.
- Сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Описание:

- Получение Earnings Calendar с сервера.
- Парсинг JSON в модель.
- Отображение Earnings Calendar на экране.

Ожидаемый результат:

- В EarningsVC вернулся непустой массив [EarningsDayModel].
- В APIDataManager вернулся непустой массив [EarningsDayModel].
- На экране отображается Earnings Calendar.

Тест 36.

Flow: Earnings Calendar.

Методы:

1. APIDataManager: `func getEarningsCalendar(completionHandler: @escaping ([EarningsDayModel]?, ErrorType?) -> ())`

2. EarningsDayModel: `func getEarningsCalendar(from data: NSDictionary?) -> [EarningsDayModel]?`

3. EarningsVC: `func getEarnings()`

4. EarningsVC: `func sortEarningsCalendar(by key: SortType)`

Тип: Общий.

Этап интеграции: 1.3.

Исходные данные:

- Подключение к Интернет присутствует.
- Сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Описание:

- Получение Earnings Calendar с сервера.
- Парсинг JSON в модель.
- Отображение Earnings Calendar на экране.
- Сортировка Earnings Calendar.

Ожидаемый результат:

- В EarningsVC вернулся непустой массив [EarningsDayModel].
- В APIDataManager вернулся непустой массив [EarningsDayModel].
- На экране отображается Earnings Calendar.
- По нажатию на заголовки столбцов, Earnings Calendar на экране сортируется.

Тест 37.

Flow: Watch List.

Методы:

1. APIDataManager: `func getWatchList(completionHandler: @escaping ([WatchListTickerModel]?, ErrorType?) -> ())`

2. WatchListTickerModel: `func getWatchList(from data: NSDictionary?) -> [WatchListTickerModel]?`

Тип: Общий.

Этап интеграции: 2.1.

Исходные данные:

- Подключение к Интернет присутствует.
- Сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Описание:

- Получение Watch List с сервера.
- Парсинг JSON в модель.

Ожидаемый результат:

- В WatchListVC вернулся непустой массив [WatchListTickerModel].
- В APIDataManager вернулся непустой массив [WatchListTickerModel].

Тест 38.

Flow: Watch List.

Методы:

1. APIDataManager: `func getWatchList(completionHandler: @escaping ([WatchListTickerModel]?, ErrorType?) -> ())`

2. WatchListTickerModel: `func getWatchList(from data: NSDictionary?) -> [WatchListTickerModel]?`

3. WatchListVC: `func showWatchList()`

Тип: Общий.

Этап интеграции: 2.2.

Исходные данные:

- Подключение к Интернет присутствует.

- Сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Описание:

- Получение Watch List с сервера.
- Парсинг JSON в модель.
- Отображение Watch List на экране.

Ожидаемый результат:

- В WatchListVC вернулся непустой массив [WatchListTickerModel].
- В APIDataManager вернулся непустой массив [WatchListTickerModel].
- На экране отображается Watch List.

Тест 39.

Flow: Watch List.

Методы:

1. APIDataManager: `func getWatchList(completionHandler: @escaping ([WatchListTickerModel]?, ErrorType?) -> ())`
2. WatchListTickerModel: `func getWatchList(from data: NSDictionary?) -> [WatchListTickerModel]?`
3. WatchListVC: `func showWatchList()`
4. WatchListVC: `func sortWatchList(by key: SortType)`

Тип: Общий.

Этап интеграции: 2.3.

Исходные данные:

- Подключение к Интернет присутствует.
- Сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Описание:

- Получение Watch List с сервера.
- Парсинг JSON в модель.

- Отображение Watch List на экране.
- Сортировка Watch List.

Ожидаемый результат:

- В WatchListVC вернулся непустой массив [WatchListTickerModel].
- В APIDataManager вернулся непустой массив [WatchListTickerModel].
- На экране отображается Watch List.
- По нажатию на заголовки столбцов, Watch List на экране сортируется.

Тест 40.

Flow: Search.

Метод:

1. APIDataManager: `func getSearchList(completionHandler: @escaping ([SearchModel]?, ErrorType?) -> ())`

2. SearchModel: `func getSearchList(from data: NSDictionary?) -> [SearchModel]?`

Тип: Общий.

Этап интеграции: 3.1.

Исходные данные:

- Подключение к Интернет присутствует.
- Сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Описание:

- Получение списка всех компаний с сервера.
- Парсинг JSON в модель.

Ожидаемый результат:

- В SearchVC вернулся непустой массив [SearchModel].
- В APIDataManager вернулся непустой массив [SearchModel].

Тест 41.

Flow: Search.

Метод:

1. APIDataManager: `func getSearchList(completionHandler: @escaping ([SearchModel]?, ErrorType?) -> ())`

2. SearchModel: `func getSearchList(from data: NSDictionary?) -> [SearchModel]?`

3. SearchVC: `func showSearchList()`

Тип: Общий.

Этап интеграции: 3.2.

Исходные данные:

- Подключение к Интернет присутствует.
- Сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Описание:

- Получение списка всех компаний с сервера.
- Парсинг JSON в модель.
- Отображение списка всех компаний на экране.

Ожидаемый результат:

- В SearchVC вернулся непустой массив [SearchModel].
- В APIDataManager вернулся непустой массив [SearchModel].
- На экране отображается список всех компаний.

Тест 42.

Flow: Search.

Метод:

1. APIDataManager: `func getSearchList(completionHandler: @escaping ([SearchModel]?, ErrorType?) -> ())`

2. SearchModel: `func getSearchList(from data: NSDictionary?) -> [SearchModel]?`

3. SearchVC: `func showSearchList()`

4. SearchVC: `func saveUserTickers()`

Тип: Общий.

Этап интеграции: 3.3.

Исходные данные:

- Подключение к Интернет присутствует.
- Сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Описание:

- Получение списка всех компаний с сервера.
- Парсинг JSON в модель.
- Отображение списка всех компаний на экране.
- Сохранение списка избранных компаний в памяти устройства.

Ожидаемый результат:

- В SearchVC вернулся непустой массив [SearchModel].
- В APIDataManager вернулся непустой массив [SearchModel].
- На экране отображается список всех компаний.
- Список выбранных компаний сохраняется в памяти.

Тест 43.

Flow: Search.

Метод:

1. APIDataManager: `func getSearchList(completionHandler: @escaping ([SearchModel]?, ErrorType?) -> ())`

2. SearchModel: `func getSearchList(from data: NSDictionary?) -> [SearchModel]?`

3. SearchVC: `func showSearchList()`

4. SearchVC: `func saveUserTickers()`

5. APIDataManager: `func setUserTickers(tickers: [String], completionHandler: @escaping (ErrorType?) -> ())`

Тип: Общий.

Этап интеграции: 3.4.

Исходные данные:

- Подключение к Интернет присутствует.
- Сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Описание:

- Получение списка всех компаний с сервера.
- Парсинг JSON в модель.
- Отображение списка всех компаний на экране.
- Сохранение списка избранных компаний в памяти устройства.
- Отправка списка избранных компаний на сервер.

Ожидаемый результат:

- В SearchVC вернулся непустой массив [SearchModel].
- В APIDataManager вернулся непустой массив [SearchModel].
- На экране отображается список всех компаний.
- Список выбранных компаний сохраняется в памяти.
- Отправка запроса на сервер. При следующем переходе на экран Watch List, список избранных компаний будет изменен.

Тест 44.

Flow: Filter.

Методы:

1. FilterModel: `func getFilters() -> [FilterModel]?`
2. FilterVC: `func showFilters()`

Тип: Общий.

Этап интеграции: 4.1.

Исходные данные:

- Подключение к Интернет присутствует.

- Сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Описание:

- Парсинг JSON в модель.
- Отображение списка фильтров на экране.

Ожидаемый результат:

- В FilterVC вернулся непустой массив [FilterModel].
- На экране отображается список фильтров.

Тест 45.

Flow: Filter.

Методы:

1. FilterModel: `func getFilters() -> [FilterModel]?`
2. FilterVC: `func showFilters()`
3. FilterVC: `func saveUserFilters()`

Тип: Общий.

Этап интеграции: 4.2.

Исходные данные:

- Подключение к Интернет присутствует.
- Сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Описание:

- Парсинг JSON в модель.
- Отображение списка фильтров на экране.
- Сохранение списка установленных фильтров в памяти устройства.

Ожидаемый результат:

- В FilterVC вернулся непустой массив [FilterModel].
- На экране отображается список фильтров.

- Список установленных фильтров сохраняется в памяти. При следующем переходе на экран Earnings Calendar, на экране будут отображаться только компании, соответствующие установленным фильтрам.

Описание аттестационных тестов

Тест 46.

Экран Earnings Calendar.

Тип: Позитивный.

Функциональные требования: 1.1.

Описание: Проверка получения с сервера JSON-файла с календарём, его парсинга, отображения доступных недель и списка компаний с курсом акций.

Алгоритм: Запустить приложение. Перейти на экран Earnings Calendar.

Исходные данные: Подключение к Интернет присутствует, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: На экране отображается индикатор загрузки (спинер). После окончания загрузки, спинер исчезает, на экране отображается календарь, а также заполненный список компаний с курсом акций.

Тест 47.

Экран Earnings Calendar.

Тип: Негативный.

Функциональные требования: 1.1.

Описание: Проверка получения с сервера JSON-файла с календарём, его парсинга, отображения доступных недель и списка компаний с курсом акций.

Алгоритм: Запустить приложение. Перейти на экран Earnings Calendar.

Исходные данные: Подключение к Интернет присутствует, сервер недоступен (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: На экране отображается индикатор загрузки. После окончания загрузки, спинер исчезает, на экране отображается сообщение с ошибкой «Ошибка сервера».

Тест 48.

Экран Earnings Calendar.

Тип: Негативный.

Функциональные требования: 1.1

Описание: Проверка получения с сервера JSON-файла с календарём, его парсинга, отображения доступных недель и списка компаний с курсом акций

Алгоритм: Запустить приложение. Перейти на экран Earnings Calendar.

Исходные данные: Подключение к Интернет отсутствует.

Ожидаемый результат: На экране отображается индикатор загрузки. После окончания загрузки, спинер исчезает, на экране отображается сообщение с ошибкой «Сбой подключения к Интернет».

Тест 49.

Экран Earnings Calendar.

Тип: Позитивный.

Функциональные требования: 1.2.

Описание: Проверка переключения дней недели и обновления списка акций.

Алгоритм: Запустить приложение. Перейти на экран Earnings Calendar. Дождаться окончания загрузки данных с сервера.

Исходные данные: Пользователь нажимает на день недели N.

Ожидаемый результат: В таблице отображаются данные на день N.

Тест 50.

Экран Earnings Calendar.

Тип: Позитивный.

Функциональные требования: 1.3.

Описание: Проверка сортировки списка компаний по умолчанию.

Алгоритм: Запустить приложение. Перейти на экран Earnings Calendar. Дождаться окончания загрузки данных с сервера.

Ожидаемый результат: Данные в таблице отсортированы по ключу имени (от A до Z).

Тест 51.

Экран Earnings Calendar.

Тип: Позитивный.

Функциональные требования: 1.3.

Описание: Проверка сортировки списка компаний по различным параметрам (название, стоимость акций и прочее).

Алгоритм: Запустить приложение. Перейти на экран Earnings Calendar. Дождаться окончания загрузки данных с сервера.

Исходные данные: Пользователь нажимает на заголовок N одного из столбцов таблицы.

Ожидаемый результат: Данные в таблице отсортированы по ключу N (по возрастанию).

Тест 52.

Экран Earnings Calendar.

Тип: Позитивный.

Функциональные требования: 1.4.

Описание: Проверка сортировки списка компаний по различным параметрам (название, стоимость акций и прочее).

Алгоритм: Запустить приложение. Перейти на экран Earnings Calendar. Дождаться окончания загрузки данных с сервера.

Исходные данные: Пользователь два раза подряд нажимает на заголовок N одного из столбцов таблицы.

Ожидаемый результат: Данные в таблице отсортированы по ключу N (по убыванию).

Тест 53.

Экран Watch List.

Тип: Позитивный.

Функциональные требования: 2.1.

Описание: Проверка получения с сервера JSON-файла, парсинга и отображения списка акций избранных компаний.

Алгоритм: Запустить приложение. Перейти на экран Watch List

Исходные данные: Подключение к Интернет присутствует, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: На экране отображается индикатор загрузки. После окончания загрузки, спинер исчезает, на экране отображается список избранных компаний с курсом акций.

Тест 54.

Экран Watch List.

Тип: Негативный.

Функциональные требования: 2.1.

Описание: Проверка получения с сервера JSON-файла, парсинга и отображения списка акций избранных компаний.

Алгоритм: Запустить приложение. Перейти на экран Watch List

Исходные данные: Подключение к Интернет присутствует, сервер недоступен (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: На экране отображается индикатор загрузки. После окончания загрузки, спинер исчезает, на экране отображается сообщение с ошибкой «Ошибка сервера».

Тест 55.

Экран Watch List.

Тип: Негативный.

Функциональные требования: 2.1.

Описание: Проверка получения с сервера JSON-файла, парсинга и отображения списка акций избранных компаний.

Алгоритм: Запустить приложение. Перейти на экран Watch List.

Исходные данные: Подключение к Интернет отсутствует.

Ожидаемый результат: На экране отображается индикатор загрузки. После окончания загрузки, спинер исчезает, на экране отображается сообщение с ошибкой «Сбой подключения к Интернет».

Тест 56.

Экран Watch List.

Тип: Позитивный.

Функциональные требования: 2.2.

Описание: Проверка удаления компании из списка избранных

Алгоритм: Запустить приложение. Перейти на экран Watch List. Свайпом справа-налево по одной из строк таблицы избранных компаний удалить компанию из списка.

Исходные данные: Подключение к Интернет присутствует, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: На экране отображается индикатор загрузки. После окончания загрузки, спинер исчезает, компания исчезает из таблицы.

Тест 57.

Экран Watch List.

Тип: Негативный.

Функциональные требования: 2.2.

Описание: Проверка удаления компании из списка избранных

Алгоритм: Запустить приложение. Перейти на экран Watch List. Свайпом справа-налево по одной из строк таблицы избранных компаний удалить компанию из списка.

Исходные данные: Подключение к Интернет присутствует, сервер недоступен (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: На экране отображается индикатор загрузки. После окончания загрузки, спинер исчезает, на экране отображается сообщение с ошибкой «Ошибка сервера», компания остаётся в таблице.

Тест 58.

Экран Watch List.

Тип: Негативный.

Функциональные требования: 2.2.

Описание: Проверка удаления компании из списка избранных.

Алгоритм: Запустить приложение. Перейти на экран Watch List. Свайпом справа-налево по одной из строк таблицы избранных компаний удалить компанию из списка.

Исходные данные: Подключение к Интернет отсутствует.

Ожидаемый результат: На экране отображается индикатор загрузки. После окончания загрузки, спинер исчезает, на экране отображается сообщение с ошибкой «Сбой подключения к Интернет», компания остаётся в таблице.

Тест 59.

Экран Watch List.

Тип: Позитивный.

Функциональные требования: 2.3.

Описание: Проверка сортировки списка компаний по умолчанию.

Алгоритм: Запустить приложение. Перейти на экран Watch List. Дождаться окончания загрузки данных с сервера.

Ожидаемый результат: Данные в таблице отсортированы по ключу имени (от A до Z).

Тест 60.

Экран Watch List.

Тип: Позитивный.

Функциональные требования: 2.3.

Описание: Проверка сортировки списка компаний по различным параметрам (название, стоимость акций и прочее)

Алгоритм: Запустить приложение. Перейти на экран Watch List. Дождаться окончания загрузки данных с сервера.

Исходные данные: Пользователь нажимает на заголовок N одного из столбцов таблицы.

Ожидаемый результат: Данные в таблице отсортированы по ключу N.

Тест 61.

Экран Watch List.

Тип: Позитивный.

Функциональные требования: 2.3.

Описание: Проверка сортировки списка компаний по различным параметрам (название, стоимость акций и прочее).

Алгоритм: Запустить приложение. Перейти на экран Watch List. Дождаться окончания загрузки данных с сервера.

Исходные данные: Пользователь два раза подряд нажимает на заголовок N одного из столбцов таблицы.

Ожидаемый результат: Данные в таблице отсортированы по ключу N (по убыванию).

Тест 62.

Экран Filters.

Тип: Позитивный.

Функциональные требования: 3.1, 3.2.

Описание: Выбор нужного набора фильтров для экрана Earnings Calendar.

Алгоритм: Запустить приложение. Перейти на экран Filters. Установить фильтры. Перейти на экран Earnings Calendar. Дождаться окончания загрузки данных с сервера.

Ожидаемый результат: На экране Earnings Calendar отображаются данные только для компаний, удовлетворяющих фильтрам.

Тест 63.

Экран Search.

Тип: Позитивный.

Функциональные требования: 4.1.

Описание: Проверка получения с сервера JSON-файла, парсинга, отображения списка всех доступных компаний.

Алгоритм: Запустить приложение. Перейти на экран Search.

Исходные данные: Подключение к Интернет присутствует, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: На экране отображается индикатор загрузки. После окончания загрузки, спинер исчезает, на экране отображается список всех доступных компаний.

Тест 64.

Экран Search.

Тип: Негативный.

Функциональные требования: 4.1.

Описание: Проверка получения с сервера JSON-файла, парсинга, отображения списка всех доступных компаний.

Алгоритм: Запустить приложение. Перейти на экран Search.

Исходные данные: Подключение к Интернет присутствует, сервер недоступен (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: На экране отображается индикатор загрузки. После окончания загрузки, спинер исчезает, на экране отображается сообщение с ошибкой «Ошибка сервера».

Тест 65.

Экран Search.

Тип: Негативный.

Функциональные требования: 4.1.

Описание: Проверка получения с сервера JSON-файла, парсинга, отображения списка всех доступных компаний.

Алгоритм: Запустить приложение. Перейти на экран Search.

Исходные данные: Подключение к Интернет отсутствует.

Ожидаемый результат: На экране отображается индикатор загрузки. После окончания загрузки, спинер исчезает, на экране отображается сообщение с ошибкой «Сбой подключения к Интернет».

Тест 66.

Экран Search.

Тип: Позитивный.

Функциональные требования: 4.2, 4.3.

Описание: Изменение списка избранных компаний для экрана Watch List.

Алгоритм: Запустить приложение. Перейти на экран Search. Дождаться окончания загрузки данных с сервера. Установить список избранных компаний. Перейти на экран Watch List. Дождаться окончания загрузки данных с сервера.

Исходные данные: Подключение к Интернет присутствует, сервер работает корректно (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: На экране Watch List отображаются данные только для выбранных компаний.

Тест 67.

Экран Search.

Тип: Негативный.

Функциональные требования: 4.2, 4.3.

Описание: Изменение списка избранных компаний для экрана Watch List.

Алгоритм: Запустить приложение. Перейти на экран Search. Дождаться окончания загрузки данных с сервера. Установить список избранных компаний. Перейти на экран Watch List. Дождаться окончания загрузки данных с сервера.

Исходные данные: Подключение к Интернет присутствует, сервер недоступен (см. раздел «Корректность возвращаемых данных»).

Ожидаемый результат: На экране отображается индикатор загрузки. После окончания загрузки, спинер исчезает, на экране отображается сообщение с ошибкой «Ошибка сервера». Данные на экране Watch List не изменились.

Тест 68.

Экран Search.

Тип: Негативный.

Функциональные требования: 4.2, 4.3.

Описание: Изменение списка избранных компаний для экрана Watch List.

Алгоритм: Запустить приложение. Перейти на экран Search. Дождаться окончания загрузки данных с сервера. Установить список избранных компаний. Перейти на экран Watch List. Дождаться окончания загрузки данных с сервера.

Исходные данные: Подключение к Интернет отсутствует.

Ожидаемый результат: На экране отображается индикатор загрузки. После окончания загрузки, спинер исчезает, на экране отображается сообщение с ошибкой «Сбой подключения к Интернет». Данные на экране Watch List не изменились.

Описание конфигурационных тестов

Все тесты, описанные выше, будут тестироваться на следующих устройствах:

1. Устройство: Apple iPhone

Модель: 7

Версия ОС: iOS 12.1

Разрешение экрана: 1,334 x 750 (326 ppi)

2. Устройство: Apple iPhone

Модель: 7+

Версия ОС: iOS 11.4

Разрешение экрана: 1,920 x 1,080 (401 ppi)

3. Устройство: Apple iPhone

Модель: X

Версия ОС: iOS 12.0

Разрешение экрана: 2,436 x 1,125 (458 ppi)

Целью конфигурационного тестирования является проверка работы заявленного выше функционала на различных устройствах, имеющих разные версии опера-

ционной системы и разные разрешения экранов. Также необходимо убедиться в том, что вёрстка приложения соответствует дизайну.

Примеры реализации тестов

Пример реализации теста получения списка всех компаний с сервера:

```
    func testGetSearchList() {
        //given
        var responseError: ErrorType?

        //when
         dataManager?.getSearchList(completionHandler: { response-
Data, error in
            if error != nil {
                responseError = error
            }
        })

        //then
          XCTAssertNil(responseError, «error: failed when getting
search list»)
    }
```

Пример реализации теста парсинга JSON-файла с Earnings Calendar в модель:

```
    func testEarningsCalendarJsonMappingCorrect() {
        //given
        let countOfItems = 5
        var itemCount: Int? = 0

        //when
             if let path = Bundle(for: type(of: self)).path(for-
Resource: «EarningsCalendar», ofType: «json»),
            let data = NSData(contentsOfFile: path) {
            do {
```

```
                    if let responseData = try? JSONSerialization.json-
Object(with: data as Data, options: []) as? NSDictionary {
                        let itemsArray = EarningsDayModel.getEarnings-
Calendar(from: responseData )
                        itemCount = itemsArray?.count
                    }
                }
            }

            //then
                XCTAssertEqual(countOfItems, itemCount, «error: failed
parsed calendar data»)
        }
```

Отчет о проведении тестирования

№Теста	Дата	Результат (iPhone 7)	Результат (iPhone 7+)	Результат (iPhone X)	Отчёт об ошибке
Результаты модульного тестирования					
1	02.12.2018	Пройден	Пройден	Пройден	-
2	02.12.2018	Пройден	Пройден	Пройден	-
3	02.12.2018	Пройден	Пройден	Пройден	-
4	02.12.2018	Пройден	Пройден	Пройден	-
5	02.12.2018	Ошибка	Ошибка	Ошибка	Отчёт №1
6	02.12.2018	Пройден	Пройден	Пройден	-
7	02.12.2018	Пройден	Пройден	Пройден	-
8	02.12.2018	Пройден	Пройден	Пройден	-
9	02.12.2018	Пройден	Пройден	Пройден	-
10	02.12.2018	Пройден	Пройден	Пройден	-
11	02.12.2018	Пройден	Пройден	Пройден	-
12	02.12.2018	Пройден	Пройден	Пройден	-
13	02.12.2018	Пройден	Пройден	Пройден	-
14	02.12.2018	Пройден	Пройден	Пройден	-
15	02.12.2018	Пройден	Пройден	Пройден	-
16	02.12.2018	Пройден	Пройден	Пройден	-
17	02.12.2018	Пройден	Пройден	Пройден	-
18	02.12.2018	Пройден	Пройден	Пройден	-
19	02.12.2018	Пройден	Пройден	Пройден	-
20	02.12.2018	Пройден	Пройден	Пройден	-
21	02.12.2018	Пройден	Пройден	Пройден	-
22	02.12.2018	Пройден	Пройден	Пройден	-
23	02.12.2018	Пройден	Пройден	Пройден	-
24	02.12.2018	Пройден	Пройден	Пройден	-
25	02.12.2018	Пройден	Пройден	Пройден	-
26	02.12.2018	Пройден	Пройден	Пройден	-

27	02.12.2018	Пройден	Пройден	Пройден	-
28	02.12.2018	Пройден	Пройден	Пройден	-
29	02.12.2018	Пройден	Пройден	Пройден	-
30	02.12.2018	Ошибка	Ошибка	Ошибка	Отчёт №2
31	02.12.2018	Пройден	Пройден	Пройден	-
32	02.12.2018	Пройден	Пройден	Пройден	-
33	02.12.2018	Пройден	Пройден	Пройден	-
Результаты интеграционного тестирования					
34	02.12.2018	Пройден	Пройден	Пройден	-
35	02.12.2018	Пройден	Пройден	Пройден	-
36	02.12.2018	Пройден	Пройден	Пройден	-
37	02.12.2018	Пройден	Пройден	Пройден	-
38	02.12.2018	Пройден	Пройден	Пройден	-
39	02.12.2018	Пройден	Пройден	Пройден	-
40	02.12.2018	Пройден	Пройден	Пройден	-
41	02.12.2018	Пройден	Пройден	Пройден	-
42	02.12.2018	Пройден	Пройден	Пройден	-
43	02.12.2018	Пройден	Пройден	Пройден	-
44	02.12.2018	Пройден	Пройден	Пройден	-
45	02.12.2018	Пройден	Пройден	Пройден	-
Результаты аттестационного тестирования					
46	02.12.2018	Пройден	Пройден	Пройден	-
47	02.12.2018	Пройден	Пройден	Пройден	-
48	02.12.2018	Пройден	Пройден	Пройден	-
49	02.12.2018	Пройден	Пройден	Пройден	-
50	02.12.2018	Пройден	Пройден	Пройден	-
51	02.12.2018	Пройден	Пройден	Пройден	-
52	02.12.2018	Пройден	Пройден	Пройден	-
53	02.12.2018	Пройден	Пройден	Пройден	-
54	02.12.2018	Пройден	Пройден	Пройден	-

55	02.12.2018	Пройден	Пройден	Пройден	-
56	02.12.2018	Пройден	Пройден	Пройден	-
57	02.12.2018	Пройден	Пройден	Пройден	-
58	02.12.2018	Пройден	Пройден	Пройден	-
59	02.12.2018	Пройден	Пройден	Пройден	-
60	02.12.2018	Пройден	Пройден	Пройден	-
61	02.12.2018	Пройден	Пройден	Пройден	-
62	02.12.2018	Ошибка	Ошибка	Ошибка	Отчёт №3
63	02.12.2018	Пройден	Пройден	Пройден	-
64	02.12.2018	Пройден	Пройден	Пройден	-
65	02.12.2018	Пройден	Пройден	Пройден	-
66	02.12.2018	Пройден	Пройден	Пройден	-
67	02.12.2018	Пройден	Пройден	Пройден	-
68	02.12.2018	Пройден	Пройден	Пройден	-

Отчеты об ошибках

Отчёт №1.

Краткое описание: Ошибка при выполнении теста 5. Проверка получения ошибки при отсутствии подключения к Интернет.

Ожидаемый результат: Возврат ошибки networkError.

Фактический результат: Функция возвращает ошибку serverError.

Отчёт №2.

Краткое описание: Ошибка при выполнении теста 30. Проверка парсинга данных из JSON-файла в модель.

Ожидаемый результат: Возврат массива [SearchModel] с количеством элементов, равным количеству элементов в JSON-файле (10).

Фактический результат: Возврат массива [SearchModel] с количеством элементов, равным 9.

Отчёт №3.

Краткое описание: Ошибка при выполнении теста 66. Проверка получения и отображения данных.

Ожидаемый результат: Отображение индикатора загрузки (спинера) на экране во время загрузки данных с сервера.

Фактический результат: Индикатор загрузки отсутствует.

Заключение

Во время тестирования приложения было выявлено несколько ошибок, представленных в отчётах об ошибках №1–3. Среди найденных ошибок только одна является критической: ошибка парсинга данных из JSON-файла в модель. По результатам тестирования необходимо исправить найденные ошибки и повторно протестировать приложение.

Расчет тестового покрытия относительно исполняемого кода приложения проводится по формуле:

Covering = tested_length / code_length * 100%

- tested_length – это количество строк кода, покрытых тестами.
- code_length – это общее количество строк кода в приложении.

Covering = 1873 / 2136 * 100% = 88%