Министерство образования Республики Беларусь

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**КУРСОВОЙ ПРОЕКТ**

по дисциплине **«Модульное тестирование»**

на тему **«Электронный магазин для компьютерных комплектующих»**

Выполнил:

студент группы 10701116 Багиров Р.А.

Руководитель: Попова Ю.Б.

Минск 2018

Министерство образования Республики Беларусь

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту**

по дисциплине **«Модульное тестирование»**

на тему **«Электронный магазин для компьютерных комплектующих»**

Выполнил:

студент группы 10701116 Багиров Р.А.

Руководитель: Попова Ю.Б.

Минск 2018

Введение 4

1. Требования к программному продукту 6

1.1. Регистрация 6

1.2. Аутентификация 8

1.3. Администратор 8

1.4. Авторизованный пользователь 10

1.5. Неавторизованный пользователь 12

2. Проектирование программных модулей 14

2.1. Архитектура приложения 14

2.2. Описание классов 14

2.3. Схема базы данных 18

3. Реализация программных модулей 19

4. Проектирование модульных тестов 20

4.1. Построение ориентированных графов Мак-кейба 20

4.2. Формирование проходов 41

5. Реализация модульного тестирования 49

5.1. Реализация автоматизированных скриптов 49

5.2. Анализ результатов тестов 49

5.3. Отладка тестов 49

Заключение 50

Список использованных источников 51

ПРИЛОЖЕНИЕ А 52

Листинг исходных кодов 52

ПРИЛОЖЕНИЕ Б 95

Листинг автоматизированных скриптов 95

# Введение

Данный курсовой проект посвящен дисциплине «модульное тестирование». Тема курсового проекта – электронный магазин компьютерных комплектующих. Представленный проект предполагает проектирование и реализацию веб-сайта, а также модульное тестирование разработанных в ходе выполнения курсового проекта модулей.

Тестирование программного обеспечения — процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта.

На протяжении десятилетий развития разработки ПО к вопросам тестирования и обеспечения качества подходили очень и очень по-разному. Можно выделить несколько основных «эпох тестирования».

В 50–60-х годах прошлого века процесс тестирования был предельно формализован, отделён от процесса непосредственной разработки ПО и «математизирован». Фактически тестирование представляло собой скорее отладку программ. Существовала концепция т. н. «исчерпывающего тестирования» — проверки всех возможных путей выполнения кода со всеми возможными входными данными. Но очень скоро было выяснено, что исчерпывающее тестирование невозможно, т.к. количество возможных путей и входных данных очень велико, а также при таком подходе сложно найти проблемы в документации.

В 70-х годах фактически родились две фундаментальные идеи тестирования: тестирование сначала рассматривалось как процесс доказательства работоспособности программы в некоторых заданных условиях, а затем — строго наоборот: как процесс доказательства неработоспособности программы в некоторых заданных условиях. Это внутреннее противоречие не только не исчезло со временем, но и в наши дни многими авторами совершенно справедливо отмечается как две взаимодополняющие цели тестирования.

В 80-х годах произошло ключевое изменение места тестирования в разработке ПО: вместо одной из финальных стадий создания проекта тестирование стало применяться на протяжении всего цикла разработки, что позволило в очень многих случаях не только быстро обнаруживать и устранять проблемы, но даже предсказывать и предотвращать их появление.

В этот же период времени отмечено бурное развитие и формализация методологий тестирования и появление первых элементарных попыток автоматизировать тестирование.

В 90-х годах произошёл переход от тестирования как такового к более всеобъемлющему процессу, который называется «обеспечение качества», охватывает весь цикл разработки ПО и затрагивает процессы планирования, проектирования, создания и выполнения тест-кейсов, поддержку имеющихся тест-кейсов и тестовых окружений. Тестирование вышло на качественно новый уровень, который естественным образом привёл к дальнейшему развитию методологий, появлению достаточно мощных инструментов управления процессом тестирования и инструментальных средств автоматизации тестирования, уже вполне похожих на своих нынешних потомков.

В нулевые годы нынешнего века развитие тестирования продолжалось в контексте поиска всё новых и новых путей, методологий, техник и подходов к обеспечению качества. Серьёзное влияние на понимание тестирования оказало появление гибких методологий разработки и таких подходов, как «разработка под управлением тестированием (test-driven development, TDD)». Автоматизация тестирования уже воспринималась как обычная неотъемлемая часть большинства проектов, а также стали популярны идеи о том, что во главу процесса тестирования следует ставить не соответствие программы требованиям, а её способность предоставить конечному пользователю возможность эффективно решать свои задачи.

Если же отметить вкратце основные характеристики современного этапа, то получится примерно такой список: гибкие методологии и гибкое тестирование, глубокая интеграция с процессом разработки, широкое использование автоматизации, колоссальный набор технологий и инструментальных средств, кроссфункциональность команды (когда тестировщик и программист во многом могут выполнять работу друг друга).

В данном курсовом проекте необходимо осуществить модульное тестирование программных компонентов, которые будут созданы и использованы в разработанном веб-сайте.

Модульное тестирование направлено на проверку отдельных небольших частей приложения, которые (как правило) можно исследовать изолированно от других подобных частей. При выполнении данного тестирования могут проверяться отдельные функции или методы классов, сами классы, взаимодействие классов, небольшие библиотеки, отдельные части приложения. Часто данный вид тестирования реализуется с использованием специальных технологий и инструментальных средств автоматизации тестирования, значительно упрощающих и ускоряющих разработку соответствующих тест-кейсов.

# Требования к программному продукту

Система предназначена для заказа компьютерных комплектующих через Интернет.

В системе предусмотрено три роли пользователей:

* Администратор – зарегистрированный пользователь, имеющий доступ к серверной части приложения
* Пользователь – зарегистрированный пользователь, имеющий доступ только к клиентской части приложения
* Гость – незарегистрированный пользователь, имеющий доступ только к клиентской части приложения

Функции администратора: выйти, добавление категорий, удаление категорий, добавление товаров, удаление товаров, редактирование товаров, поиск пользователя по логину, просмотр списка пользователей (логин и история покупок), удаление пользователей, просмотр списка заказов, подтверждение заказа, просмотр всей истории заказов, удаление записи из истории заказов, очистка истории заказов.

Функции пользователя: выйти, просмотр истории покупок, удаление записи из истории, очистка истории, сортировка товаров, поиск товаров по критериям, просмотр подробно о товаре, добавление товара в корзину, просмотр корзины, удаление товара из корзины, оформление заказа (адрес, мобильный телефон).

Функции гостя: регистрация (логин и пароль), авторизация, сортировка товаров, поиск товаров по критериям, просмотр подробно о товаре, добавление товара в корзину, просмотр корзины, удаление товара из корзины, оформление заказа (адрес, мобильный телефон).

## Регистрация

|  |  |
| --- | --- |
| R1.1 | Для создания нового аккаунта используется страница регистрации (registration.php)    На странице содержатся следующие элементы:   * Поле для ввода логина * Поле для ввода почты * Поле для ввода пароля * Поле для подтверждения пароля * Кнопка для регистрации на сайте |
| R1.2 | Все поля («Логин», «Почта», «Пароль» и «Подтверждение пароля») являются обязательными для ввода. |
| R1.3 | Логин может содержать только буквы латинского алфавита в нижнем регистре, цифры, символы «\_» и «-». Допустимый размер логина – от 5 до 64 символов (включительно). |
| R1.4 | Если пользователь ввел логин, который не отвечает требованиям «R1.3», то после нажатия на кнопку «Зарегистрироваться» появится сообщение «Некорректно введен логин». |
| R1.5 | Если пользователь ввел логин, который уже занят, то при нажатии на кнопку «Зарегистрироваться» появится сообщение «Данный логин уже занят». |
| R1.6 | Почта должна соответствовать формату «a@b.c», где части «a» и «b» могут содержать только буквы латинского алфавита в нижнем регистре, цифры, символы «\_», «.» и «-». Часть «с» может содержать только буквы латинского алфавита в нижнем регистре и символ «.». Часть «с» может иметь размер не менее двух символов и не более шести.  Длина почты не должна превышать 128 символов. |
| R1.7 | Если пользователь ввел почту, которая не отвечает требованиям «R1.6», то после нажатия на кнопку «Зарегистрироваться» появится сообщение «Некорректно введен email». |
| R1.8 | Если пользователь ввел почту, которая привязана к другому зарегистрированному пользователю, то после нажатия на кнопку «Зарегистрироваться» появится сообщение «Пользователь с данной почтой уже существует». |
| R1.9 | Пароль может содержать любые символы. Допустимый размер пароля – от 5 до 32 символов. |
| R1.10 | Если пользователь ввел пароль, который не отвечает требованиям «R1.9», то после нажатия на кнопку «Зарегистрироваться» появится сообщение «Некорректно введен пароль». |
| R1.11 | Если пароли в полях «Пароль» и «Подтверждение пароля» не совпадают, то после нажатия на кнопку «Зарегистрироваться» появится сообщение «Неверно подтвержден пароль». |
| R1.12 | Если пользователь заполнил все поля корректно, то после нажатия на кнопку «Зарегистрироваться» появится новый аккаунт в системе и пользователя перенаправит на главную страницу (index.php). |

## Аутентификация

|  |  |
| --- | --- |
| R2.1 | Для аутентификации используется страница authentication.php    На странице содержатся следующие элементы:   * Поле для ввода логина * Поле для ввода пароля * Кнопка для входа |
| R2.2 | Все поля («Логин» и «Пароль») являются обязательными для ввода |
| R2.3 | Логин может содержать только буквы латинского алфавита в нижнем регистре, цифры, символы «\_» и «-». Допустимый размер логина – от 5 до 64 символов (включительно). |
| R2.4 | Если пользователь ввел логин, который не отвечает требованиям «R2.3», то при нажатии на кнопку «Войти» появится сообщение «Некорректно введен логин». |
| R2.5 | Пароль может содержать любые символы. Допустимый размер пароля – от 5 до 32 символов. |
| R2.6 | Если пользователь ввел пароль, который не отвечает требованиям «R2.5», то при нажатии на кнопку «Войти» появится сообщение «Некорректно введен пароль». |
| R2.7 | Если пользователь ввел несуществующий логин или не подходящий пароль, то при нажатии на кнопку «Войти» появится сообщение «Неправильный логин или пароль». |
| R2.8 | Если пользователь заполнил все поля корректно, то после нажатия на кнопку «Войти» пользователя перенаправит на главную страницу (index.php) или на страницу администратора (admin.php), если данный пользователь является администратором (имеет логин «admin»). |

## Администратор

**Общие сведения**

Данную часть системы может видеть только авторизованный администратор. Администратор может просматривать список пользователей, историю заказов выбранного пользователя. Администратор может удалить пользователя. Администратор может добавить или удалить категории. Администратор может добавлять, удалять и редактировать товары. Также администратор может просмотреть список заказов, подтвердить заказ, удалить заказ, просмотреть всю историю заказов, удалить запись из истории заказов, очистить историю заказов. Администратор может выйти.

|  |  |
| --- | --- |
| R3.1 | На главной странице администратора (admin.php) расположено понятное пользовательское меню в виде ссылок: |
| R3.2 | После нажатия на ссылку «Выйти» администратора перенаправляет на главную страницу (index.php). |
| R3.3 | После нажатия на ссылку «Список пользователей» открывается страница users.php |
| R3.4 | На странице users.php имеется поле для поиска по логину и кнопка «Поиск», а также таблица со всеми пользователями. |
| R3.5 | Поиск выдает всех пользователей, чей логин содержит подстроку, которая соответствует строке в поле для поиска. |
| R3.6 | Если поиск ничего не вернул, то на месте таблицы будет надпись «Пользователей с таким логином не найдено». |
| R3.7 | Таблица с пользователями содержит логин и две кнопки – «История заказов» и «Удалить». |
| R3.8 | После нажатия на кнопку «История заказов» напротив логина выбранного пользователя открывается страница userHistory.php с историей заказов выбранного пользователя. |
| R3.9 | На странице userHistory.php имеется таблица с историей заказов выбранного пользователя. |
| R3.10 | Напротив записи на странице userHistory.php имеется кнопка «Удалить», которая удалит запись, если на нее нажать. |
| R3.11 | После нажатия на кнопку «Удалить» напротив логина выбранного пользователя выбранный пользователь будет удален. |
| R3.12 | После нажатия на ссылку «Список заказов» открывается страница «order.php» с таблицей заказов. |
| R3.13 | Напротив записи на странице order.php имеется кнопка «Подтвердить» и «Удалить». |
| R3.14 | После нажатия на кнопку «Подтвердить» напротив заказа заказ подтверждается. |
| R3.15 | После нажатия на кнопку «Удалить» напротив заказа заказ удаляется. |
| R3.16 | После нажатия на ссылку «Список товаров» открывается страница adminGoods.php с таблицей товаров. |
| R3.17 | На странице adminGoods.php имеется поле для поиска по названию и кнопка «Поиск», а также таблица со всеми товарами. |
| R3.18 | Поиск выдает все товары, чье название содержит подстроку, которая соответствует строке в поле для поиска. |
| R3.19 | Если поиск ничего не вернул, то на месте таблицы будет надпись «Товаров с таким названием не найдено». |
| R3.20 | Таблица с товарами содержит название товара и две кнопки – «Изменить» и «Удалить». |
| R3.21 | После нажатия на кнопку «Изменить» открывается страница goodSet.php, которая содержит:   * Название товара * Цена товара * Описание * Изображение * Кнопка «Подтвердить» |
| R3.22 | Название товара не должно быть меньше пяти символов и не должно быть больше 128 символов. |
| R3.23 | Если администратор нажал на «подтвердить» и название товара не соответствует требованиям «R3.20», то появится сообщение «Некорректное название товара». |
| R3.24 | Цена товара должна быть целым неотрицательным числом, меньшим 2 147 483 648. |
| R3.25 | Если администратор нажал на «подтвердить» и цена товара не соответствует требованиям «R3.22», то появится сообщение «Некорректная цена товара». |
| R3.26 | Описание не является обязательным. Описание должно содержать не более 10000 символов. |
| R3.27 | Если администратор нажал на «подтвердить» и описание содержит более 10000 символов, то появится сообщение «Слишком большое описание». |
| R3.28 | Изображение является необязательным атрибутом. |
| R3.29 | Администратор может загружать изображения формата «.jpeg», «.jpg» и «.png». |
| R3.30 | Если все поля заполнены корректно, то после нажатия на кнопку «Подтвердить» принимаются изменения. |
| R3.31 | После нажатия на ссылку «Добавить товар» открывается страница addGoods.php со списком категорий. |
| R3.32 | После нажатия на одну из категорий открывается страница goodSet.php |
| R3.33 | Требования для страницы goodSet.php соответствуют требованиям «R3.19 – R3.28» |
| R3.34 | После нажатия на ссылку «Добавить категорию» открывается страница addCategories.php, которая содержит:   * Название категории * Изображение * Кнопка «Подтвердить» |
| R3.35 | Размер названия категории не должен быть меньше двух символов и не больше 64 символов. |
| R3.36 | Если администратор нажал на «подтвердить» и название категории не соответствует требованиям «R3.33», то появится сообщение «Некорректное название категории». |
| R3.37 | Категория (т.е. ее название) не должна дублироваться. |
| R3.38 | Если администратор нажал на «подтвердить» и название категории не соответствует требованиям «R3.37», то появится сообщение «Категория с таким названием уже существует». |
| R3.39 | Изображение является необязательным атрибутом. |
| R3.40 | Администратор может загружать изображения формата «.jpeg», «.jpg» и «.png». |
| R3.41 | Если все поля заполнены корректно, то после нажатия на кнопку «Подтвердить» создается новая категория. |
| R3.42 | После нажатия на ссылку «Удалить» открывается страница categories.php с таблицей категорий. |
| R3.43 | Напротив названия категории в таблице на странице categories.php имеется кнопка «Удалить». |
| R3.44 | После нажатия на кнопку «Удалить» выбранная категория будет удалена. Также будут «каскадно» удалены все товары, принадлежащие данной категории. |
| R3.45 | После нажатия на ссылку «История заказов» открывается страница history.php с таблицей историй заказов и кнопкой «очистить историю». |
| R3.46 | Напротив истории на странице history.php имеется кнопка «Удалить» |
| R3.47 | После нажатия на кнопку «Удалить» запись удаляется. |
| R3.48 | После нажатия на кнопку «Очистить историю» история будет полностью очищена. |

## Авторизованный пользователь

**Общие сведения**

Данную часть системы может видеть авторизованный пользователь. Пользователь может выбрать одну из категорий товара, просмотреть товары, которые имеются в базе данных, после чего он может добавить (удалить из) их в корзину и оформить заказ. Также пользователь может просматривать свою историю покупок и удалять из нее записи (или очистить всё).

|  |  |
| --- | --- |
| R4.1 | На главной странице index.php находится шапка и ссылки по категориям: |
| R4.2 | После нажатия на одну из категорий открывается страница goods.php c таблицей товаров, строкой поиска и кнопкой «Поиск», ссылками сортировки «по названию» и «по цене» и шапкой как в index.php. |
| R4.3 | Поиск выдает все товары, чьё название содержит подстроку, которая соответствует строке в поле для поиска. |
| R4.4 | Если поиск ничего не вернул, то на месте таблицы будет надпись «Товаров с таким названием не найдено». |
| R4.5 | Если нажать на ссылку сортировки «по названию», то все строки в таблице сортируются по названию. |
| R4.6 | Если еще раз нажать на ссылку сортировки «по названию», то все строки в таблице сортируются по названию в обратном порядке. |
| R4.7 | Если нажать на ссылку сортировки «по цене», то все строки в таблице сортируются по цене. |
| R4.8 | Если еще раз нажать на ссылку сортировки «по цене», то все строки в таблице сортируются по цене в обратном порядке. |
| R4.9 | После нажатия на название товара в таблице открывается страница good\_page.php c названием, ценой, изображением, описанием товара и кнопкой «Добавить в корзину» |
| R4.10 | После нажатия на кнопку «Добавить в корзину» товар добавляется в корзину. |
| R4.11 | “Пекарь” в шапке – это ссылка на главную страницу index.php |
| R4.12 | После нажатия на ссылку «История» открывается страница «userHistory.php» |
| R4.13 | На странице userHistory.php имеется таблица с историей заказов, кнопка «Очистить историю». |
| R4.14 | Напротив записи на странице userHistory.php имеется кнопка «Удалить», которая удалит запись, если на нее нажать. |
| R4.15 | После нажатия на кнопку «Очистить историю» история будет полностью очищена. |
| R4.16 | После нажатия на ссылку «Корзина» открывается страница basket.php. |
| R4.17 | Если корзина пуста, то на странице basket.php будет только ссылка «Корзина пуста. Добавьте что-нибудь» на главную страницу index.php. |
| R4.18 | Если корзина не пуста, то на странице будет таблица товаров с кнопками «Удалить» напротив каждой записи, информация об общей стоимости заказа и кнопка «Оформить заказ». |
| R4.19 | Если пользователь нажмет на кнопку «Удалить», то запись из корзины будет удалена |
| R4.20 | Если пользователь нажмет на кнопку «Оформить заказ», то откроется страница createOrder.php с полями для ввода адреса и телефона и кнопкой «Отправить». |
| R4.21 | Адрес должен содержать не менее семи символов и не более ста символов. |
| R4.22 | Если пользователь нажал на «Отправить» и адрес не соответствует требованиям «R4.20», то появится сообщение «Некорректный адрес». |
| R4.23 | Телефон должен состоять из семи цифр. |
| R4.24 | Если пользователь нажал на «Отправить» и телефон не соответствует требованиям «R4.22», то появится сообщение «Некорректный телефон». |
| R4.25 | Если пользователь заполнил все поля корректно, то заказ отправится на рассмотрение. |
| R4.26 | После нажатия на кнопку «Выход» пользователя перенаправляет на главную страницу index.php. |

## Неавторизованный пользователь

**Общие сведения**

Данную часть системы может видеть неавторизованный пользователь. Пользователь может выбрать одну из категорий товара, просмотреть товары, которые имеются в базе данных, после чего он может добавить (удалить из) их в корзину и оформить заказ. Неавторизованный пользователь может войти в систему или зарегистрироваться.

|  |  |
| --- | --- |
| R5.1 | На главной странице index.php находится шапка и ссылки по категориям: |
| R5.2 | После нажатия на одну из категорий открывается страница goods.php c таблицей товаров, строкой поиска и кнопкой «Поиск», ссылками сортировки «по названию» и «по цене» и шапкой как в index.php. |
| R5.3 | Поиск выдает все товары, чьё название содержит подстроку, которая соответствует строке в поле для поиска. |
| R5.4 | Если поиск ничего не вернул, то на месте таблицы будет надпись «Товаров с таким названием не найдено». |
| R5.5 | Если нажать на ссылку сортировки «по названию», то все строки в таблице сортируются по названию. |
| R5.6 | Если еще раз нажать на ссылку сортировки «по названию», то все строки в таблице сортируются по названию в обратном порядке. |
| R5.7 | Если нажать на ссылку сортировки «по цене», то все строки в таблице сортируются по цене. |
| R5.8 | Если еще раз нажать на ссылку сортировки «по цене», то все строки в таблице сортируются по цене в обратном порядке. |
| R5.9 | После нажатия на название товара в таблице открывается страница goodPage.php c названием, ценой, изображением, описанием товара и кнопкой «Добавить в корзину» |
| R5.10 | После нажатия на кнопку «Добавить в корзину» товар добавляется в корзину. |
| R5.11 | “Пекарь” в шапке – это ссылка на главную страницу index.php |
| R5.12 | После нажатия на ссылку «Корзина» открывается страница basket.php. |
| R5.13 | Если корзина пуста, то на странице basket.php будет только ссылка «Корзина пуста. Добавьте что-нибудь» на главную страницу index.php. |
| R5.14 | Если корзина не пуста, то на странице будет таблица товаров с кнопками «Удалить» напротив каждой записи, информация об общей стоимости заказа и кнопка «Оформить заказ». |
| R5.15 | Если пользователь нажмет на кнопку «Удалить», то запись из корзины будет удалена |
| R5.16 | Если пользователь нажмет на кнопку «Оформить заказ», то откроется страница createOrder.php с полями для ввода адреса и телефона и кнопкой «Отправить». |
| R5.17 | Адрес должен содержать не менее семи символов и не более ста символов. |
| R5.18 | Если пользователь нажал на «Отправить» и адрес не соответствует требованиям «R4.20», то появится сообщение «Некорректный адрес». |
| R5.19 | Телефон должен состоять из семи цифр. |
| R5.20 | Если пользователь нажал на «Отправить» и телефон не соответствует требованиям «R4.22», то появится сообщение «Некорректный телефон». |
| R5.21 | Если пользователь заполнил все поля корректно, то заказ отправится на рассмотрение. |
| R5.22 | После нажатия на ссылку «Регистрация» открывается страница registration.php |
| R5.23 | После нажатия на ссылку «Вход» открывается страница authentication.php |
| R5.24 | После нажатия на ссылку «Выход» пользователя перенаправляет на главную страницу index.php. |

# Проектирование программных модулей

## Архитектура приложения

Архитектура данного приложения представляет из себя несколько слоев. Самыми «нижними» слоями являются слой сущностей и слой DAO. Слой DAO предоставляет интерфейс для хранения сущностей (как правило, сущности хранятся в базе данных). Затем идет слой логики. Для проверки правильности данных, которые представил клиент, используются валидаторы. Слой «View» используется для оформления HTML страниц и рендера отдельных частей страницы. Верхний уровень ответственен за обработку запросов и ответ на принятые запросы.

## Описание классов

Прежде всего необходимо описать сущности, с которыми нужно иметь дело при создании простого электронного магазина.

Так как многие сущности неповторимы, то для описания их неповторимости можно создать им общего потомка «Identificational», который будет содержать одно свойство – «id», то есть уникальное число.

Магазин – это сервис для пользователя, поэтому первым делом надо описать сущность пользователя (User). Пользователь является уникальной сущностью и имеет три поля – логин (login), адрес электронной почты (email) и пароль (password).

Неотъемлемая часть любого магазина – товар. Однако товары бывают очень разными, а магазины в общем случае не узкоспециализированные, поэтому товар нужно, что называется, «разложить по полочкам» для удобства покупателя. Данными «полочками» могут являться категории товаров. Категории (Category) тоже будут уникальными. У категории есть поле «name», обозначающее название категории.

Теперь, когда у нас есть категории товаров, мы можем приступить к описанию товара (Product). Товар, как и вышеописанные сущности, является уникальным. Товар имеет название (name) и цену (price). Также товар обязательно должен принадлежать какой-либо категории, для этого есть поле «category». Покупателям в магазине будет куда проще делать выбор в сторону того или иного товара если у товаров будут краткие описания (description).

Пользователь может иметь желание купить не один, а несколько товаров, что, конечно же, положительно отразится на доходе нашего магазина. Для удобства пользователь может складывать товары в корзину. В нашем случае корзиной будет временное хранилище, в котором содержаться записи (CartRecord) с товаром и его количеством.

Пусть наш электронный магазин будет интернет-магазином. Появляется еще одна задача, стоящая перед магазином – доставка товаров покупателю. При оформлении покупки пользователь будет указывать свой номер и адрес, что позволит осуществить доставку. За хранение запросов на доставку будут отвечать уникальные записи запросов (OrderRecord), в которых будут указываться: пользователь(гser), товар (product), количество товара (count), адрес (address), телефон (phone) и дата осуществления заказа (orderDate).

Однако в действительности не все так радужно, как хотелось бы. Товар до пользователя может быть не доставлен по многим причинам, поэтому нам надо фиксировать подтверждение доставки заказа, чтобы быть уверенными, что продажа состоялась. После подтверждения запись запроса (OrderRecord) станет записью истории (HistoryRecord), в которой добавлена дата подтверждения заказа.

Сущности и их взаимоотношения показаны на рис. 2.2.1.

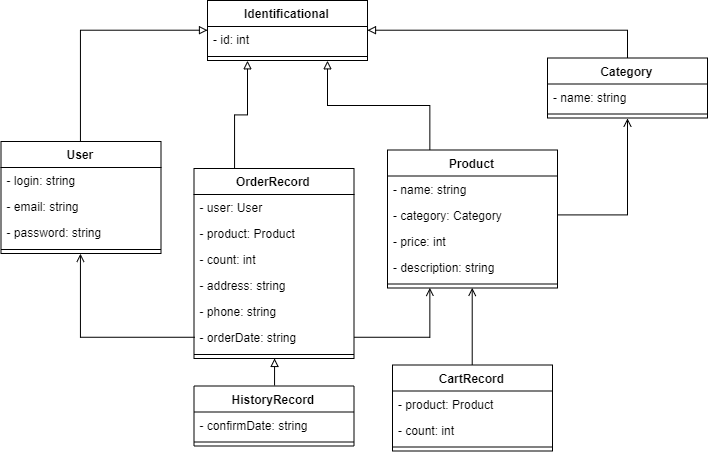


Рис. 2.2.1. – UML-диаграмма классов сущностей

Теперь, когда вопрос с сущностями решен, необходимо разработать интерфейс для доступа к данным сущностям.

Опишем интерфейс IDao с основными методами, которые должны быть реализованы во всех классах слоя «DAO».

Для доступа к корзине будем использовать класс CartDao, который реализует методы интерфейса IDao.

Остальные сущности будут храниться в базе данных. Опишем абстрактный класс DBDao, использующий базу данных в качестве хранилища для сущностей.

Также опишем абстрактный класс IdentificationalDao, унаследованный от DBDao, в котором будет описан и реализован метод delete для сущностей, обладающих полем «id».

От абстрактного класса IdentificationalDao унаследуем остальные классы (CategoryDao, UserDao, ProductDao, OrderRecordDao, HistoryRecordDao), которые предоставляют доступ к отсальным сущностям.

Диаграммы классов слоя «DAO» показаны на рис. 2.2.2

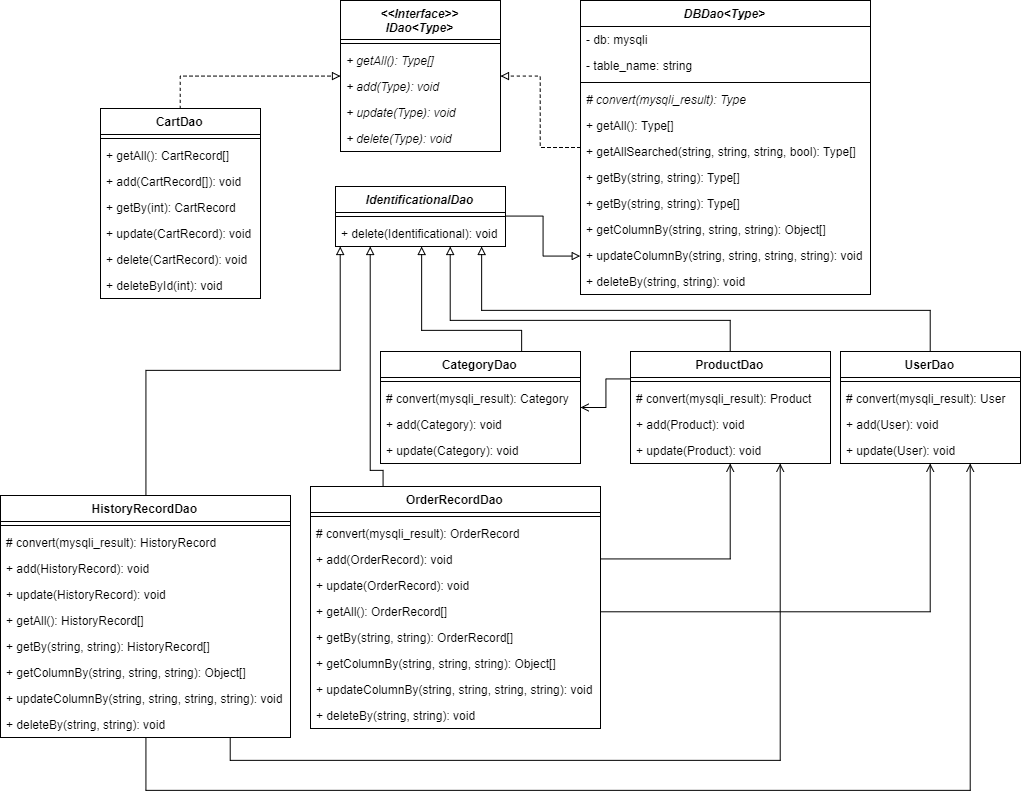


Рис. 2.2.2. – UML-диаграмма классов слоя DAO

Следующим по порядку идет слой логики. Опишем в этом слое классы для верификации полученных от пользователя данных. Для этого будут использованы классы AuthenticationVerifyer, CategoryVerifyer, OrderVerifyer, ProductVerifyer и RegistrationVerifyer. Диаграммы классов верификации представлены на рис. 2.2.3.

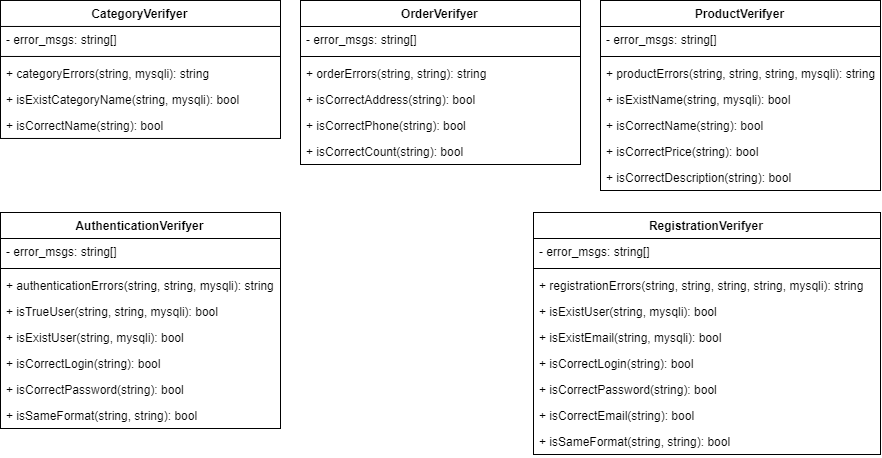


Рис. 2.2.3. – UML-диаграмма классов верификации

Также опишем класс Cashier, который имеет метод getTotalPrice для вычисления общей стоимости товаров в корзину. Класс FaceControl будет проверять, какой пользователь авторизован. Класс Authenticator будет ответственен за аутентификацию пользователя, а класс Registrator – за регистрацию. Класс URIResolver упростит работу с url строкой. На рис. 2.2.4 представлена диаграмма остальных классов слоя логики.

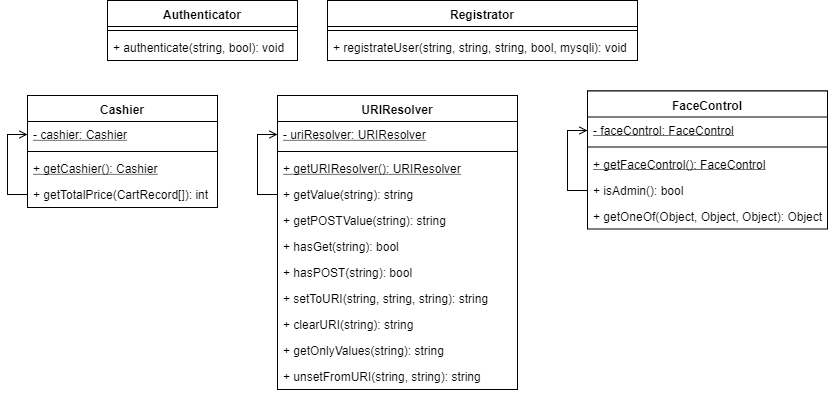


Рис. 2.2.4. – UML-диаграмма остальных классов слоя логики

Классы слоя «View» предназначены для рендера отдельных частей HTML-страниц. Диаграмма классов данного слоя представлена на рис 2.2.5.

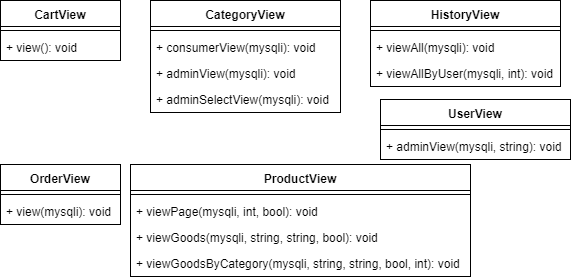


Рис. 2.2.5. – UML-диаграмма остальных классов слоя «View»

## Схема базы данных

Многие из описанных выше сущностей должны быть сохранены в определенном хранилище. В качестве такого хранилища будем использовать базу данных. Для сущности User используется таблица users. Для сущности Category – categories. Таблица products используется для сущности Product. Сущности OrderRecord и HistoryRecord используется таблица log – если поле confirm\_date – null – то данная запись является сущностью OrderRecord, иначе – HistoryRecord. Схема базы данных представлена на рис. 2.3.1.

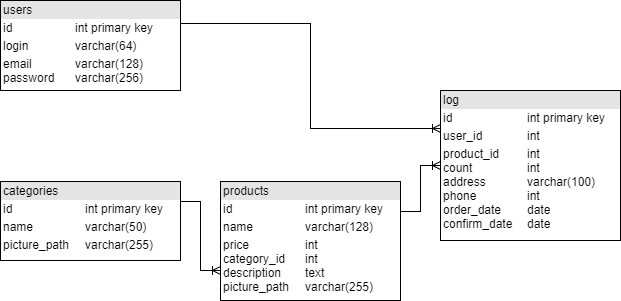


Рис. 2.3.1. – Общая схема базы данных

# Реализация программных модулей

Спроектированные программные модуль были реализованы с помощью скриптового языка PHP. Каждый класс расположен в отдельном php-файле.

Для слоев «сущности», «DAO» и «логика» была создана общая папка model. В папке model для слоя «сущности» была создана папка entity, для слоя «логика» - папка logic. Для утилитных скриптов и слоя «DAO» была создана папка utils, в которой есть папка dao для классов слоя «DAO».

В папке «view» расположены каскадные стили и классы слоя «View».

Для взаимодействия классов слоя «DAO» с базой данных используется встроенный класс mysqli.

При регистрации пользователь должен указать свой логин, адрес электронной почты и пароль. Но в базе данных будет храниться не сам пароль, а его хеш. Для генерации хеш-кода пароля используется встроенная функция password\_hash($password, $algo, $options=0) с параметром $algo, равным значению константы PASSWORD\_DEFAULT.

Во время аутентификации проверяется пароль, введенный пользователем, при помощи встроенной функции password\_verify($password, $hash).

Как только пользователь (или администратор) авторизовался, для него создается сессия $\_SESSION[“logged\_user”], в которой хранится его логин.

В качестве временного хранилища для записей корзины пользователя (не важно, авторизован он или нет) используется суперглобальный массив $\_SESSION.

Исходный код разработанных программных модулей можно посмотреть в Приложении А.

# Проектирование модульных тестов

## Построение ориентированных графов Мак-кейба

**CartDao::add**

Исходный код программного компонента:

**public function** add($record)  
{  
 **if** (!*is\_array*($record))  
 {  
 **throw new** InvalidArgumentException(**"not array"**);  
 }  
 **foreach** ($record **as** $rec)  
 {  
 **if** ($rec **instanceof** CartRecord)  
 {  
 $key = **"product\_id:"** . $rec->getProduct()->getId();  
 $count = $rec->getCount();  
 $\_SESSION[$key] = $count;  
 }  
 }  
}

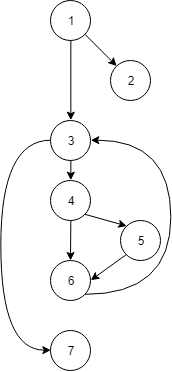


Рис. 4.1.1. – Граф Маккейба для CartDao::add

**CartDao::delete**

Исходный код программного компонента:

**public function** delete($record)  
{  
 **if** ($record **instanceof** CartRecord)  
 {  
 **unset**($\_SESSION[**"product\_id:"** . $record->getProduct()->getId()]);  
 }  
}

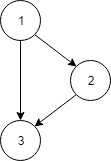


Рис. 4.1.2. – Граф Маккейба для CartDao::delete

**CartDao::deleteById**

Исходный код программного компонента:

**public function** deleteById($id)  
{  
 **if** (**isset**($\_SESSION[**"product\_id:"** . $id])) {  
 **unset**($\_SESSION[**"product\_id:"** . $id]);  
 }  
}

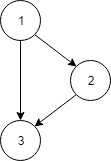


Рис. 4.1.3. – Граф Маккейба для CartDao::deleteById

**CategoryDao::add**

Исходный код программного компонента:

**public function** add($record)  
{  
 **if** ($record **instanceof** Category)  
 {  
 $name = $record->getName();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"INSERT INTO `**$table\_name**` (`name`)   
 VALUES ('**$name**')"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"category"**);  
 }  
}



Рис. 4.1.4. – Граф Маккейба для CategoryDao::add

**CategoryDao::update**

Исходный код программного компонента:

**public function** update($record)  
{  
 **if** ($record **instanceof** Category)  
 {  
 $id = $record->getId();  
 $name = $record->getName();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"UPDATE `**$table\_name**` SET `name` = '**$name**' WHERE `id` = '**$id**'"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"category"**);  
 }  
}



Рис. 4.1.5. – Граф Маккейба для CategoryDao::update

**DBDao::getAll**

Исходный код программного компонента:

**public function** getAll()  
{  
 $items = **array**();  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$this->**table\_name`"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
}

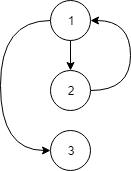


Рис. 4.1.6. – Граф Маккейба для DBDao::getAll

**DBDao::getAllSearched**

Исходный код программного компонента:

**public function** getAllSearched($column, $pattern=**"%"**, $sort\_column=**""**, $desc=**false**)  
{  
 $items = **array**();  
 **if** ($sort\_column == **""**)  
 {  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$this->**table\_name` WHERE `**$column**` LIKE '**$pattern**'"**);  
 }  
 **else if** ($desc)  
 {  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$this->**table\_name` WHERE `**$column**` LIKE '**$pattern**'  
 ORDER BY `**$sort\_column**` DESC"**);  
 }  
 **else** {  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$this->**table\_name` WHERE `**$column**` LIKE '**$pattern**'  
 ORDER BY `**$sort\_column**`"**);  
 }  
  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
}

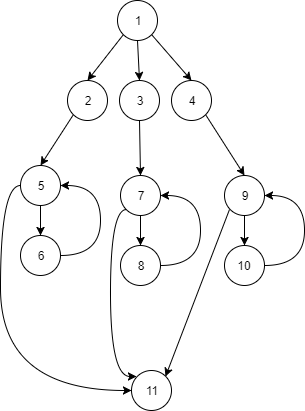


Рис. 4.1.7. – Граф Маккейба для DBDao::getAllSearched

**DBDao::getBy**

Исходный код программного компонента:

**public function** getBy($column\_name, $value)  
{  
 $items = **array**();  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$this->**table\_name` WHERE `**$column\_name**` = '**$value**'"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
}

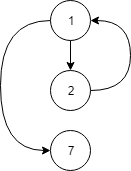


Рис. 4.1.8. – Граф Маккейба для DBDao::getBy

**DBDao::getColumnBy**

Исходный код программного компонента:

**public function** getColumnBy($column, $value, $get\_column)  
{  
 $items = **array**();  
 $records = $this->getDb()->query(**"SELECT `**$get\_column**` FROM `**$this->**table\_name` WHERE `**$column**` = '**$value**'"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $rec[$get\_column];  
 }  
 **return** $items;  
}

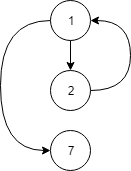


Рис. 4.1.9. – Граф Маккейба для DBDao::getColumnBy

**IdentificationalDao::delete**

Исходный код программного компонента:

**public function** delete($record)  
{  
 **if** ($record **instanceof** Identificational)  
 {  
 $id = $record->getId();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"DELETE FROM `**$table\_name**` WHERE `id` = '**$id**'"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"Non-id-able"**);  
 }  
}



Рис. 4.1.10. – Граф Маккейба для IdentificationalDao::delete

**HistoryRecordDao::add**

Исходный код программного компонента:

**public function** add($record)  
{  
 **if** ($record **instanceof** HistoryRecord)  
 {  
 $user\_id = ($record->getUser() == **null**) ? **null** : $record->getUser()->getId();  
 $product\_id = $record->getProduct()->getId();  
 $count = $record->getCount();  
 $address = $record->getAddress();  
 $phone = $record->getPhone();  
 $order\_date = $record->getOrderDate();  
 $confirm\_date = $record->getConfirmDate();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"INSERT INTO `**$table\_name**` (`user\_id`, `product\_id`, `count`, `address`,  
 `phone`, `order\_date`, `confirm\_date`)   
 VALUES ('**$user\_id**', '**$product\_id**', '**$count**', '**$address**', '**$phone**', '**$order\_date**', '**$confirm\_date**')"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"History record"**);  
 }  
}



Рис. 4.1.11. – Граф Маккейба для HistoryRecordDao::add

**HistoryRecordDao::update**

Исходный код программного компонента:

**public function** update($record)  
{  
 **if** ($record **instanceof** HistoryRecord)  
 {  
 $id = $record->getId();  
 $user\_id = ($record->getUser() == **null**) ? **null** : $record->getUser()->getId();  
 $product\_id = $record->getProduct()->getId();  
 $count = $record->getCount();  
 $address = $record->getAddress();  
 $phone = $record->getPhone();  
 $order\_date = $record->getOrderDate();  
 $confirm\_date = $record->getConfirmDate();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"UPDATE `**$table\_name**` SET `user\_id` = '**$user\_id**', `product\_id` = '**$product\_id**',  
 `count` = '**$count**', `address` = '**$address**', `phone` = '**$phone**',   
 `order\_date` = '**$order\_date**', `confirm\_date` = '**$confirm\_date**' WHERE `id` = '**$id**'"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"History record"**);  
 }  
}



Рис. 4.1.12. – Граф Маккейба для HistoryRecordDao::update

**HistoryRecord::getAll**

Исходный код программного компонента:

**public function** getAll()  
{  
 $items = **array**();  
 $table\_name = $this->getTableName();  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$table\_name**` WHERE `confirm\_date` IS NOT NULL"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
}

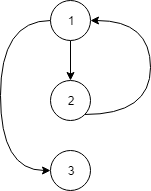


Рис. 4.1.13. – Граф Маккейба для HistoryRecordDao::getAll

**HistoryRecord::getBy**

Исходный код программного компонента:

**public function** getBy($column\_name, $value)  
{  
 $items = **array**();  
 $table\_name = $this->getTableName();  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$table\_name**` WHERE `**$column\_name**` = '**$value**'   
 AND `confirm\_date` IS NOT NULL"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
}

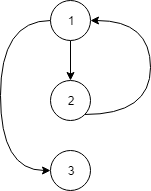


Рис. 4.1.14. – Граф Маккейба для HistoryRecordDao::getBy

**HistoryRecord::getColumnBy**

Исходный код программного компонента:

**public function** getColumnBy($column, $value, $get\_column)  
{  
 $items = **array**();  
 $table\_name = $this->getTableName();  
 $records = $this->getDb()->query(**"SELECT `**$get\_column**` FROM `**$table\_name**` WHERE `**$column**` = '**$value**'  
 AND `confirm\_date` IS NOT NULL"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $rec[$get\_column];  
 }  
 **return** $items;  
}

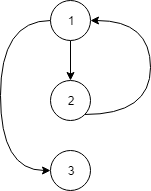


Рис. 4.1.15. – Граф Маккейба для HistoryRecordDao::getColumnBy

**OrderRecordDao::add**

Исходный код программного компонента:

**public function** add($record)  
{  
 **if** ($record **instanceof** OrderRecord)  
 {  
 $user\_id = ($record->getUser() == **null**) ? **null** : $record->getUser()->getId();  
 $product\_id = $record->getProduct()->getId();  
 $count = $record->getCount();  
 $address = $record->getAddress();  
 $phone = $record->getPhone();  
 $order\_date = $record->getOrderDate();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"INSERT INTO `**$table\_name**` (`user\_id`, `product\_id`, `count`, `address`,  
 `phone`, `order\_date`)   
 VALUES ('**$user\_id**', '**$product\_id**', '**$count**', '**$address**', '**$phone**', '**$order\_date**')"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"Order record"**);  
 }  
}



Рис. 4.1.16. – Граф Маккейба для OrderRecordDao::add

**OrderRecordDao::update**

Исходный код программного компонента:

**public function** update($record)  
{  
 **if** ($record **instanceof** OrderRecord)  
 {  
 $id = $record->getId();  
 $user\_id = ($record->getUser() == **null**) ? **null** : $record->getUser()->getId();  
 $product\_id = $record->getProduct()->getId();  
 $count = $record->getCount();  
 $address = $record->getAddress();  
 $phone = $record->getPhone();  
 $order\_date = $record->getOrderDate();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"UPDATE `**$table\_name**` SET `user\_id` = '**$user\_id**', `product\_id` = '**$product\_id**',  
 `count` = '**$count**', `address` = '**$address**', `phone` = '**$phone**',   
 `order\_date` = '**$order\_date**' WHERE `id` = '**$id**'"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"Order record"**);  
 }  
}



Рис. 4.1.17. – Граф Маккейба для OrderRecordDao::update

**OrderRecord::getAll**

Исходный код программного компонента:

**public function** getAll()  
{  
 $items = **array**();  
 $table\_name = $this->getTableName();  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$table\_name**` WHERE `confirm\_date` IS NULL"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
}

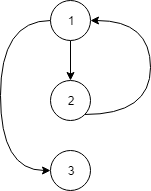


Рис. 4.1.18. – Граф Маккейба для OrderRecordDao::getAll

**OrderRecord::getBy**

Исходный код программного компонента:

**public function** getBy($column\_name, $value)  
{  
 $items = **array**();  
 $table\_name = $this->getTableName();  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$table\_name**` WHERE `**$column\_name**` = '**$value**'   
 AND `confirm\_date` IS NULL"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
}

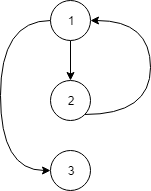


Рис. 4.1.19. – Граф Маккейба для OrderRecordDao::getBy

**OrderRecord::getColumnBy**

Исходный код программного компонента:

**public function** getColumnBy($column, $value, $get\_column)  
{  
 $items = **array**();  
 $table\_name = $this->getTableName();  
 $records = $this->getDb()->query(**"SELECT `**$get\_column**` FROM `**$table\_name**` WHERE `**$column**` = '**$value**'  
 AND `confirm\_date` IS NULL"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $rec[$get\_column];  
 }  
 **return** $items;  
}

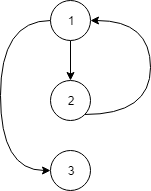


Рис. 4.1.20. – Граф Маккейба для OrderRecordDao::getColumnBy

**ProductDao::add**

Исходный код программного компонента:

**public function** add($record)  
{  
 **if** ($record **instanceof** Product)  
 {  
 $name = $record->getName();  
 $price = $record->getPrice();  
 $category\_id = $record->getCategory()->getId();  
 $description = $record->getDescription();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"INSERT INTO `**$table\_name**` (`name`, `price`, `category\_id`, `description`)   
 VALUES ('**$name**', '**$price**', '**$category\_id**', '**$description**')"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"product"**);  
 }  
}



Рис. 4.1.21. – Граф Маккейба для ProductDao::add

**ProductDao::update**

Исходный код программного компонента:

**public function** update($record)  
{  
 **if** ($record **instanceof** Product)  
 {  
 $id = $record->getId();  
 $name = $record->getName();  
 $price = $record->getPrice();  
 $category\_id = $record->getCategory()->getId();  
 $description = $record->getDescription();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"UPDATE `**$table\_name**` SET `name` = '**$name**', `price` = '**$price**',  
 `category\_id` = '**$category\_id**', `description` = '**$description**' WHERE `id` = '**$id**'"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"product"**);  
 }  
}



Рис. 4.1.22. – Граф Маккейба для ProductDao::update

**UserDao::add**

Исходный код программного компонента:

**public function** add($record)  
{  
 **if** ($record **instanceof** User)  
 {  
 $login = $record->getLogin();  
 $email = $record->getEmail();  
 $password = $record->getPassword();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"INSERT INTO `**$table\_name**` (`login`, `email`, `password`)   
 VALUES ('**$login**', '**$email**', '**$password**')"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"user"**);  
 }  
}



Рис. 4.1.23. – Граф Маккейба для UserDao::add

**UserDao::update**

Исходный код программного компонента:

**public function** update($record)  
{  
 **if** ($record **instanceof** User)  
 {  
 $id = $record->getId();  
 $login = $record->getLogin();  
 $email = $record->getEmail();  
 $password = $record->getPassword();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"UPDATE `**$table\_name**` SET `login` = '**$login**', `email` = '**$email**',  
 `password` = '**$password**' WHERE `id` = '**$id**'"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"user"**);  
 }  
}



Рис. 4.1.24. – Граф Маккейба для UserDao::update

**AuthenticationVerifyer::authenticationErrors**

Исходный код программного компонента:

**public function** authenticationErrors($login, $password, $mysqli) {  
 **if** (!$this->isCorrectLogin($login)) {  
 **return** $this->**error\_msgs**[**'incorrect\_login'**];  
 } **else if** (!$this->isCorrectPassword($password)) {  
 **return** $this->**error\_msgs**[**'incorrect\_password'**];  
 } **else if** (!$this->isTrueUser($login, $password, $mysqli)) {  
 **return** $this->**error\_msgs**[**'not\_user'**];  
 } **else** {  
 **return false**;  
 }  
}

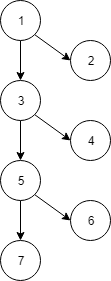


Рис. 4.1.25. – Граф Маккейба для AuthenticationVerifyer::authenticationErrors

**AuthenticationVerifyer::isTrueUser**

Исходный код программного компонента:

**public function** isTrueUser($login, $password, $mysqli) {  
 **if** ($this->isExistUser($login, $mysqli)) {  
 $dao = **new** UserDao($mysqli);  
 $user = $dao->getBy(**'login'**, $login)[0];  
 $hash = $user->getPassword();  
 **return** *password\_verify*($password, $hash);  
 }  
 **return false**;  
}

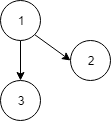


Рис. 4.1.26. – Граф Маккейба для AuthenticationVerifyer::isTrueUser

**Authenticator::authenticate**

Исходный код программного компонента:

**public function** authenticate($login, $error)  
{  
 **if** (!$error) {  
 $\_SESSION[**"logged\_user"**] = $login;  
 }  
}

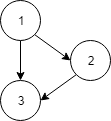


Рис. 4.1.27. – Граф Маккейба для Authenticator::authenticate

**Cashier::getTotalPrice**

Исходный код программного компонента:

**public function** getTotalPrice($cart)  
{  
 $totalPrice = 0;  
 **if** (!*is\_array*($cart))  
 {  
 **throw new** InvalidArgumentException(**"not array"**);  
 }  
 **foreach** ($cart **as** $record)  
 {  
 **if** ($record **instanceof** CartRecord)  
 {  
 $totalPrice += $record->getProduct()->getPrice()  
 \* $record->getCount();  
 }  
 }  
 **return** $totalPrice;  
}

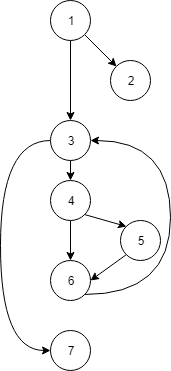


Рис. 4.1.28. – Граф Маккейба для Cashier::getTotalPrice

**CategoryVerifyer::categoryErrors**

Исходный код программного компонента:

**public function** categoryErrors($name, $mysqli) {  
 **if** (!$this->isCorrectName($name)) {  
 **return** $this->**error\_msgs**[**"incorrect\_name"**];  
 } **else if** ($this->isExistCategoryName($name, $mysqli)) {  
 **return** $this->**error\_msgs**[**"double\_name"**];  
 } **else** {  
 **return false**;  
 }  
}

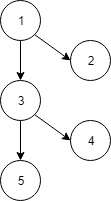


Рис. 4.1.29. – Граф Маккейба для CategoryVerifyer::categoryErrors

**FaceControl::getOneOf**

Исходный код программного компонента:

**public function** getOneOf($if\_guest, $if\_user, $if\_admin=**null**)  
{  
 **if** (!**isset**($\_SESSION[**'logged\_user'**]))  
 {  
 **return** $if\_guest;  
 }  
 **else if** ($\_SESSION[**'logged\_user'**] == **'admin'**)  
 {  
 **return** $if\_admin;  
 }  
 **else** {  
 **return** $if\_user;  
 }  
}

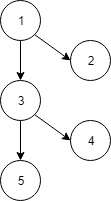


Рис. 4.1.30. – Граф Маккейба для FaceControl::getOneOf

**OrderVerifyer::orderErrors**

Исходный код программного компонента:

**public function** orderErrors($address, $phone) {  
 **if** (!$this->isCorrectAddress($address)) {  
 **return** $this->**error\_msgs**[**"incorrect\_address"**];  
 } **else if** (!$this->isCorrectPhone($phone)) {  
 **return** $this->**error\_msgs**[**"incorrect\_phone"**];  
 } **else** {  
 **return false**;  
 }  
}

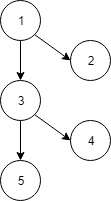


Рис. 4.1.31. – Граф Маккейба для OrderVerifyer::orderErrors

**ProductVerifyer::productErrors**

Исходный код программного компонента:

**public function** productErrors($name, $price, $description, $mysqli) {  
 **if** (!$this->isCorrectName($name)) {  
 **return** $this->**error\_msgs**[**"incorrect\_name"**];  
 } **else if** (!$this->isCorrectPrice($price)) {  
 **return** $this->**error\_msgs**[**"incorrect\_price"**];  
 } **else if** (!$this->isCorrectDescription($description)) {  
 **return** $this->**error\_msgs**[**"big\_description"**];  
 } **else if** ($this->isExistName($name, $mysqli)) {  
 **return** $this->**error\_msgs**[**"double\_name"**];  
 } **else** {  
 **return false**;  
 }  
}

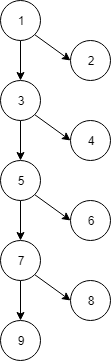


Рис. 4.1.32. – Граф Маккейба для ProductVerifyer::productErrors

**RegistrationVerifyer::registrationErrors**

Исходный код программного компонента:

**public function** registrationErrors($login, $email, $password, $password2, $mysqli) {  
 **if** (!**self**::*isCorrectLogin*($login)) {  
 **return** $this->**error\_msgs**[**"incorrect\_login"**];  
 } **else if** (!$this->isCorrectPassword($password)) {  
 **return** $this->**error\_msgs**[**"incorrect\_password"**];  
 } **else if** (!$this->isCorrectEmail($email)) {  
 **return** $this->**error\_msgs**[**"incorrect\_email"**];  
 } **else if** ($this->isExistUser($login, $mysqli)) {  
 **return** $this->**error\_msgs**[**"double\_login"**];  
 } **else if** ($this->isExistEmail($email, $mysqli)) {  
 **return** $this->**error\_msgs**[**"double\_email"**];  
 } **else if** ($password != $password2) {  
 **return** $this->**error\_msgs**[**"incorrect\_repeat"**];  
 } **else** {  
 **return false**;  
 }  
}

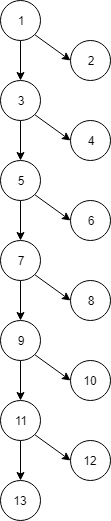


Рис. 4.1.33. – Граф Маккейба для RegistrationVerifyer::registrationErrors

**Registrator::registrateUser**

Исходный код программного компонента:

**public function** registrateUser($login, $email, $password, $errors, $db)  
{  
 **if** (!$errors) {  
 $hash = *password\_hash*($password, ***PASSWORD\_DEFAULT***);  
 $dao = **new** UserDao($db);  
 $dao->add(**new** User(0, $login, $email, $hash));  
 }  
}

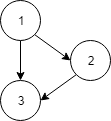


Рис. 4.1.34. – Граф Маккейба для Registrator::registrateUser

**URIResolver::clearURI**

Исходный код программного компонента:

**public function** clearURI($uri)  
{  
 **if** (*strripos*($uri, **'?'**) !== **false**) {  
 **return** *substr*($uri, 0, *strripos*($uri, **'?'**));  
 }  
 **return** $uri;  
}

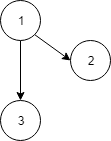


Рис. 4.1.35. – Граф Маккейба для URIResolver::clearURI

**URIResolver::getValue**

Исходный код программного компонента:

**public function** getValue($name)  
{  
 **if** ($this->hasGET($name)) {  
 **return** $\_GET[$name];  
 }  
 **return false**;  
}

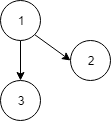


Рис. 4.1.36. – Граф Маккейба для URIResolver::getValue

**URIResolver::getPOSTValue**

Исходный код программного компонента:

**public function** getPOSTValue($name)  
{  
 **if** ($this->hasPOST($name)) {  
 **return** $\_POST[$name];  
 }  
 **return false**;  
}

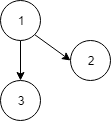


Рис. 4.1.37. – Граф Маккейба для URIResolver::getPOSTValue

**URIResolver::setToURI**

Исходный код программного компонента:

**public function** setToURI($uri, $name, $value)  
{  
 **if** (*strripos*($uri, **'?'**) !== **false**)  
 {  
 $valuess = *substr*($uri, *strripos*($uri, **'?'**) + 1);  
 $values = *explode*(**'&'**, $valuess);  
 **foreach** ($values **as** $val) {  
 **if** (*strripos*($val, $name . **'='**) === 0) {  
 $replace = $name . **'='** . $value;  
 **return** *str\_replace*($val, $replace, $uri);  
 }  
 }  
 **return** $uri . **'&'** . $name . **'='** . $value;  
 }  
 **else** {  
 **return** $uri . **'?'** . $name . **'='** . $value;  
 }  
}

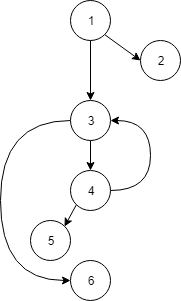


Рис. 4.1.38. – Граф Маккейба для URIResolver::setToURI

## Формирование проходов

**CartDao::add**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | $record не является массивом |
| 1-3-7 | $record является пустым массивом |
| 1-3-4-6-7 | $record – массив, который не содержит в себе объектов типа CartRecord |
| 1-3-4-5-6-7 | $record – массив, который содержит в себе объекты типа CartRecord |

**CartDao::delete**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 2 | 1-3 | $record не является объектом типа CartRecord |
| 1-2-3 | $record является объектом типа CartRecord |

**CartDao::deleteBy**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 2 | 1-3 | Записей с данным $id не существует |
| 1-2-3 | Запись с данным $id есть в массиве SESSION |

**CategoryDao::add**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | $record не является объектом типа Category |
| 1-3 | $record является объектом типа Category |

**CategoryDao::update**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | $record не является объектом типа Category |
| 1-3 | $record является объектом типа Category |

**DBDao::getAll**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 2 | 1-3 | Запрос к БД дал пустой результат |
| 1-2-1-3 | Запрос к БД вернул одну или несколько записей |

**DBDao::getAllSearched**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 6 | 1-2-5-11 | $sort\_column не указан, запрос к БД дал пустой результат |
| 1-3-7-11 | $sort\_column указан, $desc имеет значение true, запрос к БД дал пустой результат |
| 1-4-9-11 | $sort\_column указан, $desc имеет значение false, запрос к БД дал пустой результат |
| 1-2-5-6-5-11 | $sort\_column не указан, запрос к БД вернул одну или несколько записей |
| 1-3-7-8-7-11 | $sort\_column указан, $desc имеет значение true, запрос к БД вернул одну или несколько записей |
| 1-4-9-10-9-11 | $sort\_column указан, $desc имеет значение false, запрос к БД вернул одну или несколько записей |

**DBDao::getBy**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 2 | 1-3 | Запрос к БД дал пустой результат |
| 1-2-1-3 | Запрос к БД вернул одну или несколько записей |

**DBDao::getColumnBy**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 2 | 1-3 | Запрос к БД дал пустой результат |
| 1-2-1-3 | Запрос к БД вернул одну или несколько записей |

**IdentificationalDao::delete**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | $record не является объектом типа Identificational |
| 1-3 | $record является объектом типа Identificational |

**HistoryRecordDao::add**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | $record не является объектом типа HistoryRecord |
| 1-3 | $record является объектом типа HistoryRecord |

**HistoryRecordDao::update**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | $record не является объектом типа HistoryRecord |
| 1-3 | $record является объектом типа HistoryRecord |

**HistoryRecord::getAll**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 2 | 1-3 | Запрос к БД дал пустой результат |
| 1-2-1-3 | Запрос к БД вернул одну или несколько записей |

**HistoryRecord::getBy**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 2 | 1-3 | Запрос к БД дал пустой результат |
| 1-2-1-3 | Запрос к БД вернул одну или несколько записей |

**HistoryRecord::getColumnBy**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 2 | 1-3 | Запрос к БД дал пустой результат |
| 1-2-1-3 | Запрос к БД вернул одну или несколько записей |

**OrderRecordDao::add**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | $record не является объектом типа OrderRecord |
| 1-3 | $record является объектом типа OrderRecord |

**OrderRecordDao::update**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | $record не является объектом типа OrderRecord |
| 1-3 | $record является объектом типа OrderRecord |

**OrderRecord::getAll**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 2 | 1-3 | Запрос к БД дал пустой результат |
| 1-2-1-3 | Запрос к БД вернул одну или несколько записей |

**OrderRecord::getBy**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 2 | 1-3 | Запрос к БД дал пустой результат |
| 1-2-1-3 | Запрос к БД вернул одну или несколько записей |

**OrderRecord::getColumnBy**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 2 | 1-3 | Запрос к БД дал пустой результат |
| 1-2-1-3 | Запрос к БД вернул одну или несколько записей |

**ProductDao::add**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | $record не является объектом типа Product |
| 1-3 | $record является объектом типа Product |

**ProductDao::update**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | $record не является объектом типа Product |
| 1-3 | $record является объектом типа Product |

**UserDao::add**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | $record не является объектом типа User |
| 1-3 | $record является объектом типа User |

**UserDao::update**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | $record не является объектом типа User |
| 1-3 | $record является объектом типа User |

**AuthenticationVerifyer::authenticationErrors**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | Логин не является корректным |
| 1-3-4 | Пароль не является корректным |
| 1-3-5-6 | Пользователя с таким логином не существует либо введен неверный пароль |
| 1-3-5-7 | Пользователь с таким логином и паролем есть в БД |

**AuthenticationVerifyer::isTrueUser**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | Пользователя с логином $login не существует |
| 1-3 | Пользователь с логином $login есть в БД |

**Authenticator::authenticate**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 2 | 1-3 | $error имеет значение true |
| 1-2-3 | $error имеет значение false |

**Cashier::getTotalPrice**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | $cart не является массивом |
| 1-3-7 | $cart – пустой массив |
| 1-3-4-6-7 | $cart – массив, который не содержит в себе объектов типа CartRecord |
| 1-3-4-5-6-7 | $cart – массив, который содержит в себе объекты типа CartRecord |

**CategoryVerifyer::categoryErrors**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | Название категории не является корректным |
| 1-3-4 | Категория с таким именем уже существует |
| 1-3-5 | Категория имеет корректное уникальное имя |

**FaceControl::getOneOf**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | Текущий пользователь не авторизован |
| 1-3-4 | Текущий пользователь является администратором |
| 1-3-5 | Текущий пользователь авторизован (не является администратором) |

**OrderVerifyer::orderErrors**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | Адрес не является корректным |
| 1-3-4 | Телефон не является корректным |
| 1-3-5 | Адрес и телефон являются корректными |

**ProductVerifyer::productErrors**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | Название товара не является корректным |
| 1-3-4 | Цена товара является корректной |
| 1-3-5-6 | Описание товара не является корректным |
| 1-3-5-7-8 | Название товара уже существует |
| 1-3-5-7-9 | Товар имеет уникальное и корректное название и корректные цену и описание |

**RegistrationVerifyer::registrationErrors**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | Логин не является корректным |
| 1-3-4 | Пароль не является корректным |
| 1-3-5-6 | Адрес электронной почты не является корректным |
| 1-3-5-7-8 | Пользователь с таким логином уже существует |
| 1-3-5-7-9-10 | Пользователь с таким адресом электронной почты уже существует |
| 1-3-5-7-9-11-12 | Пользователь неправильно повторил пароль |
| 1-3-5-7-9-11-13 | Пользователь указал корректно все данные. Логин и адрес электронной почты уникальны |

**Registrator::registrateUser**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| 2 | 1-3 | $errors имеет значение true |
| 1-2-3 | $errors имеет значение false |

**URIResolver::clearURI**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | В строке $uri нет символа «?» |
| 1-3 | В строке $uri имеется символ «?» |

**URIResolver::getValue**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | Имеется значение $name в массиве $\_GET |
| 1-3 | Значения $name в массиве $\_GET нет |

**URIResolver::getPOSTValue**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | Имеется значение $name в массиве $\_POST |
| 1-3 | Значения $name в массиве $\_POST нет |

**URIResolver::setToURI**

|  |  |  |
| --- | --- | --- |
| Цикломатическое число | Описание прохода | Контрольные примеры, позволяющие реализовать данную ситуацию |
| Данный граф не является замкнутым | 1-2 | В строке $uri нет символа «?» |
| 1-3-6 | В строке $uri только одно значение |
| 1-3-4-5 | В строке $uri найдено такое же значение $name |
| 1-3-4-6 | В строке $uri не найдено такого же значения $name |

# Реализация модульного тестирования

## Реализация автоматизированных скриптов

Модульное тестирование разработанных компонентов было реализовано на языке PHP с использованием тестового фреймворка Codeception. Данный инструмент позволяет автоматизировать процесс модульного тестирования.

В некоторых случаях необходимо было использовать разнообразные заглушки. В частности, для тестирования взаимодействия с базой данных были использованы dummy-объекты, stub-объекты, моки и шпионы. В тестовом фреймворке Codeception для создания заглушки используются два метода класса \Codeception\Test\Unit: make и construct. Метод make используется для создания заглушки без вызова конструктора, а construct – для создания с вызовом конструктора (параметры могут передаваться через массив).

В данном курсовом проекте было реализовано девяносто девять тестовых методов для тридцати восьми программных компонентов.

Листинг автоматизированных скриптов представлен в Приложении Б.

## Анализ результатов тестов

Результаты выполнения тестов приведены ниже:

AuthenticationVerifyerTest

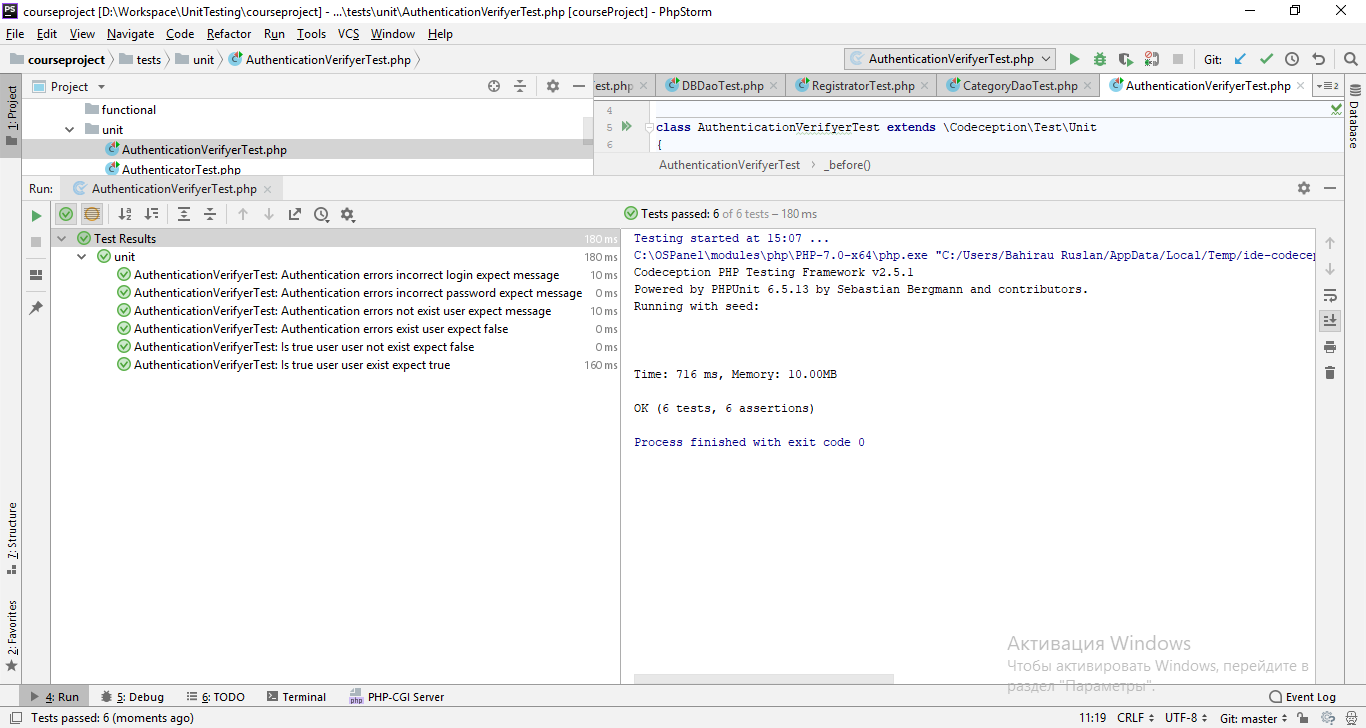


Рис. 5.2.1. – Результат выполнения тестов AuthenticationVerifyerTest

AuthenticatorTest

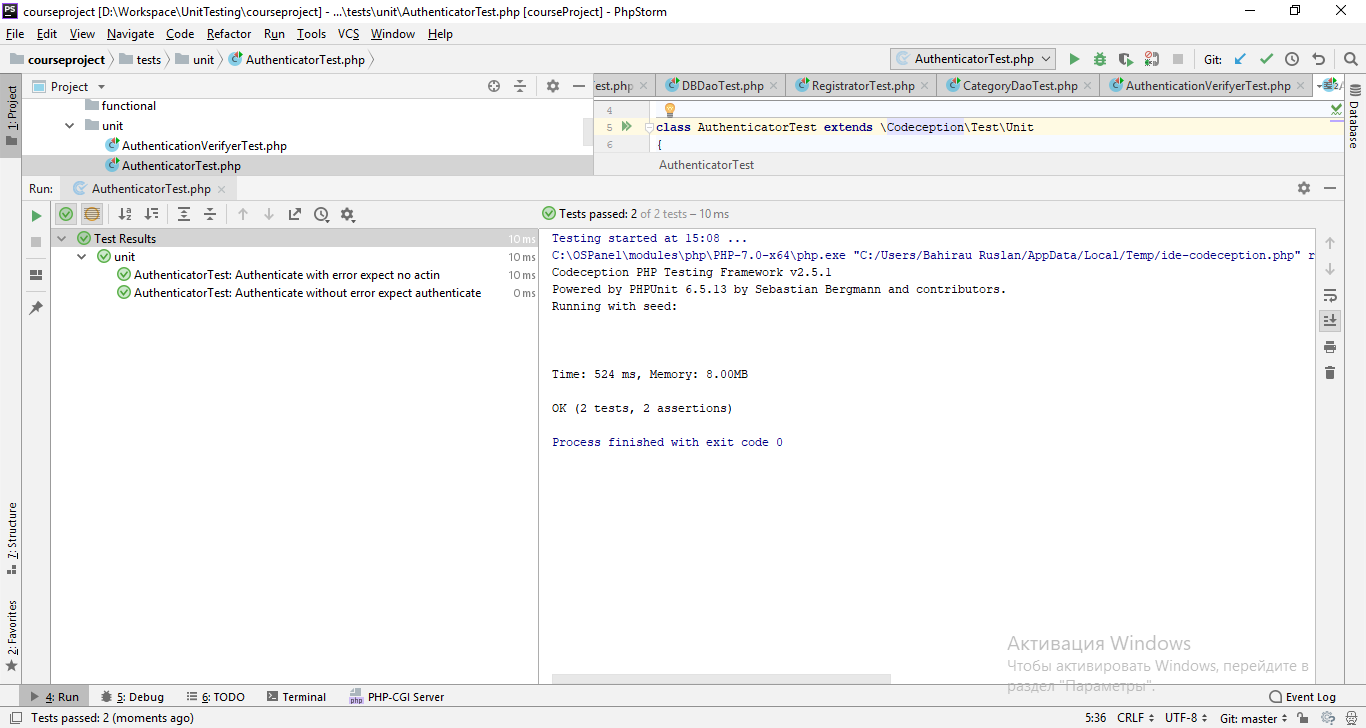


Рис. 5.2.2. – Результат выполнения тестов AuthenticatorTest

CartDaoTest

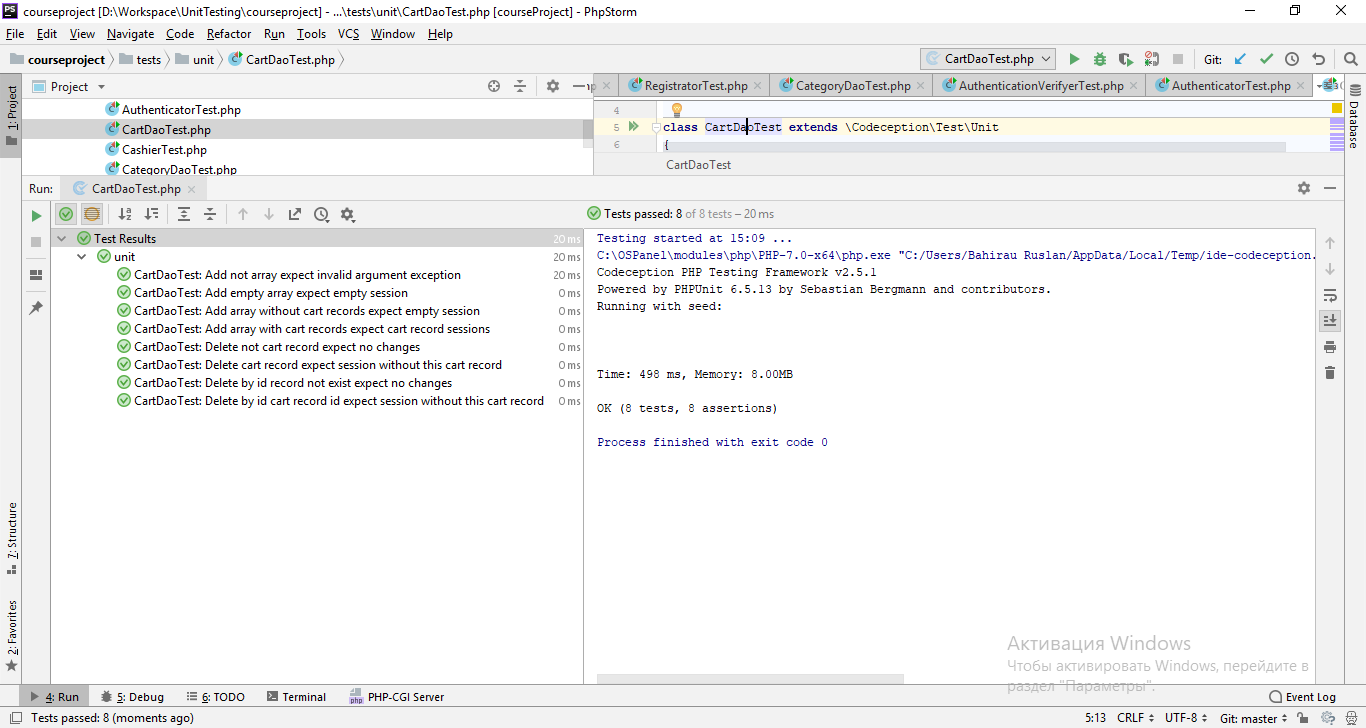


Рис. 5.2.3. – Результат выполнения тестов CartDaoTest

CashierTest

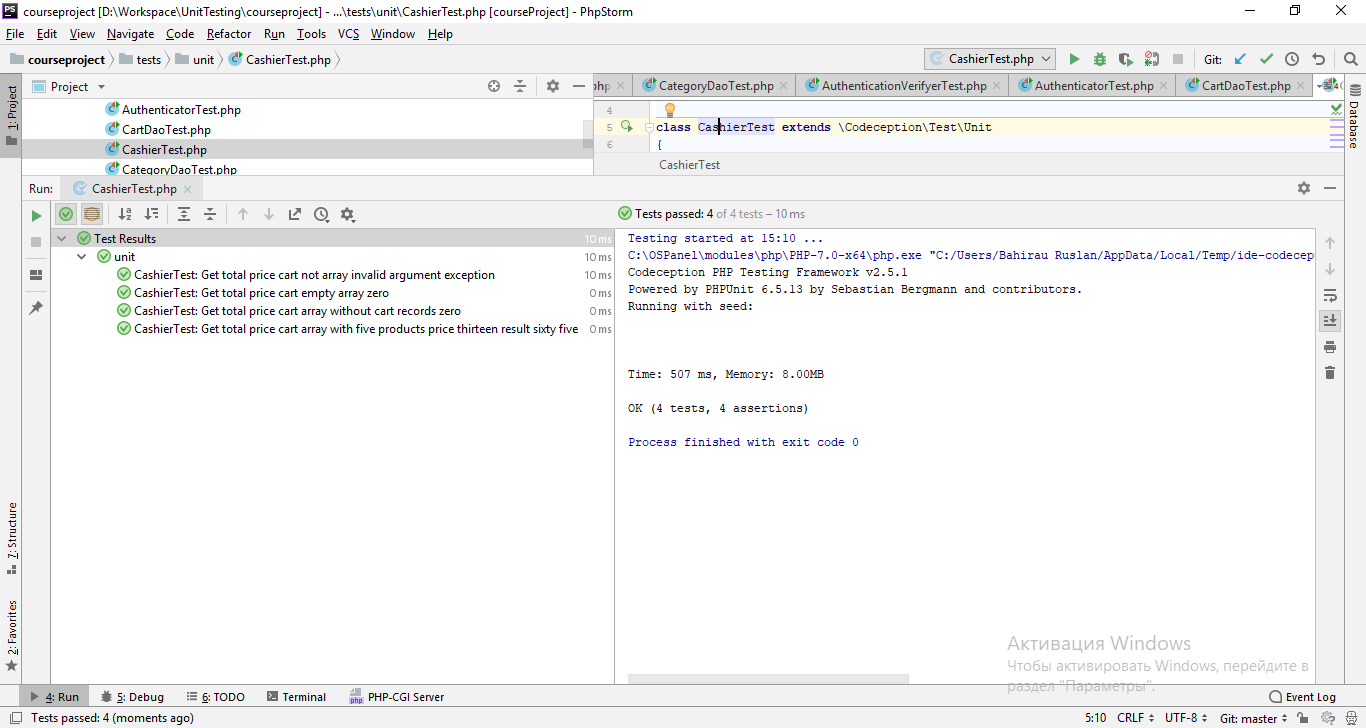


Рис. 5.2.4. – Результат выполнения тестов CashierTest

CategoryDaoTest

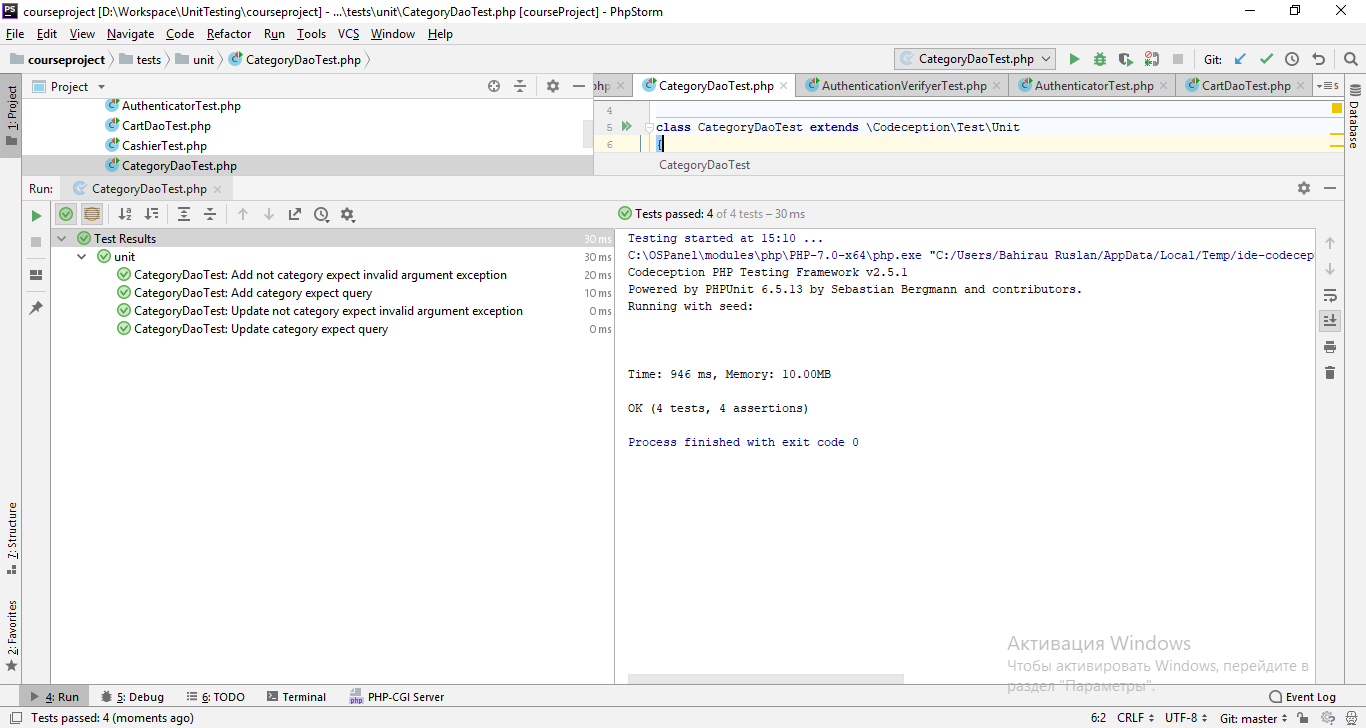


Рис. 5.2.5. – Результат выполнения тестов CategoryDaoTest

CategoryVerifyerTest

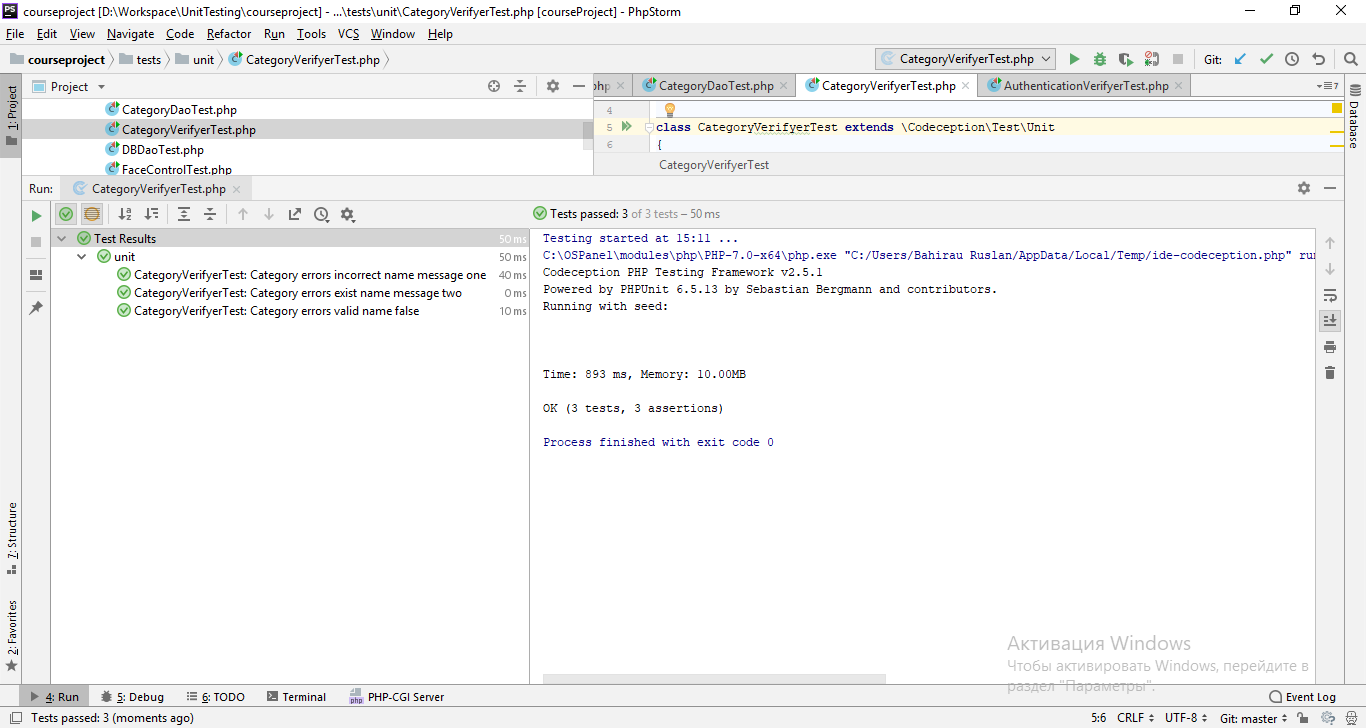


Рис. 5.2.6. – Результат выполнения тестов CategoryVerifyerTest

DBDaoTest

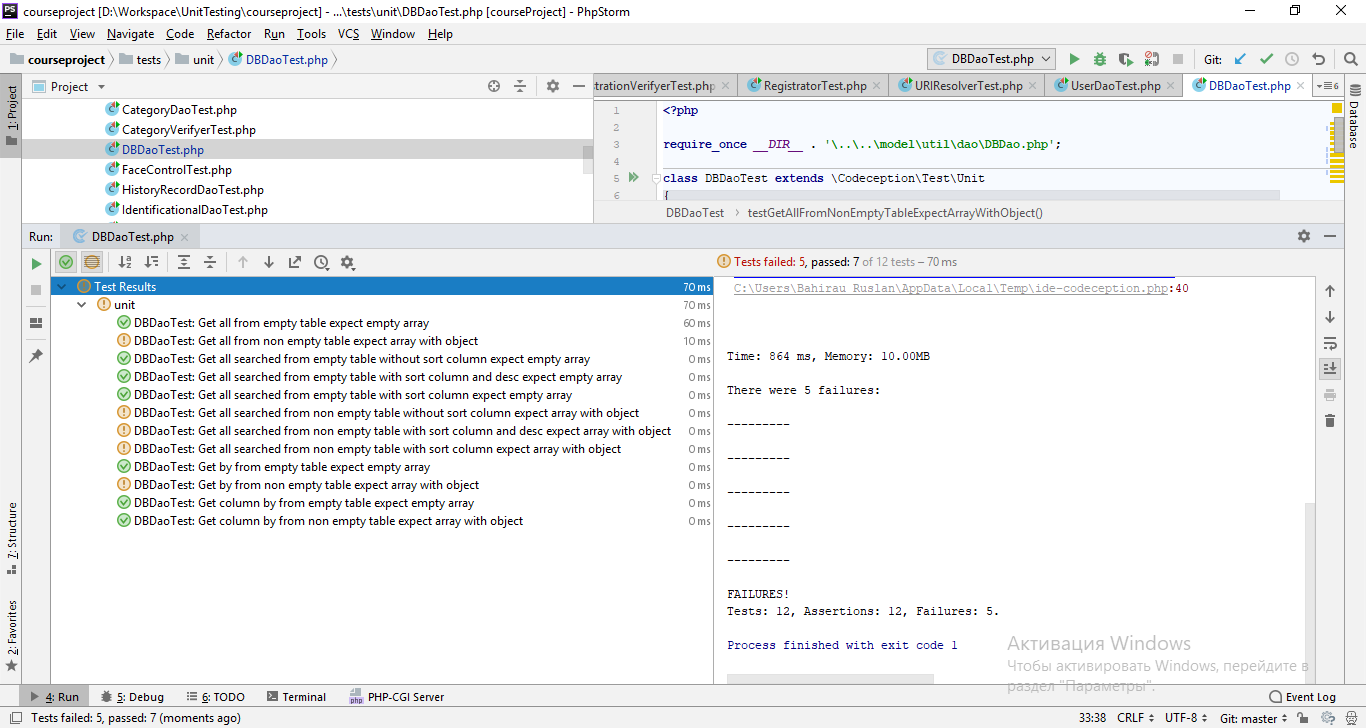


Рис. 5.2.7. – Результат выполнения тестов DBDaoTest

FaceControlTest

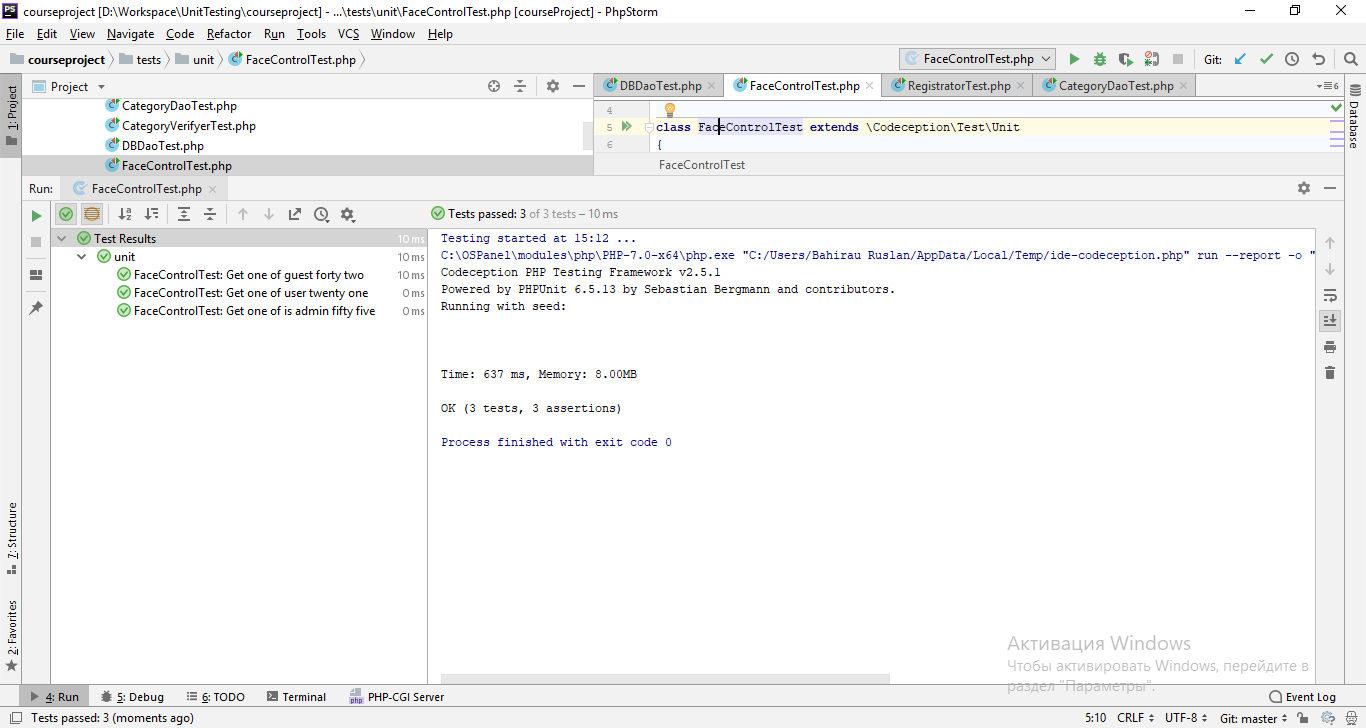


Рис. 5.2.8. – Результат выполнения тестов FaceControlTest

HistoryRecordDaoTest

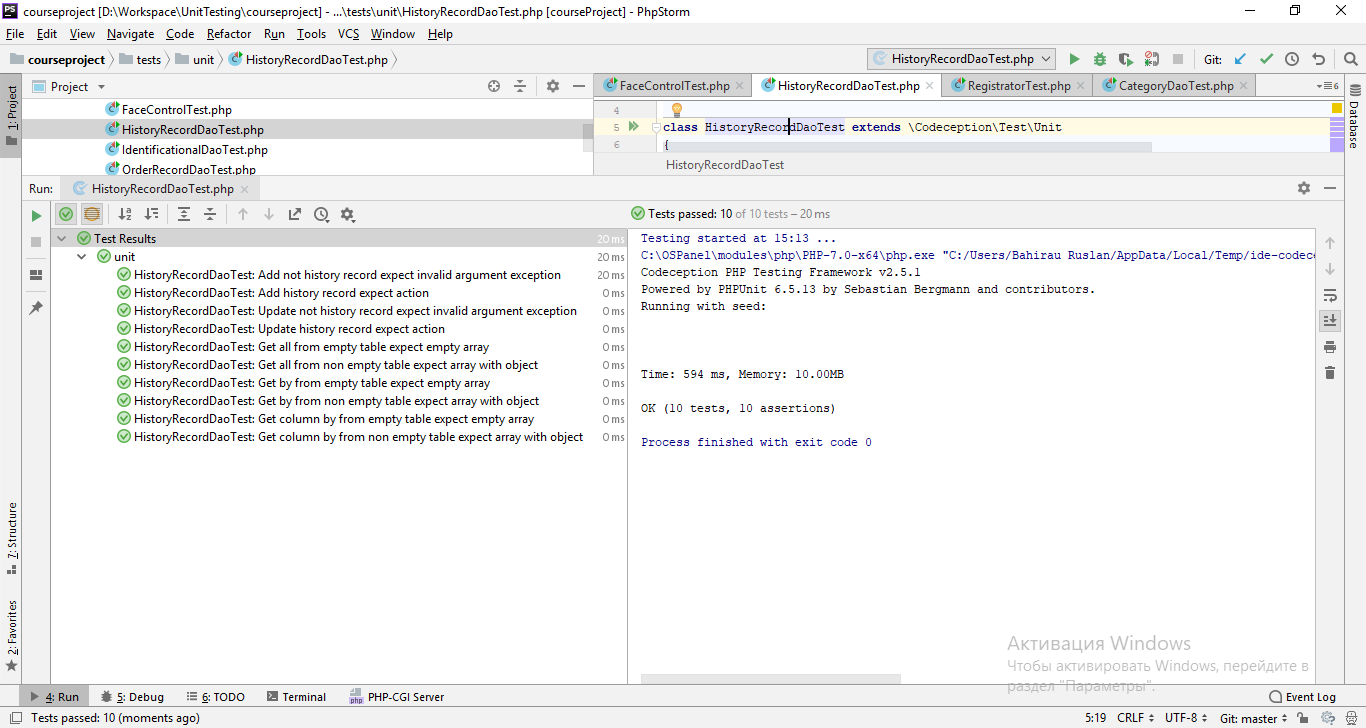


Рис. 5.2.9. – Результат выполнения тестов HistoryRecordDaoTest

IdentificationalDaoTest

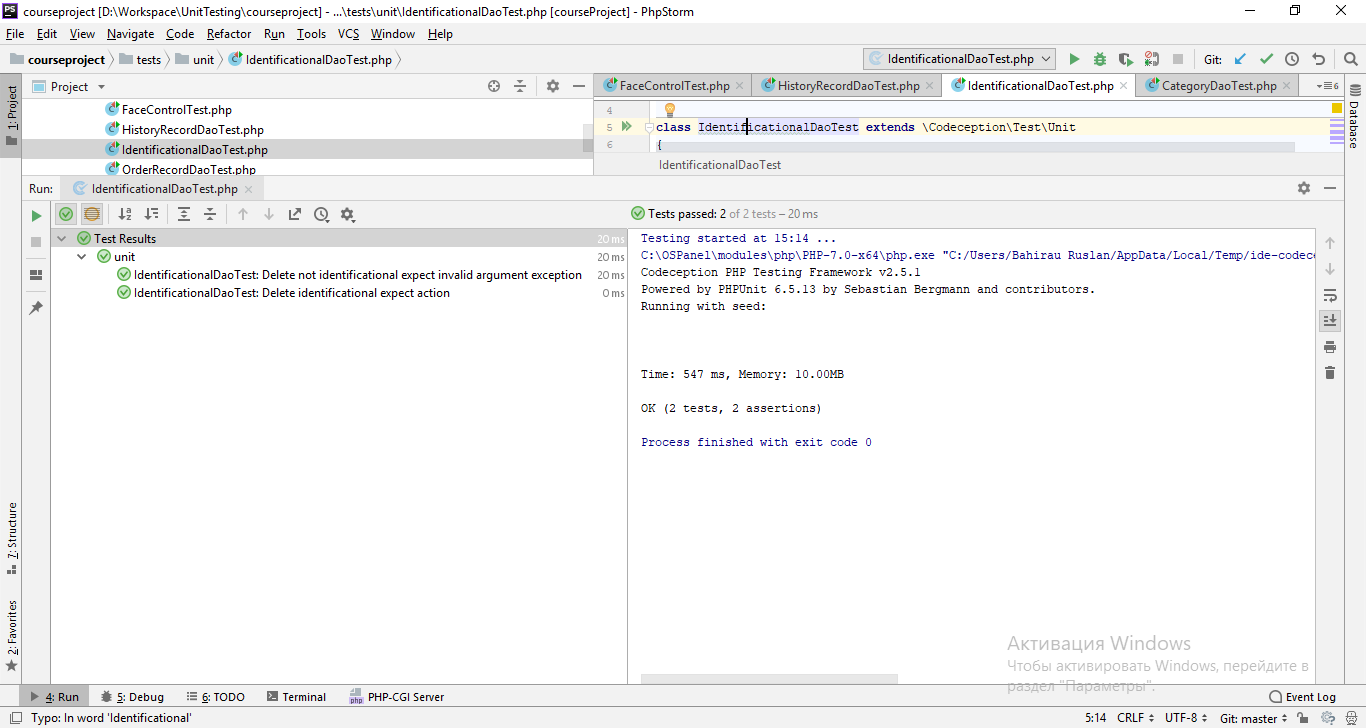


Рис. 5.2.10. – Результат выполнения тестов IdentificationalDaoTest

OrderRecordDaoTest

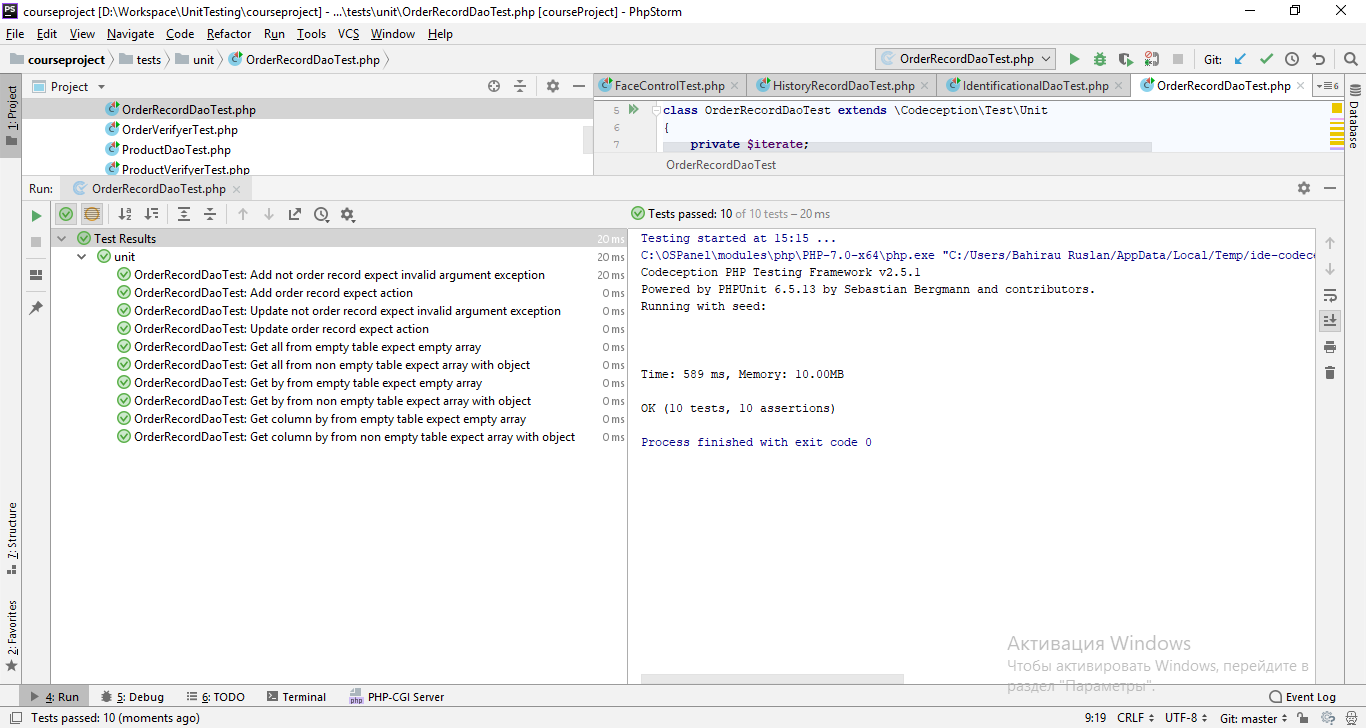


Рис. 5.2.11. – Результат выполнения тестов OrderRecordDaoTest

OrderVerifyerTest

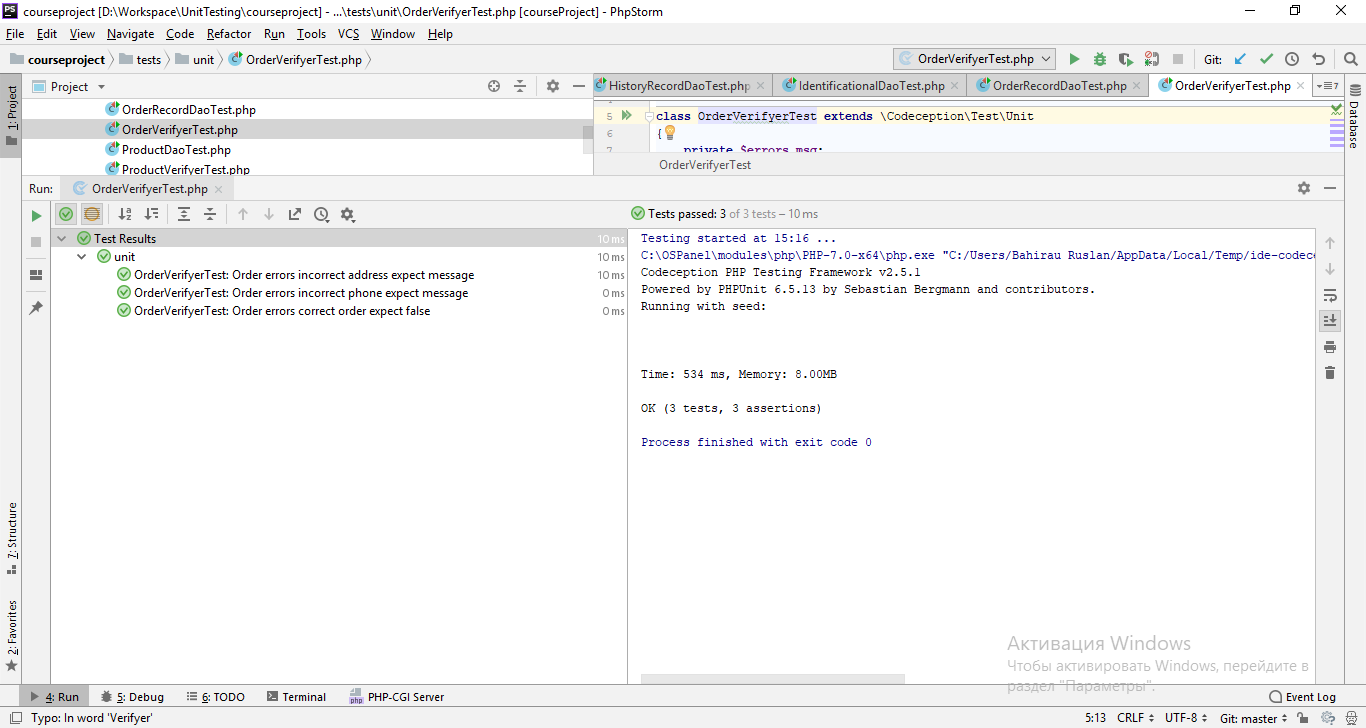


Рис. 5.2.12. – Результат выполнения тестов OrderVerifyerTest

ProductDaoTest

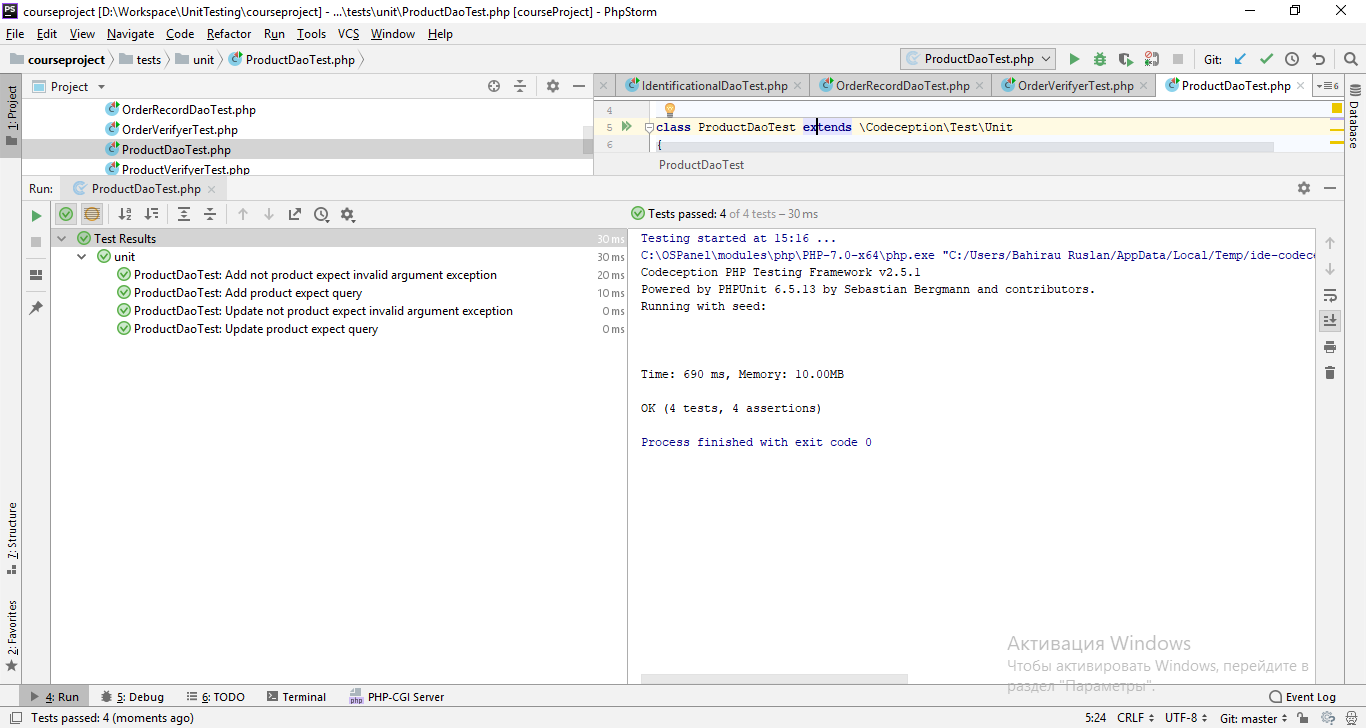


Рис. 5.2.13. – Результат выполнения тестов ProductDaoTest

ProductVerifyerTest

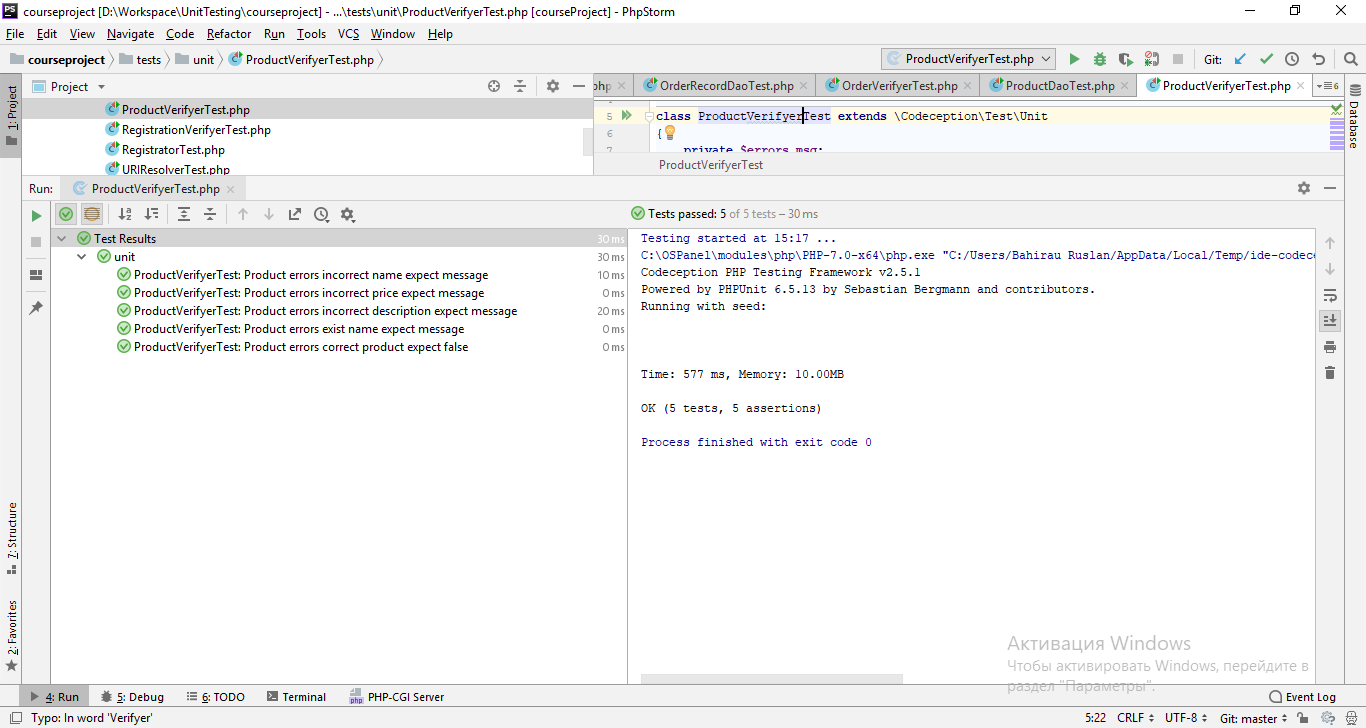


Рис. 5.2.14. – Результат выполнения тестов ProductVerifyerTest

RegistrationVerifyerTest

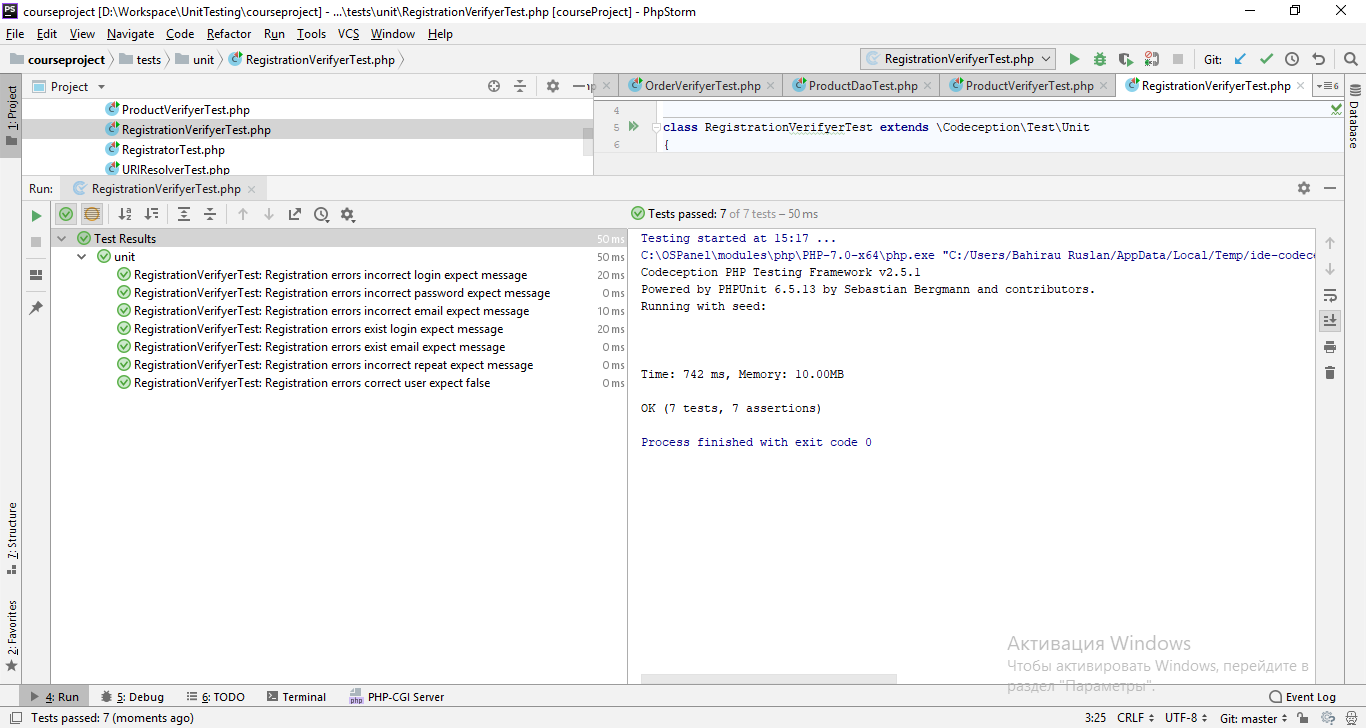


Рис. 5.2.15. – Результат выполнения тестов RegistrationVerifyerTest

RegistratorTest

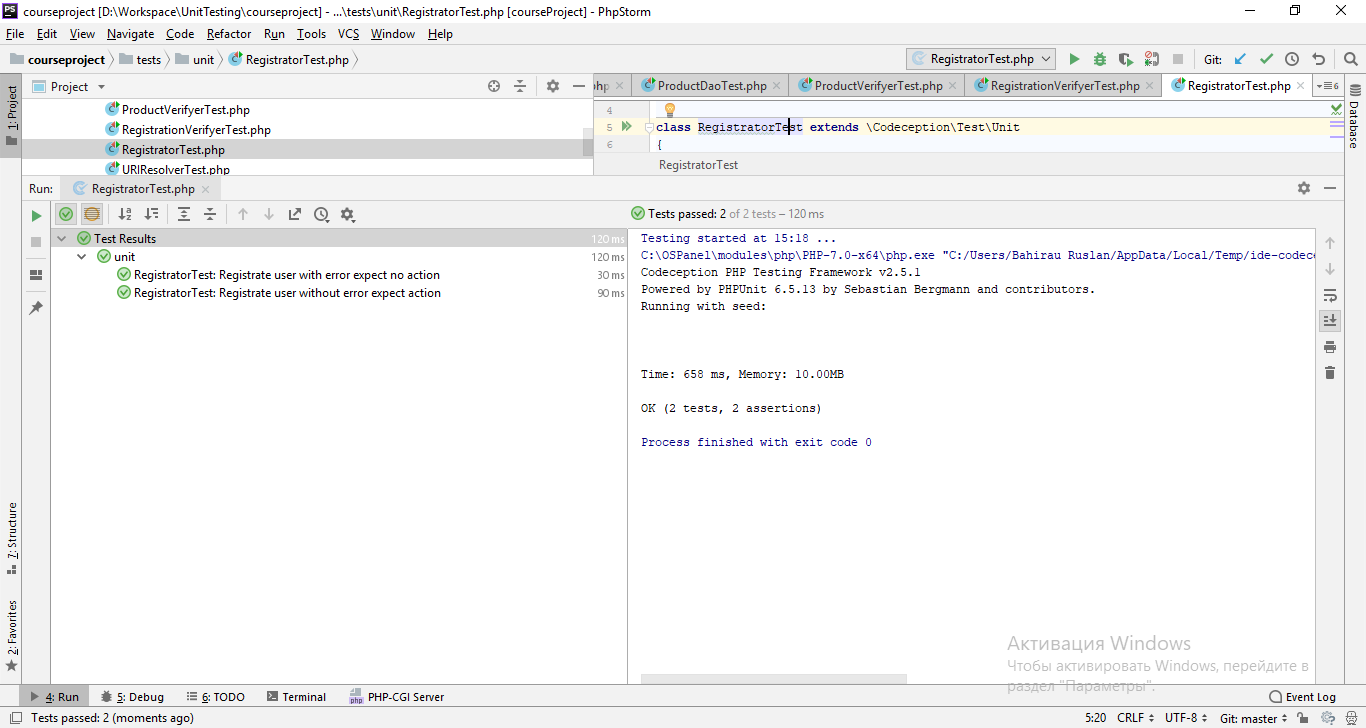


Рис. 5.2.16. – Результат выполнения тестов RegistratorTest

URIResolverTest

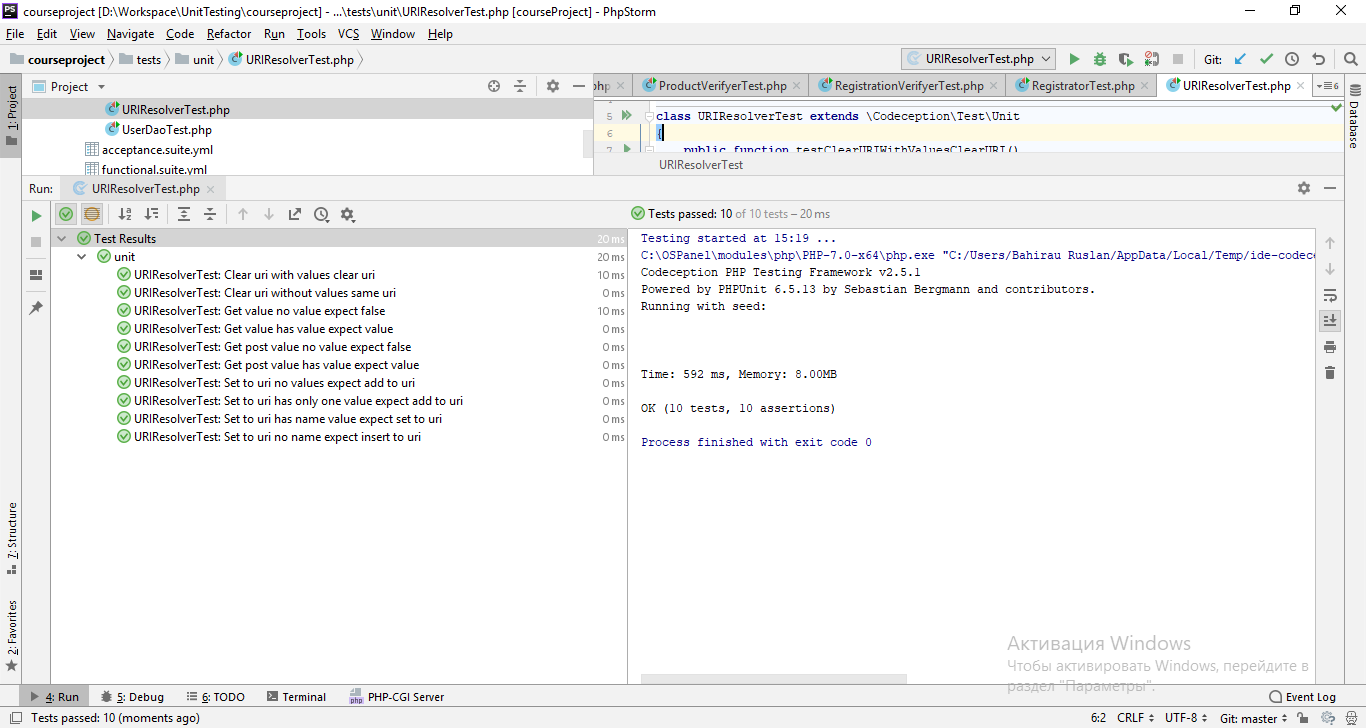


Рис. 5.2.17. – Результат выполнения тестов URIResolverTest

UserDaoTest

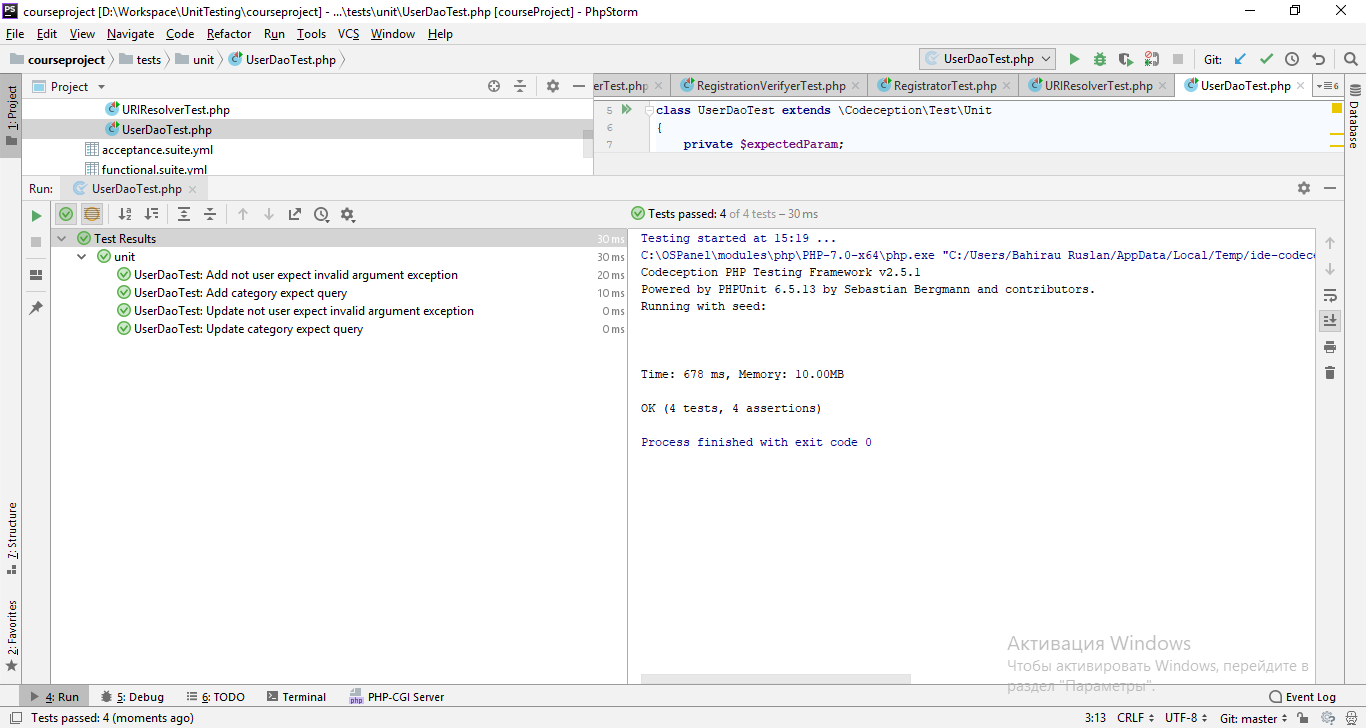


Рис. 5.2.18. – Результат выполнения тестов UserDaoTest

Как видно из рисунков выше, почти все тесты прошли успешно, что означает, что программные модули, для которых все тесты выполнены успешно, обладают именно таким поведением, которое мы от них ожидаем.

Однако в тестовом классе DBDaoTest не все тесты прошли успешно.

## Отладка тестов

При тщательном анализе скрипта тестового класса DBDaoTest была обнаружена ошибка. При инициализации переменной $expected ей присваивалось значение $object, а методы с названием get возвращают массивы с результатами, поэтому инициальзировать $expected надо не значением $object, а значением array($object), так как мы ожидаем массив значений. Исправим скрипт данного тестового класса и запустим его заново. Результат выполнения тестового скрипта приведен на рис. 5.3.1.

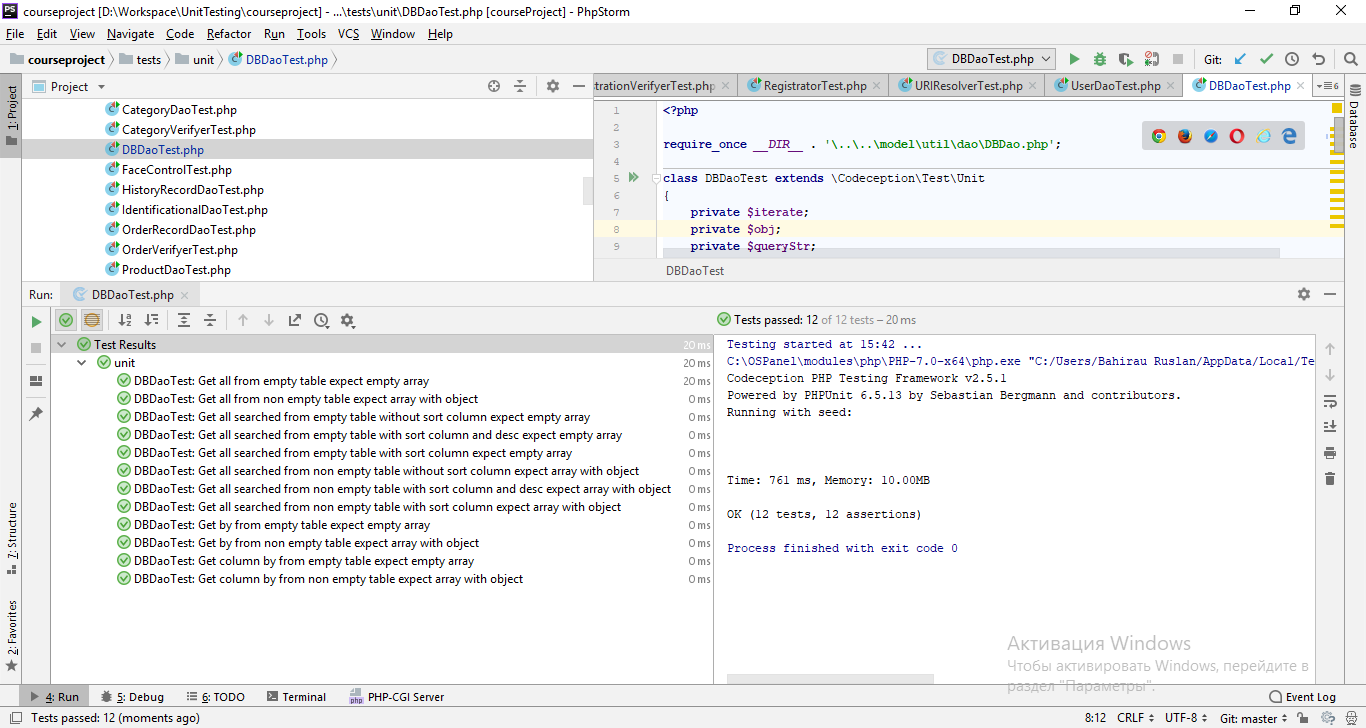


Рис. 5.3.1. – Результат выполнения исправленного теста

Как видно, тест теперь проходит, что означает, что все тесты в проекте дают положительный результат.

# Заключение

При выполнении данного курсового проекта мною были закреплены на практике умения проектировать и разрабатывать приложения. Были отточены навыки по проектированию тестов и автоматизации модульного тестирования. Закреплены на практике умения использовать разнообразные заглушки при реализации модульного теста. Также на практике было испробована разработка через тестирование (TDD) для некоторых модулей.

Результатом выполнения данного курсового проекта стала реализация на практике умений разрабатывать и тестировать приложения. В частности, в данном курсовом проекте было реализовано модульное тестирование при помощи тестового фреймворка Codeception для программных компонентов, разработанных на языке PHP.

# Список использованных источников

1. Куликов, С.С. Тестирование программного обеспечения. Базовый курс./С.С. Куликов. – Минск: четыре четверти, 2017. – 312с.
2. Юнит-тестирование в PHP. – Режим доступа: https://habr.com/post/56289/.
3. Data Access Object (DAO). Уровень класса. – Режим доступа: https://habr.com/post/262243/.
4. Странные операторы в PHP. – Режим доступа: <https://habr.com/post/412603/>.
5. Чистый код на PHP. – Режим доступа: https://habr.com/company/mailru/blog/336788/.

# ПРИЛОЖЕНИЕ А

## Листинг исходных кодов

model/entity/CartRecord.php

**<?php  
  
require\_once "Product.php"**;  
  
**class** CartRecord  
{  
 **private $product**;  
 **private $count**;  
  
 **public function** \_\_construct($product, $count)  
 {  
 $this->setProduct($product);  
 $this->setCount($count);  
 }  
  
 **public function** getProduct()  
 {  
 **return** $this->**product**;  
 }  
  
 **public function** setProduct($product)  
 {  
 **if** ($product **instanceof** Product)  
 {  
 $this->**product** = $product;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"product"**);  
 }  
 }  
  
 **public function** getCount()  
 {  
 **return** $this->**count**;  
 }  
  
 **public function** setCount($count)  
 {  
 **if** (*is\_numeric*($count)  
 && $count >= 1 && $count <= 256)  
 {  
 $this->**count** = $count;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"count"**);  
 }  
 }  
}

model/entity/Category.php

**<?php  
  
require\_once "Identificational.php"**;  
  
**class** Category **extends** Identificational  
{  
 **private $name**;  
  
 **public function** \_\_construct($id, $name)  
 {  
 **parent**::*\_\_construct*($id);  
 $this->setName($name);  
 }  
  
  
 **public function** getName()  
 {  
 **return** $this->**name**;  
 }  
  
 **public function** setName($name)  
 {  
 **if** (*strlen*($name) >= 2 && *strlen*($name) <= 64)  
 {  
 $this->**name** = $name;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"name"**);  
 }  
 }  
}

model/entity/HistoryRecord.php

**<?php  
  
require\_once "OrderRecord.php"**;  
  
**class** HistoryRecord **extends** OrderRecord  
{  
 **private $confirmDate**;  
  
 **public function** \_\_construct($id, $user, $product, $count, $address,  
 $phone, $orderDate, $confirmDate)  
 {  
 **parent**::*\_\_construct*($id, $user, $product, $count, $address, $phone,  
 $orderDate);  
 $this->setConfirmDate($confirmDate);  
 }  
  
 **public function** getConfirmDate()  
 {  
 **return** $this->**confirmDate**;  
 }  
  
 **public function** setConfirmDate($confirmDate)  
 {  
 **if** (*date\_create\_from\_format*(**"Y-m-d"**, $confirmDate)  
 && $confirmDate <= *date*(**"Y-m-d"**)  
 && $confirmDate >= $this->getOrderDate())  
 {  
 $this->**confirmDate** = $confirmDate;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"confirmDate"**);  
 }  
 }  
}

model/entity/Identificational.php

**<?php  
  
class** Identificational  
{  
 **private $id**;  
  
 **protected function** \_\_construct($id)  
 {  
 $this->setId($id);  
 }  
  
 **public function** getId()  
 {  
 **return** $this->**id**;  
 }  
  
 **public function** setId($id)  
 {  
 **if** (*is\_numeric*($id) && $id >= 0 && $id <= *pow*(2, 32))  
 {  
 $this->**id** = $id;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"id"**);  
 }  
 }  
}

model/entity/OrderRecord.php

**<?php  
  
require\_once "User.php"**;  
**require\_once "Product.php"**;  
  
**class** OrderRecord **extends** Identificational  
{  
 **private $user**;  
 **private $product**;  
 **private $count**;  
 **private $address**;  
 **private $phone**;  
 **private $orderDate**;  
  
 **public function** \_\_construct($id, $user, $product, $count, $address,  
 $phone, $orderDate)  
 {  
 **parent**::*\_\_construct*($id);  
 $this->setUser($user);  
 $this->setProduct($product);  
 $this->setCount($count);  
 $this->setAddress($address);  
 $this->setPhone($phone);  
 $this->setOrderDate($orderDate);  
 }  
  
 **public function** getUser()  
 {  
 **return** $this->**user**;  
 }  
  
 **public function** setUser($user)  
 {  
 $this->**user** = $user;  
 }  
  
 **public function** getProduct()  
 {  
 **return** $this->**product**;  
 }  
  
 **public function** setProduct($product)  
 {  
 **if** ($product **instanceof** Product)  
 {  
 $this->**product** = $product;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"product"**);  
 }  
 }  
  
 **public function** getCount()  
 {  
 **return** $this->**count**;  
 }  
  
 **public function** setCount($count)  
 {  
 **if** (*is\_numeric*($count)  
 && $count >= 1 && $count <= 256)  
 {  
 $this->**count** = $count;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"count"**);  
 }  
 }  
  
 **public function** getAddress()  
 {  
 **return** $this->**address**;  
 }  
  
 **public function** setAddress($address)  
 {  
 **if** (*strlen*($address) >= 7  
 && *strlen*($address) <= 100)  
 {  
 $this->**address** = $address;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"address"**);  
 }  
 }  
  
 **public function** getPhone()  
 {  
 **return** $this->**phone**;  
 }  
  
 **public function** setPhone($phone)  
 {  
 **if** (*strlen*($phone) == 7 && *is\_numeric*($phone))  
 {  
 $this->**phone** = $phone;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"phone"**);  
 }  
 }  
  
 **public function** getOrderDate()  
 {  
 **return** $this->**orderDate**;  
 }  
  
 **public function** setOrderDate($orderDate)  
 {  
 **if** (*date\_create\_from\_format*(**"Y-m-d"**, $orderDate)  
 && $orderDate <= *date*(**"Y-m-d"**))  
 {  
 $this->**orderDate** = $orderDate;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"orderDate"**);  
 }  
 }  
}

model/entity/Product.php

**<?php  
  
require\_once "Category.php"**;  
  
**class** Product **extends** Identificational  
{  
 **private $name**;  
 **private $category**;  
 **private $price**;  
 **private $description**;  
  
 **public function** \_\_construct($id, $name, $category, $price, $description)  
 {  
 **parent**::*\_\_construct*($id);  
 $this->setName($name);  
 $this->setCategory($category);  
 $this->setPrice($price);  
 $this->setDescription($description);  
 }  
  
 **public function** getName()  
 {  
 **return** $this->**name**;  
 }  
  
 **public function** setName($name)  
 {  
 **if** (*strlen*($name) >= 5 && *strlen*($name) <= 128)  
 {  
 $this->**name** = $name;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"name"**);  
 }  
 }  
  
 **public function** getCategory()  
 {  
 **return** $this->**category**;  
 }  
  
 **public function** setCategory($category)  
 {  
 **if** ($category **instanceof** Category)  
 {  
 $this->**category** = $category;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"category"**);  
 }  
 }  
  
 **public function** getPrice()  
 {  
 **return** $this->**price**;  
 }  
  
 **public function** setPrice($price)  
 {  
 **if** (*is\_numeric*($price) && $price >= 0  
 && $price <= *pow*(2, 32))  
 {  
 $this->**price** = $price;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"price"**);  
 }  
 }  
  
 **public function** getDescription()  
 {  
 **return** $this->**description**;  
 }  
  
 **public function** setDescription($description)  
 {  
 **if** (*strlen*($description) <= 10000)  
 {  
 $this->**description** = $description;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"description"**);  
 }  
 }  
}

model/entity/User.php

**<?php  
  
require\_once "Identificational.php"**;  
  
**class** User **extends** Identificational  
{  
 **private $login**;  
 **private $email**;  
 **private $password**;  
  
 **public function** \_\_construct($id, $login, $email, $password)  
 {  
 **parent**::*\_\_construct*($id);  
 $this->setLogin($login);  
 $this->setEmail($email);  
 $this->setPassword($password);  
 }  
  
 **public function** getLogin()  
 {  
 **return** $this->**login**;  
 }  
  
 **public function** setLogin($login)  
 {  
 **if** (*preg\_match*(**'/^[a-z0-9\_-]{5,64}$/'**, $login))  
 {  
 $this->**login** = $login;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"login"**);  
 }  
 }  
  
 **public function** getEmail()  
 {  
 **return** $this->**email**;  
 }  
  
 **public function** setEmail($email)  
 {  
 **if** (*preg\_match*(**'/^([a-z0-9\_\.-]+)@([a-z0-9\_\.-]+)\.([a-z\.]{2,6})$/'**,  
 $email) && *strlen*($email) <= 128)  
 {  
 $this->**email** = $email;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"email"**);  
 }  
 }  
  
 **public function** getPassword()  
 {  
 **return** $this->**password**;  
 }  
  
 **public function** setPassword($password)  
 {  
 $this->**password** = $password;  
 }  
}

model/logic/AuthenticationVerifyer.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\util\dao\UserDao.php'**;  
  
**class** AuthenticationVerifyer  
{  
 **private $error\_msgs** = **array**();  
  
 **public function** \_\_construct(**array** $error\_msgs)  
 {  
 $this->**error\_msgs** = $error\_msgs;  
 }  
  
 **public function** authenticationErrors($login, $password, $mysqli) {  
 **if** (!$this->isCorrectLogin($login)) {  
 **return** $this->**error\_msgs**[**'incorrect\_login'**];  
 } **else if** (!$this->isCorrectPassword($password)) {  
 **return** $this->**error\_msgs**[**'incorrect\_password'**];  
 } **else if** (!$this->isTrueUser($login, $password, $mysqli)) {  
 **return** $this->**error\_msgs**[**'not\_user'**];  
 } **else** {  
 **return false**;  
 }  
 }  
  
 **public function** isTrueUser($login, $password, $mysqli) {  
 **if** ($this->isExistUser($login, $mysqli)) {  
 $dao = **new** UserDao($mysqli);  
 $user = $dao->getBy(**'login'**, $login)[0];  
 $hash = $user->getPassword();  
 **return** *password\_verify*($password, $hash);  
 }  
 **return false**;  
 }  
  
 **public function** isExistUser($login, $mysqli) {  
 $userDao = **new** UserDao($mysqli);  
 $users = $userDao->getBy(**'login'**, $login);  
 **return** *count*($users) != 0;  
 }  
  
 **public function** isCorrectLogin($login) {  
 **return** $this->isSameFormat(**'/^[a-z0-9\_-]{5,64}$/'**, $login);  
 }  
  
 **public function** isCorrectPassword($password) {  
 **return** (*strlen*($password) >= 5) && (*strlen*($password) <= 32);  
 }  
  
 **public function** isSameFormat($regexp, $string) {  
 **return** (bool) *preg\_match*($regexp, $string);  
 }  
}

model/logic/Authenticator.php

**<?php  
  
class** Authenticator  
{  
 **public function** authenticate($login, $error)  
 {  
 **if** (!$error) {  
 $\_SESSION[**"logged\_user"**] = $login;  
 }  
 }  
}

model/logic/Cashier.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\entity\CartRecord.php'**;  
  
**class** Cashier  
{  
 **private static** *$cashier*;  
  
 **private function** \_\_construct(){}  
  
 **public static function** getCashier()  
 {  
 **if** (**self**::*$cashier* === **null**)  
 {  
 **self**::*$cashier* = **new self**();  
 }  
 **return self**::*$cashier*;  
 }  
   
 **public function** getTotalPrice($cart)  
 {  
 $totalPrice = 0;  
 **if** (!*is\_array*($cart))  
 {  
 **throw new** InvalidArgumentException(**"not array"**);  
 }  
 **foreach** ($cart **as** $record)  
 {  
 **if** ($record **instanceof** CartRecord)  
 {  
 $totalPrice += $record->getProduct()->getPrice()  
 \* $record->getCount();  
 }  
 }  
 **return** $totalPrice;  
 }  
}

model/logic/CategoryVerifyer.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\util\dao\CategoryDao.php'**;  
  
**class** CategoryVerifyer  
{  
  
 **private $error\_msgs** = **array**();  
  
 **public function** \_\_construct(**array** $error\_msgs)  
 {  
 $this->**error\_msgs** = $error\_msgs;  
 }  
  
 **public function** categoryErrors($name, $mysqli) {  
 **if** (!$this->isCorrectName($name)) {  
 **return** $this->**error\_msgs**[**"incorrect\_name"**];  
 } **else if** ($this->isExistCategoryName($name, $mysqli)) {  
 **return** $this->**error\_msgs**[**"double\_name"**];  
 } **else** {  
 **return false**;  
 }  
 }  
  
 **public function** isExistCategoryName($name, $mysqli)  
 {  
 $dao = **new** CategoryDao($mysqli);  
 $categories = $dao->getBy(**'name'**, $name);  
 **return** *count*($categories) != 0;  
 }  
  
 **public function** isCorrectName($name)  
 {  
 **return** *strlen*($name) >= 2 && *strlen*($name) <= 64;  
 }  
}

model/logic/FaceControl.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\util\session.php'**;   
  
**class** FaceControl  
{  
 **private static** *$faceControl*;  
  
 **private function** \_\_construct(){}  
  
 **public static function** getFaceControl()  
 {  
 **if** (**self**::*$faceControl* === **null**)  
 {  
 **self**::*$faceControl* = **new self**();  
 }  
 **return self**::*$faceControl*;  
 }  
  
 **public function** isAdmin()  
 {  
 **return isset**($\_SESSION[**'logged\_user'**])  
 && $\_SESSION[**'logged\_user'**] == **'admin'**;  
 }  
  
 **public function** getOneOf($if\_guest, $if\_user, $if\_admin=**null**)  
 {  
 **if** (!**isset**($\_SESSION[**'logged\_user'**]))  
 {  
 **return** $if\_guest;  
 }  
 **else if** ($\_SESSION[**'logged\_user'**] == **'admin'**)  
 {  
 **return** $if\_admin;  
 }  
 **else** {  
 **return** $if\_user;  
 }  
 }  
}

model/logic/OrderVerifyer.php

**<?php  
  
class** OrderVerifyer  
{  
  
 **private $error\_msgs** = **array**();  
  
 **public function** \_\_construct(**array** $error\_msgs)  
 {  
 $this->**error\_msgs** = $error\_msgs;  
 }  
  
 **public function** orderErrors($address, $phone) {  
 **if** (!$this->isCorrectAddress($address)) {  
 **return** $this->**error\_msgs**[**"incorrect\_address"**];  
 } **else if** (!$this->isCorrectPhone($phone)) {  
 **return** $this->**error\_msgs**[**"incorrect\_phone"**];  
 } **else** {  
 **return false**;  
 }  
 }  
  
 **public function** isCorrectAddress($name)  
 {  
 **return** *strlen*($name) >= 7 && *strlen*($name) <= 100;  
 }  
  
 **public function** isCorrectPhone($phone)  
 {  
 **return** *strlen*($phone) == 7 && *is\_numeric*($phone);  
 }  
  
 **public function** isCorrectCount($count)  
 {  
 **return** *is\_numeric*($count) && $count >= 1 && $count <= 256;  
 }  
}

model/logic/ProductVerifyer.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\util\dao\ProductDao.php'**;   
  
**class** ProductVerifyer  
{  
  
 **private $error\_msgs** = **array**();  
  
 **public function** \_\_construct(**array** $error\_msgs)  
 {  
 $this->**error\_msgs** = $error\_msgs;  
 }  
  
 **public function** productErrors($name, $price, $description, $mysqli) {  
 **if** (!$this->isCorrectName($name)) {  
 **return** $this->**error\_msgs**[**"incorrect\_name"**];  
 } **else if** (!$this->isCorrectPrice($price)) {  
 **return** $this->**error\_msgs**[**"incorrect\_price"**];  
 } **else if** (!$this->isCorrectDescription($description)) {  
 **return** $this->**error\_msgs**[**"big\_description"**];  
 } **else if** ($this->isExistName($name, $mysqli)) {  
 **return** $this->**error\_msgs**[**"double\_name"**];  
 } **else** {  
 **return false**;  
 }  
 }  
  
 **public function** isExistName($name, $mysqli)  
 {  
 $dao = **new** ProductDao($mysqli);  
 $categories = $dao->getBy(**'name'**, $name);  
 **return** *count*($categories) != 0;  
 }  
  
 **public function** isCorrectName($name)  
 {  
 **return** *strlen*($name) >= 5 && *strlen*($name) <= 128;  
 }  
  
 **public function** isCorrectPrice($price)  
 {  
 **return** $price >= 0 && $price <= 2147483648;  
 }  
  
 **public function** isCorrectDescription($description)  
 {  
 **return** *strlen*($description) <= 10000;  
 }  
}

model/logic/RegistrationVerifyer.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\util\dao\UserDao.php'**;  
  
**class** RegistrationVerifyer  
{  
 **private $error\_msgs** = **array**();  
  
 **public function** \_\_construct(**array** $error\_msgs)  
 {  
 $this->**error\_msgs** = $error\_msgs;  
 }  
  
 **public function** registrationErrors($login, $email, $password, $password2, $mysqli) {  
 **if** (!**self**::*isCorrectLogin*($login)) {  
 **return** $this->**error\_msgs**[**"incorrect\_login"**];  
 } **else if** (!$this->isCorrectPassword($password)) {  
 **return** $this->**error\_msgs**[**"incorrect\_password"**];  
 } **else if** (!$this->isCorrectEmail($email)) {  
 **return** $this->**error\_msgs**[**"incorrect\_email"**];  
 } **else if** ($this->isExistUser($login, $mysqli)) {  
 **return** $this->**error\_msgs**[**"double\_login"**];  
 } **else if** ($this->isExistEmail($email, $mysqli)) {  
 **return** $this->**error\_msgs**[**"double\_email"**];  
 } **else if** ($password != $password2) {  
 **return** $this->**error\_msgs**[**"incorrect\_repeat"**];  
 } **else** {  
 **return false**;  
 }  
 }  
  
 **public function** isExistUser($login, $mysqli) {  
 $userDao = **new** UserDao($mysqli);  
 $users = $userDao->getBy(**'login'**, $login);  
 **return** *count*($users) != 0;  
 }  
  
 **public function** isExistEmail($email, $mysqli) {  
 $userDao = **new** UserDao($mysqli);  
 $users = $userDao->getBy(**'email'**, $email);  
 **return** *count*($users) != 0;  
 }  
  
 **public function** isCorrectLogin($login) {  
 **return self**::*isSameFormat*(**'/^[a-z0-9\_-]{5,64}$/'**, $login);  
 }  
  
 **public function** isCorrectPassword($password) {  
 **return** (*strlen*($password) >= 5) && (*strlen*($password) <= 32);  
 }  
  
 **public function** isCorrectEmail($email) {  
 **return self**::*isSameFormat*(**'/^([a-z0-9\_\.-]+)@([a-z0-9\_\.-]+)\.([a-z\.]{2,6})$/'**,  
 $email) && *strlen*($email) <= 128;  
 }  
  
 **public function** isSameFormat($regexp, $string) {  
 **return** (bool) *preg\_match*($regexp, $string);  
 }  
}

model/logic/Registrator.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\util\dao\UserDao.php'**;  
  
**class** Registrator  
{  
 **public function** registrateUser($login, $email, $password, $errors, $db)  
 {  
 **if** (!$errors) {  
 $hash = *password\_hash*($password, ***PASSWORD\_DEFAULT***);  
 $dao = **new** UserDao($db);  
 $dao->add(**new** User(0, $login, $email, $hash));  
 }  
 }  
}

model/logic/URIResolver.php

**<?php  
  
class** URIResolver  
{  
 **private static** *$uriResolver*;  
  
 **private function** \_\_construct()  
 {  
 }  
  
 **public static function** getURIResolver()  
 {  
 **if** (**self**::*$uriResolver* === **null**) {  
 **self**::*$uriResolver* = **new self**();  
 }  
 **return self**::*$uriResolver*;  
 }  
  
 **public function** getValue($name)  
 {  
 **if** ($this->hasGET($name)) {  
 **return** $\_GET[$name];  
 }  
 **return false**;  
 }  
  
 **public function** getPOSTValue($name)  
 {  
 **if** ($this->hasPOST($name)) {  
 **return** $\_POST[$name];  
 }  
 **return false**;  
 }  
  
 **public function** hasGET($name)  
 {  
 **return isset**($\_GET[$name]);  
 }  
  
 **public function** hasPOST($name)  
 {  
 **return isset**($\_POST[$name]);  
 }  
  
 **public function** setToURI($uri, $name, $value)  
 {  
 **if** (*strripos*($uri, **'?'**) !== **false**)  
 {  
 $valuess = *substr*($uri, *strripos*($uri, **'?'**) + 1);  
 $values = *explode*(**'&'**, $valuess);  
 **foreach** ($values **as** $val) {  
 **if** (*strripos*($val, $name . **'='**) === 0) {  
 $replace = $name . **'='** . $value;  
 **return** *str\_replace*($val, $replace, $uri);  
 }  
 }  
 **return** $uri . **'&'** . $name . **'='** . $value;  
 }  
 **else** {  
 **return** $uri . **'?'** . $name . **'='** . $value;  
 }  
 }  
  
 **public function** clearURI($uri)  
 {  
 **if** (*strripos*($uri, **'?'**) !== **false**) {  
 **return** *substr*($uri, 0, *strripos*($uri, **'?'**));  
 }  
 **return** $uri;  
 }  
  
 **public function** getOnlyValues($uri)  
 {  
 **return** *substr*($uri, *strripos*($uri, **'?'**));  
 }  
  
 **public function** unsetFromURI($uri, $name)  
 {  
 $valuess = *substr*($uri, *strripos*($uri, **'?'**) + 1);  
 $values = *explode*(**'&'**, $valuess);  
 $valname = **""**;  
 **foreach** ($values **as** $val) {  
 **if** (*strripos*($val, $name . **'='**) === 0) {  
 $valname = $val;

**break**;  
 }  
 }  
 **if** (*strripos*($valuess, $valname) === 0 && *strripos*($valuess, **"&"**)) {  
 **return** *str\_replace*($valname . **'&'**, **''**, $uri);  
 } **else if** (*strripos*($valuess, $valname) === 0) {  
 **return** *str\_replace*(**'?'** . $valname, **''**, $uri);  
 } **else** {  
 **return** *str\_replace*(**'&'** . $valname, **''**, $uri);  
 }  
 }  
}

model/util/dao/CartDao.php

**<?php  
  
require\_once "IDao.php"**;  
**require\_once "ProductDao.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\\entity\CartRecord.php"**;  
  
**class** CartDao **implements** IDao  
{  
 **public function** add($record)  
 {  
 **if** (!*is\_array*($record))  
 {  
 **throw new** InvalidArgumentException(**"not array"**);  
 }  
 **foreach** ($record **as** $rec)  
 {  
 **if** ($rec **instanceof** CartRecord)  
 {  
 $key = **"product\_id:"** . $rec->getProduct()->getId();  
 $count = $rec->getCount();  
 $\_SESSION[$key] = $count;  
 }  
 }  
 }  
  
 **public function** getAll()  
 {  
 $records = **array**();  
 **foreach** (*array\_keys*($\_SESSION) **as** $key)  
 {  
 **if** (*substr\_count*($key, **"product\_id:"**) == 1)  
 {  
 $prod\_id = *str\_replace*(**"product\_id:"**, **""**, $key);  
 $prod\_count = $\_SESSION[$key];  
 **require "D:\Workspace\UnitTesting\courseproject\model\util\connectDB.php"**;  
 $productDao = **new** ProductDao($mysqli);  
 $product =$productDao->getBy(**'id'**, $prod\_id)[0];  
 $records[] = **new** CartRecord($product, $prod\_count);  
 }  
 }  
 **return** $records;  
 }  
  
 **public function** getBy($id)  
 {  
 $key = **"product\_id:"** . $id;  
 **if** (**isset**($\_SESSION[$key])) {  
 $prod\_count = $\_SESSION[$key];  
 **require "D:\Workspace\UnitTesting\courseproject\model\util\connectDB.php"**;  
 $productDao = **new** ProductDao($mysqli);  
 $product = $productDao->getBy(**'id'**, $id)[0];  
 **return new** CartRecord($product, $prod\_count);  
 }  
 **return false**;  
 }  
  
 **public function** update($record)  
 {  
 $this->add($record);  
 }  
  
 **public function** delete($record)  
 {  
 **if** ($record **instanceof** CartRecord)  
 {  
 **unset**($\_SESSION[**"product\_id:"** . $record->getProduct()->getId()]);  
 }  
 }  
  
 **public function** deleteById($id)  
 {  
 **if** (**isset**($\_SESSION[**"product\_id:"** . $id])) {  
 **unset**($\_SESSION[**"product\_id:"** . $id]);  
 }  
 }  
}

model/util/dao/CategoryDao.php

**<?php  
  
require\_once "IdentificationalDao.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\\entity\Category.php"**;  
  
**class** CategoryDao **extends** IdentificationalDao  
{  
 **public function** \_\_construct($db)  
 {  
 **parent**::*\_\_construct*($db, **'categories'**);  
 }  
  
 **public function** add($record)  
 {  
 **if** ($record **instanceof** Category)  
 {  
 $name = $record->getName();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"INSERT INTO `**$table\_name**` (`name`)   
 VALUES ('**$name**')"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"category"**);  
 }  
 }  
  
 **public function** update($record)  
 {  
 **if** ($record **instanceof** Category)  
 {  
 $id = $record->getId();  
 $name = $record->getName();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"UPDATE `**$table\_name**` SET `name` = '**$name**' WHERE `id` = '**$id**'"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"category"**);  
 }  
 }  
  
 **protected function** convert($rec)  
 {  
 **return new** Category($rec[**'id'**], $rec[**'name'**]);  
 }  
}

model/util/dao/DBDao.php

**<?php  
  
require\_once "IDao.php"**;  
  
**abstract class** DBDao **implements** IDao  
{  
 **private $db**;  
 **private $table\_name**;  
  
 **protected abstract function** convert($rec);  
  
 **public function** \_\_construct($mysqli, $table\_name)  
 {  
 $this->**db** = $mysqli;  
 $this->**table\_name** = $table\_name;  
 }  
  
 **public function** getAll()  
 {  
 $items = **array**();  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$this->**table\_name`"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
 }  
  
 **public function** getAllSearched($column, $pattern=**"%"**, $sort\_column=**""**, $desc=**false**)  
 {  
 $items = **array**();  
 **if** ($sort\_column == **""**)  
 {  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$this->**table\_name` WHERE `**$column**` LIKE '**$pattern**'"**);  
 }  
 **else if** ($desc)  
 {  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$this->**table\_name` WHERE `**$column**` LIKE '**$pattern**'  
 ORDER BY `**$sort\_column**` DESC"**);  
 }  
 **else** {  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$this->**table\_name` WHERE `**$column**` LIKE '**$pattern**'  
 ORDER BY `**$sort\_column**`"**);  
 }  
  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
 }  
  
 **public function** getBy($column\_name, $value)  
 {  
 $items = **array**();  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$this->**table\_name` WHERE `**$column\_name**` = '**$value**'"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
 }  
  
 **public function** getColumnBy($column, $value, $get\_column)  
 {  
 $items = **array**();  
 $records = $this->getDb()->query(**"SELECT `**$get\_column**` FROM `**$this->**table\_name` WHERE `**$column**` = '**$value**'"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $rec[$get\_column];  
 }  
 **return** $items;  
 }  
  
 **public function** updateColumnBy($column, $value, $update\_column, $update\_value)  
 {  
 $this->getDb()->query(**"UPDATE `**$this->**table\_name`   
 SET `**$update\_column**` = '**$update\_value**' WHERE `**$column**` = '**$value**'"**);  
 }  
  
 **public function** deleteBy($column\_name, $value)  
 {  
 $this->getDb()->query(**"DELETE FROM `**$this->**table\_name` WHERE `**$column\_name**` = '**$value**'"**);  
 }  
  
 **public function** getDb()  
 {  
 **return** $this->**db**;  
 }  
  
 **public function** setDb($db)  
 {  
 **if** ($db **instanceof** mysqli)  
 {  
 $this->**db** = $db;  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**'db'**);  
 }  
 }  
  
 **public function** getTableName()  
 {  
 **return** $this->**table\_name**;  
 }  
  
 **public function** setTableName($table\_name)  
 {  
 $this->**table\_name** = $table\_name;  
 }  
}

model/util/dao/HistoryRecordDao.php

**<?php  
  
require\_once "IdentificationalDao.php"**;  
**require\_once "UserDao.php"**;  
**require\_once "ProductDao.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\\entity\HistoryRecord.php"**;  
  
**class** HistoryRecordDao **extends** IdentificationalDao  
{  
 **public function** \_\_construct($db)  
 {  
 **parent**::*\_\_construct*($db, **'log'**);  
 }  
  
 **protected function** convert($rec)  
 {  
 $productDao = **new** ProductDao($this->getDb());  
 $userDao = **new** UserDao($this->getDb());  
 $user = ($rec[**'user\_id'**] == 0) ? 0 : $userDao->getBy(**'id'**, $rec[**'user\_id'**])[0];  
 $product = $productDao->getBy(**'id'**, $rec[**'product\_id'**])[0];  
 **return new** HistoryRecord($rec[**'id'**], $user, $product,  
 $rec[**'count'**], $rec[**'address'**], $rec[**'phone'**], $rec[**'order\_date'**], $rec[**'confirm\_date'**]);  
 }  
  
 **public function** add($record)  
 {  
 **if** ($record **instanceof** HistoryRecord)  
 {  
 $user\_id = ($record->getUser() == **null**) ? **null** : $record->getUser()->getId();  
 $product\_id = $record->getProduct()->getId();  
 $count = $record->getCount();  
 $address = $record->getAddress();  
 $phone = $record->getPhone();  
 $order\_date = $record->getOrderDate();  
 $confirm\_date = $record->getConfirmDate();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"INSERT INTO `**$table\_name**` (`user\_id`, `product\_id`, `count`, `address`,  
 `phone`, `order\_date`, `confirm\_date`)   
 VALUES ('**$user\_id**', '**$product\_id**', '**$count**', '**$address**', '**$phone**', '**$order\_date**', '**$confirm\_date**')"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"History record"**);  
 }  
 }  
  
 **public function** update($record)  
 {  
 **if** ($record **instanceof** HistoryRecord)  
 {  
 $id = $record->getId();  
 $user\_id = ($record->getUser() == **null**) ? **null** : $record->getUser()->getId();  
 $product\_id = $record->getProduct()->getId();  
 $count = $record->getCount();  
 $address = $record->getAddress();  
 $phone = $record->getPhone();  
 $order\_date = $record->getOrderDate();  
 $confirm\_date = $record->getConfirmDate();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"UPDATE `**$table\_name**` SET `user\_id` = '**$user\_id**', `product\_id` = '**$product\_id**',  
 `count` = '**$count**', `address` = '**$address**', `phone` = '**$phone**',   
 `order\_date` = '**$order\_date**', `confirm\_date` = '**$confirm\_date**' WHERE `id` = '**$id**'"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"History record"**);  
 }  
 }  
  
  
 **public function** getAll()  
 {  
 $items = **array**();  
 $table\_name = $this->getTableName();  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$table\_name**` WHERE `confirm\_date` IS NOT NULL"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
 }  
  
 **public function** getBy($column\_name, $value)  
 {  
 $items = **array**();  
 $table\_name = $this->getTableName();  
 $records = $this->getDb()->query(**"SELECT** *\** **FROM `**$table\_name**` WHERE `**$column\_name**` = '**$value**'   
 AND `confirm\_date` IS NOT NULL"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
 }  
  
 **public function** getColumnBy($column, $value, $get\_column)  
 {  
 $items = **array**();  
 $table\_name = $this->getTableName();  
 $records = $this->getDb()->query(**"SELECT `**$get\_column**` FROM `**$table\_name**` WHERE `**$column**` = '**$value**'  
 AND `confirm\_date` IS NOT NULL"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $rec[$get\_column];  
 }  
 **return** $items;  
 }  
  
 **public function** updateColumnBy($column, $value, $update\_column, $update\_value)  
 {  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"UPDATE `**$table\_name**`   
 SET `**$update\_column**` = '**$update\_value**' WHERE `**$column**` = '**$value**' AND `confirm\_date` IS NOT NULL"**);  
 }  
  
 **public function** deleteBy($column\_name, $value)  
 {  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"DELETE FROM `**$table\_name**` WHERE `**$column\_name**` = '**$value**' AND `confirm\_date` IS NOT NULL"**);  
 }  
}

model/util/dao/IDao.php

**<?php  
  
interface** IDao  
{  
 **public function** getAll();  
 **public function** add($record);  
 **public function** update($record);  
 **public function** delete($record);  
}

model/util/dao/IdentificationalDao.php

**<?php  
  
require\_once "DBDao.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\\entity\Identificational.php"**;  
  
**abstract class** IdentificationalDao **extends** DBDao  
{  
  
 **public function** \_\_construct($mysqli, $table\_name)  
 {  
 **parent**::*\_\_construct*($mysqli, $table\_name);  
 }  
  
 **public function** delete($record)  
 {  
 **if** ($record **instanceof** Identificational)  
 {  
 $id = $record->getId();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"DELETE FROM `**$table\_name**` WHERE `id` = '**$id**'"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"Non-id-able"**);  
 }  
 }  
}

model/util/dao/OrderRecordDao.php

**<?php  
  
require\_once "IdentificationalDao.php"**;  
**require\_once "ProductDao.php"**;  
**require\_once "UserDao.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\\entity\OrderRecord.php"**;  
  
**class** OrderRecordDao **extends** IdentificationalDao  
{  
 **public function** \_\_construct($db)  
 {  
 **parent**::*\_\_construct*($db, **'log'**);  
 }  
  
 **protected function** convert($rec)  
 {  
 $productDao = **new** ProductDao($this->getDb());  
 $userDao = **new** UserDao($this->getDb());  
 $user = ($rec[**'user\_id'**] == 0) ? 0 : $userDao->getBy(**'id'**, $rec[**'user\_id'**])[0];  
 $product = $productDao->getBy(**'id'**, $rec[**'product\_id'**])[0];  
 **return new** OrderRecord($rec[**'id'**], $user, $product,  
 $rec[**'count'**], $rec[**'address'**], $rec[**'phone'**], $rec[**'order\_date'**]);  
 }  
  
 **public function** add($record)  
 {  
 **if** ($record **instanceof** OrderRecord)  
 {  
 $user\_id = ($record->getUser() == **null**) ? **null** : $record->getUser()->getId();  
 $product\_id = $record->getProduct()->getId();  
 $count = $record->getCount();  
 $address = $record->getAddress();  
 $phone = $record->getPhone();  
 $order\_date = $record->getOrderDate();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"INSERT INTO `**$table\_name**` (`user\_id`, `product\_id`, `count`, `address`,  
 `phone`, `order\_date`)   
 VALUES ('**$user\_id**', '**$product\_id**', '**$count**', '**$address**', '**$phone**', '**$order\_date**')"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"Order record"**);  
 }  
 }  
  
 **public function** update($record)  
 {  
 **if** ($record **instanceof** OrderRecord)  
 {  
 $id = $record->getId();  
 $user\_id = ($record->getUser() == **null**) ? **null** : $record->getUser()->getId();  
 $product\_id = $record->getProduct()->getId();  
 $count = $record->getCount();  
 $address = $record->getAddress();  
 $phone = $record->getPhone();  
 $order\_date = $record->getOrderDate();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"UPDATE `**$table\_name**` SET `user\_id` = '**$user\_id**', `product\_id` = '**$product\_id**',  
 `count` = '**$count**', `address` = '**$address**', `phone` = '**$phone**',   
 `order\_date` = '**$order\_date**' WHERE `id` = '**$id**'"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"Order record"**);  
 }  
 }  
  
  
 **public function** getAll()  
 {  
 $items = **array**();  
 $table\_name = $this->getTableName();  
 $records = $this->getDb()->query(**"SELECT \* FROM `**$table\_name**` WHERE `confirm\_date` IS NULL"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
 }  
  
 **public function** getBy($column\_name, $value)  
 {  
 $items = **array**();  
 $table\_name = $this->getTableName();  
 $records = $this->getDb()->query(**"SELECT \* FROM `**$table\_name**` WHERE `**$column\_name**` = '**$value**'   
 AND `confirm\_date` IS NULL"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $this->convert($rec);  
 }  
 **return** $items;  
 }  
  
 **public function** getColumnBy($column, $value, $get\_column)  
 {  
 $items = **array**();  
 $table\_name = $this->getTableName();  
 $records = $this->getDb()->query(**"SELECT `**$get\_column**` FROM `**$table\_name**` WHERE `**$column**` = '**$value**'  
 AND `confirm\_date` IS NULL"**);  
 **while** ($rec = $records->fetch\_assoc())  
 {  
 $items[] = $rec[$get\_column];  
 }  
 **return** $items;  
 }  
  
 **public function** updateColumnBy($column, $value, $update\_column, $update\_value)  
 {  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"UPDATE `**$table\_name**`   
 SET `**$update\_column**` = '**$update\_value**' WHERE `**$column**` = '**$value**' AND `confirm\_date` IS NULL"**);  
 }  
  
 **public function** deleteBy($column\_name, $value)  
 {  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"DELETE FROM `**$table\_name**` WHERE `**$column\_name**` = '**$value**' AND `confirm\_date` IS NULL"**);  
 }  
}

model/util/dao/ProductDao.php

**<?php  
  
require\_once "CategoryDao.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\\entity\Product.php"**;  
  
**class** ProductDao **extends** IdentificationalDao  
{  
 **public function** \_\_construct($db)  
 {  
 **parent**::*\_\_construct*($db, **'products'**);  
 }  
  
 **protected function** convert($rec)  
 {  
 $categoryDao = **new** CategoryDao($this->getDb());  
 $category = $categoryDao->getBy(**'id'**, $rec[**'category\_id'**])[0];  
 **return new** Product($rec[**'id'**], $rec[**'name'**], $category,  
 $rec[**'price'**], $rec[**'description'**]);  
 }  
  
 **public function** add($record)  
 {  
 **if** ($record **instanceof** Product)  
 {  
 $name = $record->getName();  
 $price = $record->getPrice();  
 $category\_id = $record->getCategory()->getId();  
 $description = $record->getDescription();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"INSERT INTO `**$table\_name**` (`name`, `price`, `category\_id`, `description`)   
 VALUES ('**$name**', '**$price**', '**$category\_id**', '**$description**')"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"product"**);  
 }  
 }  
  
 **public function** update($record)  
 {  
 **if** ($record **instanceof** Product)  
 {  
 $id = $record->getId();  
 $name = $record->getName();  
 $price = $record->getPrice();  
 $category\_id = $record->getCategory()->getId();  
 $description = $record->getDescription();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"UPDATE `**$table\_name**` SET `name` = '**$name**', `price` = '**$price**',  
 `category\_id` = '**$category\_id**', `description` = '**$description**' WHERE `id` = '**$id**'"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"product"**);  
 }  
 }  
}

model/util/dao/UserDao.php

**<?php  
  
require\_once "IdentificationalDao.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\\entity\User.php"**;  
  
**class** UserDao **extends** IdentificationalDao  
{  
 **public function** \_\_construct($db)  
 {  
 **parent**::*\_\_construct*($db, **'users'**);  
 }  
  
 **protected function** convert($rec)  
 {  
 **return new** User($rec[**'id'**], $rec[**'login'**], $rec[**'email'**], $rec[**'password'**]);  
 }  
  
 **public function** add($record)  
 {  
 **if** ($record **instanceof** User)  
 {  
 $login = $record->getLogin();  
 $email = $record->getEmail();  
 $password = $record->getPassword();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"INSERT INTO `**$table\_name**` (`login`, `email`, `password`)   
 VALUES ('**$login**', '**$email**', '**$password**')"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"user"**);  
 }  
 }  
  
 **public function** update($record)  
 {  
 **if** ($record **instanceof** User)  
 {  
 $id = $record->getId();  
 $login = $record->getLogin();  
 $email = $record->getEmail();  
 $password = $record->getPassword();  
 $table\_name = $this->getTableName();  
 $this->getDb()->query(**"UPDATE `**$table\_name**` SET `login` = '**$login**', `email` = '**$email**',  
 `password` = '**$password**' WHERE `id` = '**$id**'"**);  
 }  
 **else** {  
 **throw new** InvalidArgumentException(**"user"**);  
 }  
 }  
}

model/util/awayIfNotAdmin.php

**<?php  
  
require\_once "utilFunc.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\logic\FaceControl.php"**;  
  
**if** (!FaceControl::*getFaceControl*()->isAdmin())  
{  
 gotoPage(**'http://localhost:63342/courseproject/index.php'**);  
}

model/util/connectDB.php

**<?php**$mysqli = **new** mysqli(**'localhost'**, **'mysql'**, **'mysql'**, **'e\_shop'**);  
  
**if** ($mysqli->**connect\_error**) {  
 **die**($mysqli->**connect\_errno**);  
}

model/util/logout.php

**<?php  
  
require\_once "utilFunc.php"**;  
logout();

model/util/session.php

**<?php***session\_start*();

model/util/utilFunc.php

**<?php  
  
require\_once "session.php"**;  
  
**function** gotoPage($url)  
{  
 **echo "<script>window.location = '**$url**'</script>"**;  
}  
  
**function** logout()  
{  
 **unset**($\_SESSION[**"logged\_user"**]);  
 gotoPage(**'http://localhost:63342/courseproject/index.php'**);  
}

view/css/admin.css

.**admin\_menu** {  
 **margin-top**: 2.5**vw**;  
 **width**: 26**vw**;  
 **margin-left**: 37**vw**;  
 **text-align**: **center**;  
 **color**: **rgb**(62, 82, 197);  
 **border**: 0.9**vw solid blue**;  
 **border-radius**: 0.5**vw**;  
}  
  
.**admin\_menu a** {  
 **font-size**: 2**vw**;  
}

view/css/categories.css

.**categories\_panel** {  
 **margin-left**: 20**vw**;  
 **width**: 60**vw**;  
}  
  
.**category img** {  
 **width**: 30**vw**;  
 **height**: 30**vw**;  
}  
  
.**category** {  
 **margin-top**: 2.5**vw**;  
 **width**: 60**vw**;  
 **height**: 30**vw**;  
}  
  
.**category\_txt** {  
 **width**: 30**vw**;  
 **height**: 3**vw**;  
 **margin-top**: 12**vw**;  
 **text-align**: **center**;  
 **font-size**: 3**vw**;  
}  
  
.**right\_block** {  
 **margin-left**: 30**vw**;  
 **margin-top**: -30**vw**;  
}  
  
.**category\_fill** {  
 **width**: 30**vw**;  
 **height**: 15**vw**;  
}

view/css/items.css

.**list** {  
 **margin-left**: 20**vw**;  
 **width**: 60**vw**;  
}  
  
.**item** {  
 **margin-top**: 1**vw**;  
 **border**: 0.1**vw solid rgb**(222, 211, 220);  
 **width**: 60**vw**;  
 **height**: 4.2**vw**;  
 **font-size**: 1.5**vw**;  
}  
  
.**item\_txt** {  
 **width**: 30**vw**;  
 **height**: 4.2**vw**;  
 **border-radius**: 0.1**vw**;  
}  
  
.**item\_control** {  
 **margin-top**: -2.1**vw**;  
 **margin-left**: 30**vw**;  
 **width**: 30**vw**;  
 **height**: 4.2**vw**;  
 **color**: **rgba**(176, 121, 208, 0.59);  
 **text-align**: **right**;  
}

view/css/product.css

.**product\_panel** {  
 **margin-left**: 20**vw**;  
 **width**: 60**vw**;  
}  
  
.**product** {  
 **margin-top**: 2.5**vw**;  
 **width**: 60**vw**;  
 **height**: 20**vw**;  
 **border**: 0.1**vw solid rgb**(222, 211, 220);  
 **border-radius**: 0.1**vw**;  
}  
  
.**product\_pic**, .**product img** {  
 **width**: 15**vw**;  
 **height**: 15**vw**;  
}  
  
.**product\_pic** {  
 **margin-left**: 5**vw**;  
 **margin-top**: 2.5**vw**;  
}  
  
.**info\_block** {  
 **width**: 30**vw**;  
 **height**: 15**vw**;  
 **text-align**: **right**;  
 **margin-left**: 25**vw**;  
 **margin-top**: -15**vw**;  
}  
  
.**name\_block**, .**price\_block**, .**control\_block** {  
 **width**: 30**vw**;  
 **height**: 5**vw**;  
}  
  
.**name\_block** {  
 **font-size**: 2**vw**;  
}  
  
.**price\_block** {  
 **font-size**: 3**vw**;  
}  
  
.**product\_page** {  
 **width**: 60**vw**;  
 **margin-left**: 20**vw**;  
}  
  
.**page\_pic**, .**product\_page img** {  
 **width**: 27.5**vw**;  
 **height**: 27.5**vw**;  
}  
  
.**name\_page** {  
 **width**: 30**vw**;  
 **height**: 9**vw**;  
 **margin-top**: -27.5**vw**;  
 **margin-left**: 30**vw**;  
 **text-align**: **right**;  
 **font-size**: 4**vw**;  
}  
  
.**price\_page** {  
 **width**: 30**vw**;  
 **height**: 9**vw**;  
 **margin-left**: 30**vw**;  
 **text-align**: **right**;  
 **font-size**: 5**vw**;  
}  
  
.**cntrl\_page** {  
 **width**: 30**vw**;  
 **height**: 9**vw**;  
 **margin-left**: 30**vw**;  
 **text-align**: **right**;  
 **font-size**: 3**vw**;  
}

view/css/style.css

**body** {  
 **font-family**: **'Gill Sans'**, **'Gill Sans MT'**, **Calibri**, **'Trebuchet MS'**, **sans-serif**;  
 **background-color**: **rgba**(224, 241, 233, 0.58);  
}  
  
**a** {  
 **color**: **rgb**(62, 82, 197);  
 **text-decoration**: **none**;  
}  
  
**#header\_title**, .**controls\_right** {  
 **height**: 5**vw**;  
 **background-color**: **rgb**(62, 82, 197);  
 **font-size**: 3**vw**;  
 **font-family**: **cursive**;  
}  
  
**header a** {   
 **color**: **white**;  
}  
  
.**header\_link** {  
 **text-align**: **center**;  
}  
  
.**controls\_right** {  
 **margin-left**: 68.6**vw**;  
 **margin-top**: -5**vw**;  
 **height**: 5**vw**;  
 **width**: 29.4**vw**;  
}  
  
.**registration\_link** {  
 **height**: 5**vw**;  
 **width**: 19.4**vw**;  
}  
  
.**autho\_link** {  
 **margin-top**: -5**vw**;  
 **margin-left**: 19.4**vw**;  
 **height**: 5**vw**;  
 **width**: 10**vw**;  
}  
  
**#header\_title** {  
 **height**: 5**vw**;  
 **width**: 68.6**vw**;  
}  
  
**#title\_name** {  
 **margin-left**: 2**vw**;  
   
 **height**: 5**vw**;  
 **width**: 53.9**vw**;  
 **text-align**: **left**;  
}  
  
**#basket** {  
 **margin-top**: -5**vw**;  
 **margin-left**: 53.9**vw**;  
 **height**: 5**vw**;  
 **width**: 14.7**vw**;  
}  
  
**footer** {  
 **margin-top**: 5**vw**;  
 **color**: **dimgrey**;  
 **text-align**: **center**;  
}  
  
**footer a** {  
 **color**: **dimgray**;  
}  
  
.**user\_identify** {  
 **margin-top**: 2.5**vw**;  
 **margin-left**: 43**vw**;  
 **width**: 14**vw**;  
 **text-align**: **left**;  
 **font-size**: 1**vw**;  
 **color**: **rgb**(62, 82, 197);  
}  
  
.**user\_identify h2** {  
 **text-align**: **center**;  
}  
  
**input** {  
 **width**: 14**vw**;  
 **border**: 0.01**vw solid blue**;  
}

view/authorizationControl.html

<**div class="controls\_right"**>  
 <**div class="registration\_link"**><**a href="registration.php"**>Регистрация</**a**></**div**>  
 <**div class="autho\_link"**><**a href="authentication.php"**>Вход</**a**></**div**>  
</**div**>

view/CartView.php

**<?php  
  
require\_once "D:\Workspace\UnitTesting\courseproject\model\util\dao\CartDao.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\logic\URIResolver.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\logic\Cashier.php"**;  
  
**class** CartView  
{  
 **public function** view()  
 {  
 $dao = **new** CartDao();  
 $cart = $dao->getAll();  
 $uriRes = URIResolver::*getURIResolver*();  
 **foreach** ($cart **as** $record) {  
 $id = $record->getProduct()->getId();  
 $name = $record->getProduct()->getName();  
 $price = $record->getProduct()->getPrice();  
 $count = $record->getCount();  
 $delete\_link = $uriRes->setToURI($uriRes->getOnlyValues($\_SERVER[**'REQUEST\_URI'**]),  
 **'delete\_rec'**, $id);  
 **echo "<div class='item'>  
 <div class='item\_txt'>**$name **(x** $count**)** $price**$</div>  
 <div class='item\_control'><a href='**$delete\_link**'>Удалить</a></div>  
 </div>"**;  
 }  
 **if** (*count*($cart) == 0)  
 {  
 **echo "<a href = 'http://localhost:63342/courseproject/index.php'>  
 Корзина пуста. Добавьте что-нибудь</a>"**;  
 }  
 **else** {  
 $price = Cashier::*getCashier*()->getTotalPrice($cart);  
 **echo "<p>Общая стоимость:** $price**$</p>"**;  
 **echo "<a href = '/courseproject/createOrder.php'>  
 Оформить заказ</a>"**;  
 }  
 }  
}

view/CategoryView.php

**<?php  
  
require\_once "D:\Workspace\UnitTesting\courseproject\model\util\dao\CategoryDao.php"**;  
  
**class** CategoryView  
{  
 **public function** consumerView($db)  
 {  
 $dao = **new** CategoryDao($db);  
 $categories = $dao->getAll();  
 $name\_left = **true**;  
 **foreach** ($categories **as** $category)  
 {  
 $id = $category->getId();  
 $name = $category->getName();  
 **require\_once "paths.php"**;  
 $picture = $PIC\_CATEGORIES\_PATH . $dao->getColumnBy(**'id'**, $id, **'picture\_path'**)[0];  
 $blocks = **""**;  
 **if** ($name\_left)  
 {  
 $blocks = **"<div class=\"category\_txt\">**$name**</div><div class='category\_fill'></div>  
 <div class='right\_block'><img src=\"**$picture**\" /></div>"**;  
 }  
 **else** {  
 $blocks = **"<div><img src=\"**$picture**\" /></div>  
 <div class='right\_block'><div class=\"category\_txt\">**$name**</div></div>"**;  
 }  
 $name\_left = !$name\_left;  
 **echo "<div class=\"category\">  
 <a href=\"goods.php?category=**$id**\">"** . $blocks .  
 **"</a>  
 </div>"**;  
 }  
 }  
  
 **public function** adminView($db)  
 {  
 $dao = **new** CategoryDao($db);  
 $categories = $dao->getAll();  
 **foreach** ($categories **as** $category)  
 {  
 $id = $category->getId();  
 $name = $category->getName();  
 **echo "<div class='item'>  
 <div class='item\_txt'>**$name**</div>  
 <div class='item\_control'><a href='?delete\_category=**$id**'>Удалить</a></div>  
 </div>"**;  
 }  
 }  
  
 **public function** adminSelectView($db)  
 {  
 $dao = **new** CategoryDao($db);  
 $categories = $dao->getAll();  
 **foreach** ($categories **as** $category)  
 {  
 $id = $category->getId();  
 $name = $category->getName();  
 **echo "  
 <div><a href='goodSet.php?category=**$id**'>**$name**</a></div>  
 "**;  
 }  
 }  
}

view/footer.html

<**p id="about\_creator"**><**a href="https://vk.com/ruslan\_bahirau"**>Bahirau Ruslan</**a**>, 2018 - ...</**p**>

view/header.html

<**div id="header\_title"**>  
 <**div id="title\_name"**><**a href="/courseproject"**>Пекарь</**a**></**div**>  
 <**div id="basket"**><**a href="basket.php" class="header\_link"**>Корзина</**a**></**div**>  
</**div**>

view/HistoryView.php

**<?php  
  
require\_once "D:\Workspace\UnitTesting\courseproject\model\util\dao\HistoryRecordDao.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\logic\URIResolver.php"**;  
  
**class** HistoryView  
{  
 **public function** viewAll($db) {  
 $dao = **new** HistoryRecordDao($db);  
 $history = $dao->getAll();  
 $uriRes = URIResolver::*getURIResolver*();  
 $clear\_link = $uriRes->setToURI($uriRes->getOnlyValues($\_SERVER[**'REQUEST\_URI'**]),  
 **'clear\_history'**, 1);  
 **echo "<a href='**$clear\_link**'>Очистить историю</a>"**;  
 **foreach** ($history **as** $record)  
 {  
 $id = $record->getId();  
 $user\_name = ($record->getUser() == **null**)  
 ? **'none(guest)'** : $record->getUser()->getLogin();  
 $product\_name = $record->getProduct()->getName();  
 $price = $record->getProduct()->getPrice() . **'$'**;  
 $count = $record->getCount();  
 $order\_date = $record->getOrderDate();  
 $confirm\_date = $record->getConfirmDate();  
 $delete\_link = $uriRes->setToURI($uriRes->getOnlyValues($\_SERVER[**'REQUEST\_URI'**]),  
 **'delete\_history'**, $id);  
 **echo "<div class='item'>  
 <div class='item\_txt'>**$product\_name**, пользователь:** $user\_name**,** $price**$ x** $count**,  
 дата оформления:** $order\_date**, дата подтверждения:** $confirm\_date**</div>  
 <div class='item\_control'><a href='**$delete\_link**'>Удалить</a></div>  
 </div>"**;  
 }  
 }  
  
 **public function** viewAllByUser($db, $user\_id) {  
 $dao = **new** HistoryRecordDao($db);  
 $history = $dao->getBy(**'user\_id'**, $user\_id);  
 $uriRes = URIResolver::*getURIResolver*();  
 $clear\_link = $uriRes->setToURI($uriRes->getOnlyValues($\_SERVER[**'REQUEST\_URI'**]),  
 **'clear\_history'**, $user\_id);  
 **echo "<a href='**$clear\_link**'>Очистить историю</a>"**;  
 **foreach** ($history **as** $record)  
 {  
 $id = $record->getId();  
 $product\_name = $record->getProduct()->getName();  
 $price = $record->getProduct()->getPrice() . **'$'**;  
 $count = $record->getCount();  
 $order\_date = $record->getOrderDate();  
 $confirm\_date = $record->getConfirmDate();  
 $delete\_link = $uriRes->setToURI($uriRes->getOnlyValues($\_SERVER[**'REQUEST\_URI'**]),  
 **'delete\_history'**, $id);  
 **echo "<div class='item'>  
 <div class='item\_txt'>**$product\_name**,** $price**$ x** $count**, дата оформления:** $order\_date**,   
 дата подтверждения:** $confirm\_date**</div>  
 <div class='item\_control'><a href='**$delete\_link**'>Удалить</a></div>  
 </div>"**;  
 }  
 }  
}

view/OrderView.php

**<?php  
  
require\_once "D:\Workspace\UnitTesting\courseproject\model\util\dao\OrderRecordDao.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\logic\URIResolver.php"**;  
  
**class** OrderView  
{  
 **public function** view($db)  
 {  
 $dao = **new** OrderRecordDao($db);  
 $orders = $dao->getAll();  
 $uriRes = URIResolver::*getURIResolver*();  
 **foreach** ($orders **as** $order)  
 {  
 $id = $order->getId();  
 $user\_name = ($order->getUser() == **null**)  
 ? **'none(guest)'** : $order->getUser()->getLogin();  
 $product\_name = $order->getProduct()->getName();  
 $product\_price = $order->getProduct()->getPrice();  
 $count = $order->getCount();  
 $date = $order->getOrderDate();  
 $submit\_link = $uriRes->setToURI($uriRes->getOnlyValues($\_SERVER[**'REQUEST\_URI'**]),  
 **'submit\_order'**, $id);  
 $delete\_link = $uriRes->setToURI($uriRes->getOnlyValues($\_SERVER[**'REQUEST\_URI'**]),  
 **'delete\_order'**, $id);  
 **echo "<div class='item'>  
 <div class='item\_txt'>Пользователь:** $user\_name**,** $product\_name $product\_price**$ x** $count**,  
 дата оформления:** $date**</div>  
 <div class='item\_control'>  
 <a href='**$submit\_link**'>Подтвердить</a>  
 <a href='**$delete\_link**'>Удалить</a>  
 </div>  
 </div>"**;  
 }  
 }  
}

view/paths.php

**<?php**$PIC\_CATEGORIES\_PATH = **"view\\pictures\\categories\\"**;  
$PIC\_PRODUCTS\_PATH = **"view\\pictures\\goods\\"**;

view/ProductView.php

**<?php  
  
require\_once "D:\Workspace\UnitTesting\courseproject\model\util\dao\ProductDao.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\logic\URIResolver.php"**;  
  
**class** ProductView  
{  
 **public function** viewPage($db, $id, $isAdmin) {  
 $dao = **new** ProductDao($db);  
 $product = $dao->getBy(**'id'**, $id)[0];  
 **require "paths.php"**;  
 $picture = $PIC\_PRODUCTS\_PATH  
 . $dao->getColumnBy(**'id'**, $id, **'picture\_path'**)[0];  
 $name = $product->getName();  
 $price = $product->getPrice();  
 $description = $product->getDescription();  
 $uriRes = URIResolver::*getURIResolver*();  
 $add\_link = $uriRes->setToURI($uriRes->getOnlyValues($\_SERVER[**'REQUEST\_URI'**]),  
 **'add\_product'**, $id);  
 $add\_div = ($isAdmin) ? **""** : **"<div><a href='**$add\_link**'>Добавить в корзину</a></div>"**;  
 **echo "<div class='page\_pic'><img src='**$picture**'></div>  
 <div class='name\_page'>**$name**</div>  
 <div class='price\_page'>Price:** $price**$</div>  
 <div class='cntrl\_page'>**$add\_div**</div>  
 <div>Описание:<br>**$description**</div>"**;  
 }  
  
 **public function** viewGoods($db, $pattern, $sort\_column, $desc)  
 {  
 $dao = **new** ProductDao($db);  
 $goods = $dao->getAllSearched(**'name'**, $pattern, $sort\_column, $desc);  
 $uriRes = URIResolver::*getURIResolver*();  
 **foreach** ($goods **as** $product)  
 {  
 $id = $product->getId();  
 $name = $product->getName();  
 $price = $product->getPrice();  
 **require "paths.php"**;  
 $picture = $PIC\_PRODUCTS\_PATH  
 . $dao->getColumnBy(**'id'**, $id, **'picture\_path'**)[0];  
 $del\_link = $uriRes->setToURI($uriRes->getOnlyValues($\_SERVER[**'REQUEST\_URI'**]),  
 **'delete\_product'**, $id);  
 **echo "<div class='product'>  
 <div class='product\_pic'><img src='**$picture**'></div>  
 <div class='info\_block'>  
 <div class='name\_block'><a href='goodPage.php?product\_id=**$id**'>**$name**</a></div>  
 <div class='price\_block'>**$price**$</div>  
 <div class='control\_block'>  
 <a href='goodSet.php?product=**$id**'>Изменить</a>  
 <a href='**$del\_link**'>Удалить</a>  
 </div>  
 </div>  
 </div>"**;  
 }  
 **if** (*count*($goods) == 0)  
 {  
 **echo "<p>Товаров с таким названием не найдено</p>"**;  
 }  
 }  
  
 **public function** viewGoodsByCategory($db, $pattern, $sort\_column, $desc, $category\_id)  
 {  
 $count = 0;  
 $dao = **new** ProductDao($db);  
 $goods = $dao->getAllSearched(**'name'**, $pattern, $sort\_column, $desc);  
 $uriRes = URIResolver::*getURIResolver*();  
 **foreach** ($goods **as** $product)  
 {  
 **if** ($product->getCategory()->getId() == $category\_id) {  
 $count++;  
 $id = $product->getId();  
 $name = $product->getName();  
 $price = $product->getPrice();  
 **require "paths.php"**;  
 $picture = $PIC\_PRODUCTS\_PATH  
 . $dao->getColumnBy(**'id'**, $id, **'picture\_path'**)[0];  
 $add\_link = $uriRes->setToURI($uriRes->getOnlyValues($\_SERVER[**'REQUEST\_URI'**]),  
 **'add\_product'**, $id);  
 **echo "<div class='product'><div class='product\_pic'><img src='**$picture**'></div>  
 <div class='info\_block'>  
 <div class='name\_block'><a href='goodPage.php?product\_id=**$id**'>**$name**</a></div>  
 <div class='price\_block'>**$price**$</div>  
 <a href='**$add\_link**'>Добавить в корзину</a>  
 </div>  
 </div>"**;  
 }  
 }  
 **if** ($count == 0)  
 {  
 **echo "<p>Товаров с таким названием не найдено</p>"**;  
 }  
 }  
}

view/userControl.html

<**div class="controls\_right"**>  
 <**div class="registration\_link"**><**a href="userHistory.php"**>История</**a**></**div**>  
 <**div class="autho\_link"**><**a href="model/util/logout.php"**>Выход</**a**></**div**>  
</**div**>

view/UserView.php

**<?php  
  
require\_once "D:\Workspace\UnitTesting\courseproject\model\util\dao\UserDao.php"**;  
**require\_once "D:\Workspace\UnitTesting\courseproject\model\logic\URIResolver.php"**;  
  
**class** UserView  
{  
 **public function** adminView($db, $pattern)  
 {  
 $dao = **new** UserDao($db);  
 $users = $dao->getAllSearched(**'login'**, $pattern, **'login'**);  
 **foreach** ($users **as** $user)  
 {  
 $id = $user->getId();  
 $login = $user->getLogin();  
 $uriRes = URIResolver::*getURIResolver*();  
 $history\_link = $uriRes->setToURI(**'userHistory.php'**, **'user'**, $id);  
 $delete\_link = $uriRes->setToURI(**''**, **'delete\_user'**, $id);  
 **echo "<div class='item'>  
 <div class='item\_txt'>**$login**</div>  
 <div class='item\_control'>  
 <a href=**$history\_link**>История заказов </a><a href=**$delete\_link**>Удалить</a>  
 </div>  
 </div>"**;  
 }  
 **if** (*count*($users) == 0)  
 {  
 **echo "<p>Пользователей с таким логином не найдено</p>"**;  
 }  
 }  
}

addCategories.php

**<?php  
require\_once "model/logic/CategoryVerifyer.php"**;  
**require\_once "model/util/connectDB.php"**;  
**require\_once "model/util/dao/CategoryDao.php"**;  
**require\_once "model/util/awayIfNotAdmin.php"**;  
**require\_once "model/util/utilFunc.php"**;  
**require\_once "view/paths.php"**;  
  
**if** (**isset**($\_POST[**"add\_category"**]))  
{  
  
 $name = $\_POST[**"name"**];  
 $errors\_msg = **array**();  
 $errors\_msg[**"incorrect\_name"**] = **"Некорректное название категории"**;  
 $errors\_msg[**"double\_name"**] = **"Категория с таким названием уже существует"**;  
  
 $verifyer = **new** CategoryVerifyer($errors\_msg);  
 $error = $verifyer->categoryErrors($name, $mysqli);  
  
 **if** (!$error && **isset**($\_FILES[**"upload"**]) && ($\_FILES[**"upload"**][**"type"**] == **"image/jpeg"** || $\_FILES[**"upload"**][**"type"**] == **"image/png"**))  
 {  
 $pic = $\_FILES[**"upload"**][**"name"**];  
 $path = ***\_\_DIR\_\_*** . **"\\"** . $PIC\_CATEGORIES\_PATH . $pic;  
 **if** (!*move\_uploaded\_file*($\_FILES[**"upload"**][**"tmp\_name"**], $path)) {  
 **echo "Изображение не было загружено"**;  
 } **else** {  
 $dao = **new** CategoryDao($mysqli);  
 $dao->add(**new** Category(0, $name));  
 $dao->updateColumnBy(**"name"**, $name, **"picture\_path"**, $pic);  
 gotoPage(**"http://localhost:63342/courseproject/index.php"**);  
 }  
 } **else if** (!$error && **isset**($\_FILES[**"upload"**]) && $\_FILES[**"upload"**][**"name"**] != **""** ) {  
 **echo "Данный тип файла не поддерживается"**;  
 } **else if** (!$error) {  
 $dao = **new** CategoryDao($mysqli);  
 $dao->add(**new** Category(0, $name));  
 gotoPage(**"http://localhost:63342/courseproject/index.php"**);  
 } **else** {  
 **echo** $error . **'</br>'**;  
 }  
  
}  
**?>**<!DOCTYPE **html**>  
<**html**>  
  
<**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Добавление категорий</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/categories.css" rel="stylesheet" type="text/css"**/>  
</**head**>  
  
<**body**>  
 <**form action="" method="POST" class="user\_identify" enctype="multipart/form-data"**>  
 <**h2**>Новая категория</**h2**>  
 <**p**>Название<**br**><**input type="text" name="name" autocomplete="off"**></**p**>  
 <**p**>Изображение<**br**><**input type="file" name="upload" accept="image/jpeg"**></**p**>  
 <**button type="submit" name="add\_category"**>Подтвердить</**button**>  
 </**form**>  
</**body**>  
  
</**html**>

addGoods.php

**<?php  
require\_once "model/util/awayIfNotAdmin.php"**;  
**?>**<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Добавление товара</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/categories.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/admin.css" rel="stylesheet" type="text/css"**/>  
 </**head**>  
  
 <**body**>  
 <**div class="admin\_menu"**>  
 **<?php  
 require\_once "view/CategoryView.php"**;  
 **require\_once "model/util/connectDB.php"**;  
 $view = **new** CategoryView();  
 $view->adminSelectView($mysqli);  
 **?>** </**div**>  
 </**body**>  
  
</**html**>

admin.php

**<?php  
require\_once "model/util/awayIfNotAdmin.php"**;  
**?>**<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Администратор</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/categories.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/admin.css" rel="stylesheet" type="text/css"**/>  
 </**head**>  
  
 <**body**>  
 <**div class="admin\_menu"**>  
 <**h2**>Администратор</**h2**>  
 <**a href="users.php"**>Список пользователей</**a**></**br**>  
 <**a href="orders.php"**>Список заказов</**a**></**br**>  
 <**a href="goods.php"**>Список товаров</**a**></**br**>  
 <**a href="addGoods.php"**>Добавить товар</**a**></**br**>  
 <**a href="addCategories.php"**>Добавить категорию</**a**></**br**>  
 <**a href="categories.php"**>Удалить категорию</**a**></**br**>  
 <**a href="history.php"**>История заказов</**a**></**br**>  
 <**a href="model/util/logout.php"**>Выйти</**a**>  
 </**div**>  
 </**body**>  
  
</**html**>

authentication.php

**<?php  
require\_once "model/util/session.php"**;  
**require\_once "model/util/connectDB.php"**;  
**require\_once "model/util/utilFunc.php"**;  
**require\_once "model/logic/AuthenticationVerifyer.php"**;  
**require\_once "model/logic/Authenticator.php"**;  
  
**if** (**isset**($\_POST[**'do\_login'**])) {  
 $login = $\_POST[**'login'**];  
 $password = $\_POST[**'password'**];  
  
 $errors\_msg = **array**();  
 $errors\_msg[**"incorrect\_login"**] = **"Некорректный логин"**;  
 $errors\_msg[**"incorrect\_password"**] = **"Некорректный пароль"**;  
 $errors\_msg[**"not\_user"**] = **"Неправильный логин или пароль"**;  
  
 $verifyer = **new** AuthenticationVerifyer($errors\_msg);  
 $error = $verifyer->authenticationErrors($login, $password, $mysqli);  
  
 **if** (!$error) {  
 (**new** Authenticator())->authenticate($login, $error);  
 *//$\_SESSION['logged\_user'] = $\_POST['login'];* $page = $\_SESSION[**'logged\_user'**] == **'admin'** ? **"http://localhost:63342/courseproject/admin.php"** : **"http://localhost:63342/courseproject/index.php"**;  
 gotoPage($page);  
 } **else** {  
 **echo** $error . **'</br>'**;  
 }  
}  
**?>**<!DOCTYPE **html**>  
<**html**>  
 <**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Авторизация</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 </**head**>  
 <**body**>  
 <**form action="authentication.php" method="POST" class="user\_identify"**>  
 <**h2**>Вход</**h2**>  
 <**p**>Логин<**br**><**input type="text" name="login" autocomplete="off"**></**p**>  
 <**p**>Пароль<**br**><**input type="password" name="password"**></**p**>  
 <**button type="submit" name="do\_login"**>Войти</**button**>  
 </**form**>  
 </**body**>  
</**html**>

basket.php

**<?php  
require\_once "model/logic/URIResolver.php"**;  
**require\_once "model/util/utilFunc.php"**;  
**require\_once "model/util/dao/CartDao.php"**;  
**require\_once "model/util/session.php"**;  
$uriRes = URIResolver::*getURIResolver*();  
**if** ($uriRes->hasGET(**'delete\_rec'**))  
{  
 $dao = **new** CartDao();  
 $dao->deleteById($uriRes->getValue(**'delete\_rec'**));  
 gotoPage($uriRes->unsetFromURI($\_SERVER[**'REQUEST\_URI'**], **'delete\_rec'**));  
}  
**?>**<!DOCTYPE **html**>  
<**html**>  
  
<**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Корзина</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/categories.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/items.css" rel="stylesheet" type="text/css"**/>  
</**head**>  
  
<**body**>  
  
 <**header**>  
 **<?php  
 require\_once "model/logic/FaceControl.php"**;  
 **if** (!FaceControl::*getFaceControl*()->isAdmin())  
 {  
 **include "view/header.html"**;  
 **include** FaceControl::*getFaceControl*()->getOneOf(**"view/authorizationControl.html"**,  
 **"view/userControl.html"**);  
 }  
 **?>** </**header**>  
  
 <**div class="list"**>  
 **<?php  
 require\_once "model/util/session.php"**;  
 **require\_once "view/CartView.php"**;  
 $view = **new** CartView();  
 $view->view();  
 **?>** </**div**>  
</**body**>  
  
</**html**>

categories.php

**<?php  
include "model/util/awayIfNotAdmin.php"**;  
**if** (**isset**($\_GET[**"delete\_category"**]))  
{  
 **require\_once "model/util/connectDB.php"**;  
 **require\_once "model/util/dao/CategoryDao.php"**;  
 **require\_once "view/paths.php"**;  
 $dao = **new** CategoryDao($mysqli);  
 $path = ***\_\_DIR\_\_*** . **"\\"** . $PIC\_CATEGORIES\_PATH  
 . $dao->getColumnBy(**'id'**, $\_GET[**"delete\_category"**], **"picture\_path"**)[0];  
 *unlink*($path);  
 $dao->deleteBy(**'id'**, $\_GET[**"delete\_category"**]);  
}  
**?>**<!DOCTYPE **html**>  
<**html**>  
  
<**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Категории</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/categories.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/items.css" rel="stylesheet" type="text/css"**/>  
</**head**>  
  
<**body**>  
 <**div class="list"**>  
 **<?php  
 require\_once "view/CategoryView.php"**;  
 **require\_once "model/util/connectDB.php"**;  
 $view = **new** CategoryView();  
 $view->adminView($mysqli);  
 **?>** </**div**>  
</**body**>  
  
</**html**>

createOrder.php

**<?php  
require\_once "model/util/session.php"**;  
**require\_once "model/util/connectDB.php"**;  
**require\_once "model/util/utilFunc.php"**;  
**require\_once "model/util/dao/UserDao.php"**;  
**require\_once "model/util/dao/CartDao.php"**;  
**require\_once "model/util/dao/OrderRecordDao.php"**;  
**require\_once "model/logic/FaceControl.php"**;  
**require\_once "model/logic/OrderVerifyer.php"**;  
**require\_once "model/logic/URIResolver.php"**;  
  
$uriRes = URIResolver::*getURIResolver*();  
**if** ($uriRes->hasPOST(**'order'**)) {  
 $dao = **new** CartDao();  
 $address = $uriRes->getPOSTValue(**'address'**);  
 $phone = $uriRes->getPOSTValue(**'phone'**);  
 $cart = $dao->getAll();  
  
 $errors\_msg = **array**();  
 $errors\_msg[**"incorrect\_address"**] = **"Некорректно введен адрес"**;  
 $errors\_msg[**"incorrect\_phone"**] = **"Некорректный телефон"**;  
  
 $verifyer = **new** OrderVerifyer($errors\_msg);  
 $error = $verifyer->orderErrors($address, $phone);  
  
 **if** (!$error && **isset**($\_SESSION[**"logged\_user"**])) {  
 $login = $\_SESSION[**"logged\_user"**];  
 $userDao = **new** UserDao($mysqli);  
 $user = $userDao->getBy(**'login'**, $login)[0];  
 $orderDao = **new** OrderRecordDao($mysqli);  
 **foreach** ($cart **as** $record) {  
 $product = $record->getProduct();  
 $count = $record->getCount();  
 $orderDao->add(**new** OrderRecord(0, $user, $product,  
 $count, $address, $phone, *date*(**"Y-m-d"**)));  
 $dao->delete($record);  
 }  
 gotoPage(**"http://localhost:63342/courseproject/index.php"**);  
 } **else if** (!$error) {  
 $orderDao = **new** OrderRecordDao($mysqli);  
 **foreach** ($cart **as** $record) {  
 $product = $record->getProduct();  
 $count = $record->getCount();  
 $orderDao->add(**new** OrderRecord(0, **null**, $product,  
 $count, $address, $phone, *date*(**"Y-m-d"**)));  
 $dao->delete($record);  
 }  
 gotoPage(**"http://localhost:63342/courseproject/index.php"**);  
 } **else** {  
 **echo** $error . **'</br>'**;  
 }  
}  
**?>**<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Заказ</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 </**head**>  
  
 <**body**>  
 <**form action="" method="POST" class="user\_identify"**>  
 <**h2**>Оформление заказа</**h2**>  
 <**p**>Адрес<**br**><**input type="text" name="address"**></**p**>  
 <**p**>Телефон<**br**><**input type="text" name="phone"**></**p**>  
 <**button type="submit" name="order"**>Отправить</**button**>  
 </**form**>  
 </**body**>  
  
</**html**>

goodPage.php

**<?php  
require\_once "model/logic/FaceControl.php"**;  
**require\_once "model/logic/URIResolver.php"**;  
**require\_once "model/util/utilFunc.php"**;  
**require\_once "model/util/connectDB.php"**;  
  
$uriRes = URIResolver::*getURIResolver*();  
  
**if** ($uriRes->hasGET(**"add\_product"**))  
{  
 **require\_once "model/util/dao/CartDao.php"**;  
 **require\_once "model/util/dao/ProductDao.php"**;  
 **require\_once "model/util/session.php"**;  
 $dao = **new** CartDao();  
 $id = $uriRes->getValue(**"add\_product"**);  
 **if** ($dao->getBy($id))  
 {  
 $record = $dao->getBy($id);  
 $record->setCount($record->getCount() + 1);  
 $dao->update(**array**($record));  
 }  
 **else** {  
 $daoProd = **new** ProductDao($mysqli);  
 **if** (**isset**($daoProd->getBy(**'id'**, $id)[0])) {  
 $dao->add(**array**(**new** CartRecord($daoProd->getBy(**'id'**, $id)[0], 1)));  
 }  
 }  
 gotoPage($uriRes->unsetFromURI($\_SERVER[**'REQUEST\_URI'**], **'add\_product'**));  
}  
**?>**<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Товар</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/categories.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/products.css" rel="stylesheet" type="text/css"**/>  
 </**head**>  
  
 <**body**>  
 <**header**>  
 **<?php  
 if** (!FaceControl::*getFaceControl*()->isAdmin())  
 {  
 **include "view/header.html"**;  
 **include** FaceControl::*getFaceControl*()->getOneOf(**"view/authorizationControl.html"**,  
 **"view/userControl.html"**);  
 }  
 **?>** </**header**>  
 <**div class="product\_page"**>  
 **<?php  
 require\_once "view/ProductView.php"**;  
  
 $view = **new** ProductView();  
 **if** ($uriRes->hasGET(**"product\_id"**)) {  
 $view->viewPage($mysqli,  
 $uriRes->getValue(**"product\_id"**),  
 FaceControl::*getFaceControl*()->isAdmin());  
 }  
 **?>** </**div**>  
 </**body**>  
  
</**html**>

goods.php

**<?php  
require\_once "model/logic/FaceControl.php"**;  
**require\_once "model/logic/URIResolver.php"**;  
**require\_once "model/util/utilFunc.php"**;  
$uriRes = URIResolver::*getURIResolver*();  
**if** ($uriRes->hasGET(**"delete\_product"**) && FaceControl::*getFaceControl*()->isAdmin())  
{  
 **require\_once "model/util/connectDB.php"**;  
 **require\_once "model/util/dao/ProductDao.php"**;  
 **require\_once "view/paths.php"**;  
 $dao = **new** ProductDao($mysqli);  
 **if** (**isset**($dao->getColumnBy(**'id'**,  
 $uriRes->getValue(**"delete\_product"**), **"picture\_path"**)[0]))  
 {  
 $path = ***\_\_DIR\_\_*** . **"\\"** . $PIC\_PRODUCTS\_PATH  
 . $dao->getColumnBy(**'id'**,  
 $uriRes->getValue(**"delete\_product"**), **"picture\_path"**)[0];  
 *unlink*($path);  
 }  
 $dao->deleteBy(**'id'**, $uriRes->getValue(**"delete\_product"**));  
 gotoPage($uriRes->unsetFromURI($\_SERVER[**'REQUEST\_URI'**], **'delete\_product'**));  
}  
  
**if** ($uriRes->hasGET(**"add\_product"**))  
{  
 **require\_once "model/util/dao/CartDao.php"**;  
 **require\_once "model/util/dao/ProductDao.php"**;  
 **require\_once "model/util/session.php"**;  
 $dao = **new** CartDao();  
 $id = $uriRes->getValue(**"add\_product"**);  
 **if** ($dao->getBy($id))  
 {  
 $record = $dao->getBy($id);  
 $record->setCount($record->getCount() + 1);  
 $dao->update(**array**($record));  
 }  
 **else** {  
 **require\_once "model/util/connectDB.php"**;  
 $daoProd = **new** ProductDao($mysqli);  
 **if** (**isset**($daoProd->getBy(**'id'**, $id)[0])) {  
 $dao->add(**array**(**new** CartRecord($daoProd->getBy(**'id'**, $id)[0], 1)));  
 }  
 }  
 gotoPage($uriRes->unsetFromURI($\_SERVER[**'REQUEST\_URI'**], **'add\_product'**));  
}  
**?>**<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Товары</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/categories.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/products.css" rel="stylesheet" type="text/css"**/>  
 </**head**>  
  
 <**body**>  
 <**header**>  
 **<?php  
 if** (!FaceControl::*getFaceControl*()->isAdmin())  
 {  
 **include "view/header.html"**;  
 **include** FaceControl::*getFaceControl*()->getOneOf(**"view/authorizationControl.html"**,  
 **"view/userControl.html"**);  
 }  
 **?>** </**header**>  
 <**div class="product\_panel"**>  
 <**form class="search"**>  
 <**input type="search" name="product\_name" placeholder="Поиск товара"**>  
 **<?php  
 if** ($uriRes->hasGET(**"category"**))  
 {  
 $val = $uriRes->getValue(**'category'**);  
 **echo "<input type='hidden' name='category' value='**$val**'>"**;  
 }  
 **if** ($uriRes->hasGET(**"sort\_by"**))  
 {  
 $val = $uriRes->getValue(**'sort\_by'**);  
 **echo "<input type='hidden' name='sort\_by' value='**$val**'>"**;  
 }  
 **if** ($uriRes->hasGET(**"desc"**))  
 {  
 $val = $uriRes->getValue(**'desc'**);  
 **echo "<input type='hidden' name='desc' value='**$val**'>"**;  
 }  
 **?>** <**input type="submit" value="найти"**>  
 </**form**>  
 <**a href="  
 <?php** $uri = $uriRes->setToURI($uriRes->getOnlyValues($\_SERVER[**'REQUEST\_URI'**]),  
 **'sort\_by'**, **'name'**);  
 $desc = ($uriRes->hasGET(**'desc'**)) ? !$uriRes->getValue(**'desc'**) : **false**;  
 $uri = $uriRes->setToURI($uri, **'desc'**, $desc);  
 **echo** $uri;  
 **?>"**>По названию</**a**>  
  
 <**a href="  
 <?php** $uri = $uriRes->setToURI($uriRes->getOnlyValues($\_SERVER[**'REQUEST\_URI'**]),  
 **'sort\_by'**, **'price'**);  
 $desc = ($uriRes->hasGET(**'desc'**)) ? !$uriRes->getValue(**'desc'**) : **false**;  
 $uri = $uriRes->setToURI($uri, **'desc'**, $desc);  
 **echo** $uri;  
 **?>"**>По цене</**a**>  
 **<?php  
 require\_once "view/ProductView.php"**;  
 **require\_once "model/util/connectDB.php"**;  
 **require\_once "model/logic/FaceControl.php"**;  
 $view = **new** ProductView();  
 $pattern = (**isset**($\_GET[**"product\_name"**])) ? **"%"** . $\_GET[**"product\_name"**] . **"%"** : **"%"**;  
 $sort\_by = (**isset**($\_GET[**"sort\_by"**])) ? $\_GET[**"sort\_by"**] : **"name"**;  
 $desc = (**isset**($\_GET[**"desc"**])) ? (bool) $\_GET[**"desc"**] : **false**;  
 **if** (FaceControl::*getFaceControl*()->isAdmin())  
 {  
 $view->viewGoods($mysqli, $pattern, $sort\_by, $desc);  
 }  
 **else if** (**isset**($\_GET[**"category"**]))  
 {  
 $view->viewGoodsByCategory($mysqli, $pattern, $sort\_by, $desc, $\_GET[**"category"**]);  
 }  
 **?>** </**div**>  
 </**body**>  
  
</**html**>

goodSet.php

**<?php  
require\_once "model/logic/ProductVerifyer.php"**;  
**require\_once "model/util/connectDB.php"**;  
**require\_once "model/util/dao/ProductDao.php"**;  
**require\_once "model/util/awayIfNotAdmin.php"**;  
**require\_once "model/util/utilFunc.php"**;  
**require\_once "view/paths.php"**;  
  
**if** (**isset**($\_POST[**"set\_product"**]))  
{  
  
 $name = $\_POST[**"name"**];  
 $price = $\_POST[**"price"**];  
 $description = $\_POST[**"description"**];  
  
 $errors\_msg = **array**();  
 $errors\_msg[**"incorrect\_name"**] = **"Некорректное название товара"**;  
 $errors\_msg[**"incorrect\_price"**] = **"Некорректная цена товара"**;  
 $errors\_msg[**"big\_description"**] = **"Слишком большое описание"**;  
 $errors\_msg[**"double\_name"**] = **"Товар с таким названием уже существует"**;  
  
  
 $verifyer = **new** ProductVerifyer($errors\_msg);  
 $error = $verifyer->productErrors($name, $price, $description, $mysqli);  
  
 **if** (!$error && **isset**($\_FILES[**"upload"**]) && ($\_FILES[**"upload"**][**"type"**] == **"image/jpeg"** || $\_FILES[**"upload"**][**"type"**] == **"image/png"**))  
 {  
 $pic = $\_FILES[**"upload"**][**"name"**];  
 $path = ***\_\_DIR\_\_*** . **"\\"** . $PIC\_PRODUCTS\_PATH . $pic;  
 **if** (!*move\_uploaded\_file*($\_FILES[**"upload"**][**"tmp\_name"**], $path)) {  
 **echo "Изображение не было загружено"**;  
 } **else if** (**isset**($\_GET[**"category"**])) {  
 $dao = **new** CategoryDao($mysqli);  
 $category = $dao->getBy(**"id"**, $\_GET[**"category"**])[0];  
 $dao = **new** ProductDao($mysqli);  
 $dao->add(**new** Product(0, $name, $category, $price, $description));  
 $dao->updateColumnBy(**"name"**, $name, **"picture\_path"**, $pic);  
 gotoPage(**"http://localhost:63342/courseproject/index.php"**);  
 } **else** {  
 $id = $\_GET[**"product"**];  
 $dao = **new** ProductDao($mysqli);  
 $product = $dao->getBy(**"id"**, $id)[0];  
 $product->setName($name);  
 $product->setPrice($price);  
 $product->setDescription($description);  
 $dao->update($product);  
 $dao->updateColumnBy(**"name"**, $name, **"picture\_path"**, $pic);  
 gotoPage(**"http://localhost:63342/courseproject/index.php"**);  
 }  
 } **else if** (!$error && **isset**($\_FILES[**"upload"**]) && $\_FILES[**"upload"**][**"name"**] != **""** ) {  
 **echo "Данный тип файла не поддерживается"**;  
 } **else if** (!$error) {  
 **if** (**isset**($\_GET[**"category"**])) {  
 $dao = **new** CategoryDao($mysqli);  
 $category = $dao->getBy(**"id"**, $\_GET[**"category"**])[0];  
 $dao = **new** ProductDao($mysqli);  
 $dao->add(**new** Product(0, $name, $category, $price, $description));  
 } **else** {  
 $id = $\_GET[**"product"**];  
 $dao = **new** ProductDao($mysqli);  
 $product = $dao->getBy(**"id"**, $id);  
 $product->setName($name);  
 $product->setPrice($price);  
 $product->setDescription($description);  
 $dao->update($product);  
 }  
 gotoPage(**"http://localhost:63342/courseproject/index.php"**);  
 } **else** {  
 **echo** $error . **'</br>'**;  
 }  
}  
**?>**<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Добавить категорию</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/categories.css" rel="stylesheet" type="text/css"**/>  
 </**head**>  
  
 <**body**>  
 <**form action="" method="POST" class="user\_identify" enctype="multipart/form-data"**>  
 <**h2**>Товар</**h2**>  
 <**p**>Название товара<**br**><**input type="text" name="name" autocomplete="off"**></**p**>  
 <**p**>Цена товара<**br**><**input type="text" name="price" autocomplete="off"**></**p**>  
 <**p**>Описание<**br**><**input type="text" name="description" autocomplete="off"**></**p**>  
 <**p**>Изображение<**br**><**input type="file" name="upload" accept="image/jpeg"**></**p**>  
 <**button type="submit" name="set\_product"**>Подтвердить</**button**>  
 </**form**>  
 </**body**>  
  
</**html**>

history.php

**<?php  
require\_once "model/util/awayIfNotAdmin.php"**;  
**require\_once "model/util/connectDB.php"**;  
**require\_once "model/util/dao/HistoryRecordDao.php"**;  
**require\_once "model/logic/URIResolver.php"**;  
**require\_once "model/util/session.php"**;  
**require\_once "model/util/utilFunc.php"**;  
$uriRes = URIResolver::*getURIResolver*();  
**if** ($uriRes->hasGET(**'delete\_history'**))  
{  
 $dao = **new** HistoryRecordDao($mysqli);  
 $dao->deleteBy(**'id'**, $uriRes->getValue(**'delete\_history'**));  
 gotoPage($uriRes->clearURI($\_SERVER[**'REQUEST\_URI'**]));  
}  
  
**if** ($uriRes->hasGET(**'clear\_history'**))  
{  
 $dao = **new** HistoryRecordDao($mysqli);  
 $history = $dao->getAll();  
 **foreach** ($history **as** $record)  
 {  
 $dao->delete($record);  
 }  
 gotoPage($uriRes->clearURI($\_SERVER[**'REQUEST\_URI'**]));  
}  
**?>**<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Пользователи</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/categories.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/items.css" rel="stylesheet" type="text/css"**/>  
 </**head**>  
  
 <**body**>  
 <**div class="list"**>  
 **<?php  
 require\_once "view/HistoryView.php"**;  
 $view = **new** HistoryView();  
 $view->viewAll($mysqli);  
 **?>** </**div**>  
 </**body**>  
  
</**html**>

index.php

**<?php require\_once "model/util/session.php"**; **?>**<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Пекарь</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/categories.css" rel="stylesheet" type="text/css"**/>  
 </**head**>  
  
 <**body**>  
 <**header**>  
 **<?php  
 require\_once "model/util/utilFunc.php"**;  
 **require\_once "model/logic/FaceControl.php"**;  
 **if** (FaceControl::*getFaceControl*()->isAdmin()) {  
 gotoPage(**"admin.php"**);  
 } **else** {  
 **include "view/header.html"**;  
 **include** FaceControl::*getFaceControl*()->getOneOf(**"view/authorizationControl.html"**,  
 **"view/userControl.html"**);  
 }  
 **?>** </**header**>  
  
 <**div class="categories\_panel"**>  
 **<?php  
 require\_once "view/CategoryView.php"**;  
 **require\_once "model/util/connectDB.php"**;  
 $view = **new** CategoryView();  
 $view->consumerView($mysqli);  
 **?>** </**div**>  
   
 <**footer**>  
 **<?php include\_once "view/footer.html"**; **?>** </**footer**>  
 </**body**>  
  
</**html**>

orders.php

**<?php  
require\_once "model/util/awayIfNotAdmin.php"**;  
**require\_once "model/logic/URIResolver.php"**;  
**require\_once "model/util/utilFunc.php"**;  
**require\_once "model/util/connectDB.php"**;  
**require\_once "model/util/dao/OrderRecordDao.php"**;  
  
$uriRes = URIResolver::*getURIResolver*();  
**if** ($uriRes->hasGET(**'delete\_order'**))  
{  
 **require\_once "model/util/connectDB.php"**;  
 $dao = **new** OrderRecordDao($mysqli);  
 $dao->deleteBy(**'id'**, $uriRes->getValue(**'delete\_order'**));  
 gotoPage($uriRes->clearURI($\_SERVER[**'REQUEST\_URI'**]));  
}  
  
**if** ($uriRes->hasGET(**"submit\_order"**))  
{  
 $dao = **new** OrderRecordDao($mysqli);  
 $id = $uriRes->getValue(**"submit\_order"**);  
 $dao->updateColumnBy(**'id'**, $id, **'confirm\_date'**, *date*(**'Y-m-d'**));  
}  
**?>**<!DOCTYPE **html**>  
<**html**>  
  
<**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Пользователи</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/categories.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/items.css" rel="stylesheet" type="text/css"**/>  
</**head**>  
  
<**body**>  
 <**div class="list"**>  
 **<?php  
 require\_once "view/OrderView.php"**;  
 $view = **new** OrderView();  
 $view->view($mysqli);  
 **?>** </**div**>  
</**body**>  
  
</**html**>

registration.php

**<?php  
require\_once "model/util/session.php"**;  
**require\_once "model/util/connectDB.php"**;  
**require\_once "model/util/utilFunc.php"**;  
**require\_once "model/logic/RegistrationVerifyer.php"**;  
**require\_once "model/util/dao/UserDao.php"**;  
**require\_once "model/logic/Registrator.php"**;  
  
**if** (**isset**($\_POST[**"do\_registration"**])) {  
 $login = $\_POST[**'login'**];  
 $email = $\_POST[**'email'**];  
 $password = $\_POST[**'password'**];  
 $password2 = $\_POST[**'password2'**];  
  
 $errors\_msg = **array**();  
 $errors\_msg[**"incorrect\_login"**] = **"Некорректно введен логин"**;  
 $errors\_msg[**"incorrect\_password"**] = **"Некорректно введен пароль"**;  
 $errors\_msg[**"incorrect\_email"**] = **"Некорректно введен email"**;  
 $errors\_msg[**"double\_login"**] = **"Данный логин уже занят"**;  
 $errors\_msg[**"double\_email"**] = **"Пользователь с данной почтой уже существует"**;  
 $errors\_msg[**"incorrect\_repeat"**] = **"Неверно подтвержден пароль"**;  
  
  
 $verifyer = **new** RegistrationVerifyer($errors\_msg);  
 $error = $verifyer->registrationErrors($login, $email, $password,  
 $password2, $mysqli);  
   
 **if** (!$error) {  
 (**new** Registrator())->registrateUser($login, $email, $password, $error, $mysqli);  
 gotoPage(**"http://localhost:63342/courseproject/index.php"**);  
 } **else** {  
 **echo** $error . **'</br>'**;  
 }  
}  
**?>**<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Регистрация</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 </**head**>  
  
 <**body**>  
 <**form action="" method="POST" class="user\_identify"**>  
 <**h2**>Регистрация</**h2**>  
 <**p**>Логин<**br**><**input type="text" name="login" autocomplete="off"**></**p**>  
 <**p**>Почта<**br**><**input type="text" name="email" autocomplete="off"**></**p**>  
 <**p**>Пароль<**br**><**input type="password" name="password"**></**p**>  
 <**p**>Подтверждение пароля<**br**><**input type="password" name="password2"**></**p**>  
 <**button type="submit" name="do\_registration"**>Зарегистрироваться</**button**>  
 </**form**>  
 </**body**>  
  
</**html**>

userHistory.php

**<?php  
require\_once "model/util/connectDB.php"**;  
**require\_once "model/util/dao/HistoryRecordDao.php"**;  
**require\_once "model/logic/URIResolver.php"**;  
**require\_once "model/util/session.php"**;  
**require\_once "model/util/utilFunc.php"**;  
$uriRes = URIResolver::*getURIResolver*();  
**if** ($uriRes->hasGET(**'delete\_history'**))  
{  
 $dao = **new** HistoryRecordDao($mysqli);  
 $dao->deleteBy(**'id'**, $uriRes->getValue(**'delete\_history'**));  
 gotoPage($uriRes->clearURI($\_SERVER[**'REQUEST\_URI'**]));  
}  
  
**if** ($uriRes->hasGET(**'clear\_history'**))  
{  
 $dao = **new** HistoryRecordDao($mysqli);  
 $dao->deleteBy(**'user\_id'**, $uriRes->getValue(**'clear\_history'**));  
 gotoPage($uriRes->clearURI($\_SERVER[**'REQUEST\_URI'**]));  
}  
**?>**<!DOCTYPE **html**>  
<**html**>  
  
<**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Пользователи</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/categories.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/items.css" rel="stylesheet" type="text/css"**/>  
</**head**>  
  
<**body**>  
 <**header**>  
 **<?php  
 require\_once "model/logic/FaceControl.php"**;  
 **if** (!FaceControl::*getFaceControl*()->isAdmin())  
 {  
 **include "view/header.html"**;  
 **include** FaceControl::*getFaceControl*()->getOneOf(**"view/authorizationControl.html"**,  
 **"view/userControl.html"**);  
 }  
 **?>** </**header**>  
 <**div class="list"**>  
 **<?php  
 require\_once "view/HistoryView.php"**;  
 $view = **new** HistoryView();  
 **if** ($uriRes->hasGET(**"user"**))  
 {  
 $user\_id = $uriRes->getValue(**"user"**);  
 }  
 **else** {  
 **require\_once "model/util/dao/UserDao.php"**;  
 $userDao = **new** UserDao($mysqli);  
 $user = $userDao->getBy(**'login'**, $\_SESSION[**"logged\_user"**])[0];  
 $user\_id = $user->getId();  
 }  
 $view->viewAllByUser($mysqli, $user\_id);  
 **?>** </**div**>  
</**body**>  
  
</**html**>

users.php

**<?php  
 require\_once "model/util/awayIfNotAdmin.php"**;  
 **require\_once "model/logic/URIResolver.php"**;  
 **require\_once "model/util/utilFunc.php"**;  
 $uriRes = URIResolver::*getURIResolver*();  
 **if** ($uriRes->hasGET(**'delete\_user'**))  
 {  
 **require\_once "model/util/connectDB.php"**;  
 **require\_once "model/util/dao/UserDao.php"**;  
 $dao = **new** UserDao($mysqli);  
 $dao->deleteBy(**'id'**, $uriRes->getValue(**'delete\_user'**));  
 gotoPage($uriRes->clearURI($\_SERVER[**'REQUEST\_URI'**]));  
 }  
 **?>**<!DOCTYPE **html**>  
<**html**>  
  
 <**head**>  
 <**meta charset="UTF-8"**/>  
 <**title**>Пользователи</**title**>  
 <**link rel="shortcut icon" href="view/pictures/main.ico" type="image/x-icon"**>  
 <**link href="view/css/style.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/categories.css" rel="stylesheet" type="text/css"**/>  
 <**link href="view/css/items.css" rel="stylesheet" type="text/css"**/>  
 </**head**>  
  
 <**body**>  
 <**div class="list"**>  
 <**form class="search"**>  
 <**input type="search" name="user\_login" placeholder="Поиск пользователя"**>  
 <**input type="submit" value="найти"**>  
 </**form**>  
 **<?php  
 require\_once "view/UserView.php"**;  
 **require\_once "model/util/connectDB.php"**;  
 $view = **new** UserView();  
 $pattern = ($uriRes->hasGET(**'user\_login'**)) ? **"%"** . $uriRes->getValue(**'user\_login'**) . **"%"** : **"%"**;*/\*(isset($\_GET["user\_login"])) ? "%" . $\_GET["user\_login"] . "%" : "%";\*/* $view->adminView($mysqli, $pattern);  
 **?>** </**div**>   
 </**body**>  
  
</**html**>

# ПРИЛОЖЕНИЕ Б

## Листинг автоматизированных скриптов

CartDaoTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\util\dao\CartDao.php'**;  
  
**class** CartDaoTest **extends** \Codeception\Test\Unit  
{  
 **private $cartDao**;  
  
 **protected function** setUp()  
 {  
 $this->**cartDao** = **new** CartDao();  
 $\_SESSION = **array**();  
 }  
  
 **public function** testAddNotArrayExpectInvalidArgumentException()  
 {  
 $notArray = **"test"**;  
 $this->expectException(InvalidArgumentException::**class**);  
 $this->**cartDao**->add($notArray);  
 }  
  
 **public function** testAddEmptyArrayExpectEmptySession()  
 {  
 $emptyArray = **array**();  
 $expected = 0;  
 $this->**cartDao**->add($emptyArray);  
 $actual = *count*($\_SESSION);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testAddArrayWithoutCartRecordsExpectEmptySession()  
 {  
 $arrayWithoutCartRecords = **array**(225, **"test"**, 1.337);  
 $expected = 0;  
 $this->**cartDao**->add($arrayWithoutCartRecords);  
 $actual = *count*($\_SESSION);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testAddArrayWithCartRecordsExpectCartRecordSessions()  
 {  
 $cart = **array**();  
 $product1\_id = 12;  
 $product2\_id = 16;  
 $product1\_count = 5;  
 $product2\_count = 2;  
 $cart[] = **new** CartRecord(**new** Product($product1\_id, **"i5 3570"**,  
 **new** Category(11, **"CPU"**), 13, **"3-th gen"**), $product1\_count);  
 $cart[] = **new** CartRecord(**new** Product($product2\_id, **"8 x 2 GB Fury"**,  
 **new** Category(51, **"RAM"**), 10, **"fury"**), $product2\_count);  
 $expected = **array**(**"product\_id:"** . $product1\_id => $product1\_count,  
 **"product\_id:"** . $product2\_id => $product2\_count);  
 $this->**cartDao**->add($cart);  
 $actual = $\_SESSION;  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testDeleteNotCartRecordExpectNoChanges()  
 {  
 $\_SESSION = **array**(**"product\_id:23"** => 12,  
 **"product\_id:14"** => 11);  
 $notCartRecord = **"test"**;  
 $expected = $\_SESSION;  
 $this->**cartDao**->delete($notCartRecord);  
 $actual = $\_SESSION;  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testDeleteCartRecordExpectSessionWithoutThisCartRecord()  
 {  
 $product\_id = 23;  
 $product\_count = 12;  
 $\_SESSION = **array**(**"product\_id:"** . $product\_id => $product\_count,  
 **"product\_id:14"** => 11);  
 $cartRecord = **new** CartRecord(**new** Product($product\_id, **"i5 3570"**,  
 **new** Category(11, **"CPU"**), 13, **"3-th gen"**), $product\_count);  
 $expected = **array**(**"product\_id:14"** => 11);  
 $this->**cartDao**->delete($cartRecord);  
 $actual = $\_SESSION;  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testDeleteByIdRecordNotExistExpectNoChanges()  
 {  
 $\_SESSION = **array**(**"product\_id:12"** => 12, **"product\_id:17"** => 2);  
 $notExistId = 19;  
 $expected = $\_SESSION;  
 $this->**cartDao**->deleteById($notExistId);  
 $actual = $\_SESSION;  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testDeleteByIdCartRecordIdExpectSessionWithoutThisCartRecord()  
 {  
 $id = 12;  
 $\_SESSION = **array**(**"product\_id:"** . $id => 4, **"product\_id:17"** => 2);  
 $expected = **array**(**"product\_id:17"** => 2);  
 $this->**cartDao**->deleteById($id);  
 $actual = $\_SESSION;  
 $this->assertEquals($expected, $actual);  
 }  
}

CategoryDaoTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\util\dao\CategoryDao.php'**;  
  
**class** CategoryDaoTest **extends** \Codeception\Test\Unit  
{  
 **private $expectedParam**;  
  
 **private function** createMysqliQuerySpy($expectedParam)  
 {  
 $this->**expectedParam** = $expectedParam;  
 **return** $this->make(mysqli::**class**,  
 [**"query"** => **function**($actualParam)  
 { $this->assertEquals($this->**expectedParam**, $actualParam); }]);  
 }  
  
 **public function** testAddNotCategoryExpectInvalidArgumentException()  
 {  
 $dummy = $this->make(mysqli::**class**);  
 $categoryDao = **new** CategoryDao($dummy);  
 $notCategoryObject = **"test"**;  
 $this->expectException(InvalidArgumentException::**class**);  
 $categoryDao->add($notCategoryObject);  
 }  
  
 **public function** testAddCategoryExpectQuery()  
 {  
 $name = **"Motherboards"**;  
 $category = **new** Category(12, $name);  
 $table\_name = **'categories'**;  
 $query = **"INSERT INTO `**$table\_name**` (`name`)   
 VALUES ('**$name**')"**;  
 $mysqliSpy = $this->createMysqliQuerySpy($query);  
 $categoryDao = **new** CategoryDao($mysqliSpy);  
 $categoryDao->add($category);  
 }  
  
 **public function** testUpdateNotCategoryExpectInvalidArgumentException()  
 {  
 $dummy = $this->make(mysqli::**class**);  
 $categoryDao = **new** CategoryDao($dummy);  
 $notCategoryObject = **"test"**;  
 $this->expectException(InvalidArgumentException::**class**);  
 $categoryDao->update($notCategoryObject);  
 }  
  
 **public function** testUpdateCategoryExpectQuery()  
 {  
 $id = 123;  
 $name = **"Motherboards"**;  
 $category = **new** Category($id, $name);  
 $table\_name = **'categories'**;  
 $query = **"UPDATE `**$table\_name**` SET `name` = '**$name**' WHERE `id` = '**$id**'"**;  
 $mysqliSpy = $this->createMysqliQuerySpy($query);  
 $categoryDao = **new** CategoryDao($mysqliSpy);  
 $categoryDao->update($category);  
 }  
}

DBDaoTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\util\dao\DBDao.php'**;  
  
**class** DBDaoTest **extends** \Codeception\Test\Unit  
{  
 **private $iterate**;  
 **private $obj**;  
 **private $queryStr**;  
 **private $mysqliResultStub**;  
 **private $mysqliStub**;  
 **private $dbDaoStub**;  
  
 **public function** testGetAllFromEmptyTableExpectEmptyArray()  
 {  
 $this->generateMysqliResultStub();  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `name`"**);  
 $this->generateDBDaoStub();  
 $expected = **array**();  
 $actual = $this->**dbDaoStub**->getAll();  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetAllFromNonEmptyTableExpectArrayWithObject()  
 {  
 $object = **array**(2, **"123"**, 14.8);  
 $this->generateMysqliResultStub($object);  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `name`"**);  
 $this->generateDBDaoStub();  
 $expected = **array**($object); *//$object!* $actual = $this->**dbDaoStub**->getAll();  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetAllSearchedFromEmptyTableWithoutSortColumnExpectEmptyArray()  
 {  
 $pattern = **"%"**;  
 $column = **"column"**;  
 $this->generateMysqliResultStub();  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `name` WHERE `**$column**` LIKE '**$pattern**'"**);  
 $this->generateDBDaoStub();  
 $expected = **array**();  
 $actual = $this->**dbDaoStub**->getAllSearched($column, $pattern);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetAllSearchedFromEmptyTableWithSortColumnAndDescExpectEmptyArray()  
 {  
 $pattern = **"%"**;  
 $column = **"column"**;  
 $sort\_column = **"sort\_column"**;  
 $this->generateMysqliResultStub();  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `name` WHERE `**$column**` LIKE '**$pattern**'  
 ORDER BY `**$sort\_column**` DESC"**);  
 $this->generateDBDaoStub();  
 $expected = **array**();  
 $actual = $this->**dbDaoStub**->getAllSearched(  
 $column, $pattern, $sort\_column, **true**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetAllSearchedFromEmptyTableWithSortColumnExpectEmptyArray()  
 {  
 $pattern = **"%"**;  
 $column = **"column"**;  
 $sort\_column = **"sort\_column"**;  
 $this->generateMysqliResultStub();  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `name` WHERE `**$column**` LIKE '**$pattern**'  
 ORDER BY `**$sort\_column**`"**);  
 $this->generateDBDaoStub();  
 $expected = **array**();  
 $actual = $this->**dbDaoStub**->getAllSearched(  
 $column, $pattern, $sort\_column);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetAllSearchedFromNonEmptyTableWithoutSortColumnExpectArrayWithObject()  
 {  
 $pattern = **"%"**;  
 $column = **"column"**;  
 $object = **"test"**;  
 $this->generateMysqliResultStub($object);  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `name` WHERE `**$column**` LIKE '**$pattern**'"**);  
 $this->generateDBDaoStub();  
 $expected = **array**($object);  
 $actual = $this->**dbDaoStub**->getAllSearched($column, $pattern);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetAllSearchedFromNonEmptyTableWithSortColumnAndDescExpectArrayWithObject()  
 {  
 $pattern = **"%"**;  
 $column = **"column"**;  
 $sort\_column = **"sort\_column"**;  
 $object = **"test"**;  
 $this->generateMysqliResultStub($object);  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `name` WHERE `**$column**` LIKE '**$pattern**'  
 ORDER BY `**$sort\_column**` DESC"**);  
 $this->generateDBDaoStub();  
 $expected = **array**($object);  
 $actual = $this->**dbDaoStub**->getAllSearched(  
 $column, $pattern, $sort\_column, **true**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetAllSearchedFromNonEmptyTableWithSortColumnExpectArrayWithObject()  
 {  
 $pattern = **"%"**;  
 $column = **"column"**;  
 $sort\_column = **"sort\_column"**;  
 $object = **"test"**;  
 $this->generateMysqliResultStub($object);  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `name` WHERE `**$column**` LIKE '**$pattern**'  
 ORDER BY `**$sort\_column**`"**);  
 $this->generateDBDaoStub();  
 $expected = **array**($object);  
 $actual = $this->**dbDaoStub**->getAllSearched(  
 $column, $pattern, $sort\_column);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetByFromEmptyTableExpectEmptyArray()  
 {  
 $column\_name = **"column\_name"**;  
 $value = **"value"**;  
 $this->generateMysqliResultStub();  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `name` WHERE `**$column\_name**` = '**$value**'"**);  
 $this->generateDBDaoStub();  
 $expected = **array**();  
 $actual = $this->**dbDaoStub**->getBy($column\_name, $value);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetByFromNonEmptyTableExpectArrayWithObject()  
 {  
 $column\_name = **"column\_name"**;  
 $value = **"value"**;  
 $object = **"test"**;  
 $this->generateMysqliResultStub($object);  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `name` WHERE `**$column\_name**` = '**$value**'"**);  
 $this->generateDBDaoStub();  
 $expected = **array**($object);  
 $actual = $this->**dbDaoStub**->getBy($column\_name, $value);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetColumnByFromEmptyTableExpectEmptyArray()  
 {  
 $get\_column = **"getcolumn"**;  
 $column = **"column"**;  
 $value = **"value"**;  
 $this->generateMysqliResultStub();  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT `**$get\_column**` FROM `name` WHERE `**$column**` = '**$value**'"**);  
 $this->generateDBDaoStub();  
 $expected = **array**();  
 $actual = $this->**dbDaoStub**->getColumnBy($column, $value, $get\_column);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetColumnByFromNonEmptyTableExpectArrayWithObject()  
 {  
 $get\_column = **"getcolumn"**;  
 $column = **"column"**;  
 $value = **"value"**;  
 $object = **array**( $get\_column => **"test"**);  
 $this->generateMysqliResultStub($object);  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT `**$get\_column**` FROM `name` WHERE `**$column**` = '**$value**'"**);  
 $this->generateDBDaoStub();  
 $expected = **array**(**"test"**);  
 $actual = $this->**dbDaoStub**->getColumnBy($column, $value, $get\_column);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **private function** generateMysqliResultStub($obj=**false**)  
 {  
 $this->**iterate** = 1;  
 $this->**obj** = $obj;  
 $this->**mysqliResultStub** = $this->make(mysqli\_result::**class**,  
 [**"fetch\_assoc"** => **function**()  
 {  
 **if** ($this->**iterate** == 1) {  
 $this->**iterate** = 2;  
 **return** $this->**obj**;  
 }  
 **return false**;  
 }]);  
 }  
  
 **private function** generateMysqliStub($resultStub, $queryStr)  
 {  
 $this->**mysqliResultStub** = $resultStub;  
 $this->**queryStr** = $queryStr;  
 $this->**mysqliStub** = $this->make(mysqli::**class**,  
 [**"query"** => **function**($queryS)  
 {  
 **if** ($this->**queryStr** == $queryS)  
 {  
 **return** $this->**mysqliResultStub**;  
 }  
 **return null**;  
 }]);  
 }  
  
 **private function** generateDBDaoStub()  
 {  
 $this->**dbDaoStub** = $this->construct(DBDao::**class**,  
 [$this->**mysqliStub**, **"table\_name"** => **"name"**], [  
 **"convert"** => **function**($obj) {**return** $obj;},  
 **"add"** => **null**,  
 **"update"** => **null**,  
 **"delete"** => **null**]);  
 }  
}

IdentificationalDaoTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\util\dao\IdentificationalDao.php'**;  
  
**class** IdentificationalDaoTest **extends** \Codeception\Test\Unit  
{  
 **private $expectedParam**;  
  
 **public function** testDeleteNotIdentificationalExpectInvalidArgumentException()  
 {  
 $spy = $this->createMysqliQuerySpy(**null**);  
 $dao = $this->createIdentificationalDaoStub($spy);  
 $this->expectException(InvalidArgumentException::**class**);  
 $dao->delete(**"test"**);  
 }  
  
 **public function** testDeleteIdentificationalExpectAction()  
 {  
 $id = 14;  
 $identificational = $this->make(Identificational::**class**, [**"getId"** => $id]);  
 $spy = $this->createMysqliQuerySpy(**"DELETE FROM `name` WHERE `id` = '**$id**'"**);  
 $dao = $this->createIdentificationalDaoStub($spy);  
 $dao->delete($identificational);  
 }  
  
 **private function** createMysqliQuerySpy($expectedParam)  
 {  
 $this->**expectedParam** = $expectedParam;  
 **return** $this->make(mysqli::**class**,  
 [**"query"** => **function**($actualParam)  
 { $this->assertEquals($this->**expectedParam**, $actualParam); }]);  
 }  
  
 **private function** createIdentificationalDaoStub($mysqliSpy)  
 {  
 **return** $this->construct(IdentificationalDao::**class**,  
 [$mysqliSpy, **"table\_name"** => **"name"**], [  
 **"convert"** => **function**($obj) {**return** $obj;},  
 **"add"** => **null**,  
 **"update"** => **null**]);  
 }  
}

HistoryRecordDaoTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\util\dao\HistoryRecordDao.php'**;  
  
**class** HistoryRecordDaoTest **extends** \Codeception\Test\Unit  
{  
 **private $iterate**;  
 **private $obj**;  
 **private $queryStr**;  
 **private $mysqliResultStub**;  
 **private $mysqliStub**;  
 **private $expectedParam**;  
  
 **public function** testAddNotHistoryRecordExpectInvalidArgumentException()  
 {  
 $object = **"test"**;  
 $db = $this->createMysqliQuerySpy(**null**);  
 $dao = **new** HistoryRecordDao($db);  
 $this->expectException(InvalidArgumentException::**class**);  
 $dao->add($object);  
 }  
  
 **public function** testAddHistoryRecordExpectAction()  
 {  
 $user\_id = 14;  
 $product\_id = 16;  
 $count = 2;  
 $address = **"Htlr street, 14/oooo"**;  
 $phone = **"1234567"**;  
 $order\_date = **'2018-01-01'**;  
 $confirm\_date = **'2018-01-02'**;  
 $object = **new** HistoryRecord(12, **new** User($user\_id, **"aaaaaaa"**,  
 **"aswer@test.se"**, **"12345678"**), **new** Product($product\_id,  
 **"namename"**, **new** Category(123, **"lalala"**), 143,  
 **"lalala"**), $count,  
 $address, $phone, $order\_date, $confirm\_date);  
 $db = $this->createMysqliQuerySpy(**"INSERT INTO `log` (`user\_id`, `product\_id`, `count`, `address`,  
 `phone`, `order\_date`, `confirm\_date`)   
 VALUES ('**$user\_id**', '**$product\_id**', '**$count**', '**$address**', '**$phone**', '**$order\_date**', '**$confirm\_date**')"**);  
 $dao = **new** HistoryRecordDao($db);  
 $dao->add($object);  
 }  
  
 **public function** testUpdateNotHistoryRecordExpectInvalidArgumentException()  
 {  
 $object = **"test"**;  
 $db = $this->createMysqliQuerySpy(**null**);  
 $dao = **new** HistoryRecordDao($db);  
 $this->expectException(InvalidArgumentException::**class**);  
 $dao->update($object);  
 }  
  
 **public function** testUpdateHistoryRecordExpectAction()  
 {  
 $id = 12;  
 $user\_id = 14;  
 $product\_id = 16;  
 $count = 2;  
 $address = **"Htlr street, 14/oooo"**;  
 $phone = **"1234567"**;  
 $order\_date = **'2018-01-01'**;  
 $confirm\_date = **'2018-01-02'**;  
 $object = **new** HistoryRecord($id, **new** User($user\_id, **"aaaaaaa"**,  
 **"aswer@test.se"**, **"12345678"**), **new** Product($product\_id,  
 **"namename"**, **new** Category(123, **"lalala"**), 143,  
 **"lalala"**), $count,  
 $address, $phone, $order\_date, $confirm\_date);  
 $db = $this->createMysqliQuerySpy(**"UPDATE `log` SET `user\_id` = '**$user\_id**', `product\_id` = '**$product\_id**',  
 `count` = '**$count**', `address` = '**$address**', `phone` = '**$phone**',   
 `order\_date` = '**$order\_date**', `confirm\_date` = '**$confirm\_date**' WHERE `id` = '**$id**'"**);  
 $dao = **new** HistoryRecordDao($db);  
 $dao->update($object);  
 }  
  
 **public function** testGetAllFromEmptyTableExpectEmptyArray()  
 {  
 $this->generateMysqliResultStub();  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `log` WHERE `confirm\_date` IS NOT NULL"**);  
 $dao = **new** HistoryRecordDao($this->**mysqliStub**);  
 $expected = **array**();  
 $actual = $dao->getAll();  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetAllFromNonEmptyTableExpectArrayWithObject()  
 {  
 $id = 12;  
 $user\_id = 14;  
 $product\_id = 16;  
 $count = 2;  
 $address = **"Htlr street, 14/oooo"**;  
 $phone = **"1234567"**;  
 $order\_date = **'2018-01-01'**;  
 $confirm\_date = **'2018-01-02'**;  
 $object = **new** HistoryRecord($id, **new** User($user\_id, **"aaaaaaa"**,  
 **"aswer@test.se"**, **"12345678"**), **new** Product($product\_id,  
 **"namename"**, **new** Category(123, **"lalala"**), 143,  
 **"lalala"**), $count,  
 $address, $phone, $order\_date, $confirm\_date);  
 $this->generateMysqliResultStub($object);  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `log` WHERE `confirm\_date` IS NOT NULL"**);  
 $dao = $this->construct(HistoryRecordDao::**class**, [$this->**mysqliStub**],  
 [**"convert"** => **function**($obj) { **return** $obj; }]);  
 $expected = **array**($object);  
 $actual = $dao->getAll();  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetByFromEmptyTableExpectEmptyArray()  
 {  
 $column\_name = **"column\_name"**;  
 $value = **"val"**;  
 $this->generateMysqliResultStub();  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `log` WHERE `**$column\_name**` = '**$value**'   
 AND `confirm\_date` IS NOT NULL"**);  
 $dao = **new** HistoryRecordDao($this->**mysqliStub**);  
 $expected = **array**();  
 $actual = $dao->getBy($column\_name, $value);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetByFromNonEmptyTableExpectArrayWithObject()  
 {  
 $column\_name = **"column\_name"**;  
 $value = **"val"**;  
 $id = 12;  
 $user\_id = 14;  
 $product\_id = 16;  
 $count = 2;  
 $address = **"Htlr street, 14/oooo"**;  
 $phone = **"1234567"**;  
 $order\_date = **'2018-01-01'**;  
 $confirm\_date = **'2018-01-02'**;  
 $object = **new** HistoryRecord($id, **new** User($user\_id, **"aaaaaaa"**,  
 **"aswer@test.se"**, **"12345678"**), **new** Product($product\_id,  
 **"namename"**, **new** Category(123, **"lalala"**), 143,  
 **"lalala"**), $count,  
 $address, $phone, $order\_date, $confirm\_date);  
 $this->generateMysqliResultStub($object);  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `log` WHERE `**$column\_name**` = '**$value**'   
 AND `confirm\_date` IS NOT NULL"**);  
 $dao = $this->construct(HistoryRecordDao::**class**, [$this->**mysqliStub**],  
 [**"convert"** => **function**($obj) { **return** $obj; }]);  
 $expected = **array**($object);  
 $actual = $dao->getBy($column\_name, $value);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetColumnByFromEmptyTableExpectEmptyArray()  
 {  
 $get\_column = **"get\_column"**;  
 $column = **"column"**;  
 $value = **"val"**;  
 $this->generateMysqliResultStub();  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT `**$get\_column**` FROM `log` WHERE `**$column**` = '**$value**'  
 AND `confirm\_date` IS NOT NULL"**);  
 $dao = **new** HistoryRecordDao($this->**mysqliStub**);  
 $expected = **array**();  
 $actual = $dao->getColumnBy($column, $value, $get\_column);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetColumnByFromNonEmptyTableExpectArrayWithObject()  
 {  
 $get\_column = **"get\_column"**;  
 $column = **"column"**;  
 $value = **"val"**;  
 $object = **array**($get\_column => **"test"**);  
 $this->generateMysqliResultStub($object);  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT `**$get\_column**` FROM `log` WHERE `**$column**` = '**$value**'  
 AND `confirm\_date` IS NOT NULL"**);  
 $dao = $this->construct(HistoryRecordDao::**class**, [$this->**mysqliStub**],  
 [**"convert"** => **function**($obj) { **return** $obj; }]);  
 $expected = **array**(**"test"**);  
 $actual = $dao->getColumnBy($column, $value, $get\_column);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **private function** generateMysqliResultStub($obj=**false**)  
 {  
 $this->**iterate** = 1;  
 $this->**obj** = $obj;  
 $this->**mysqliResultStub** = $this->make(mysqli\_result::**class**,  
 [**"fetch\_assoc"** => **function**()  
 {  
 **if** ($this->**iterate** == 1) {  
 $this->**iterate** = 2;  
 **return** $this->**obj**;  
 }  
 **return false**;  
 }]);  
 }  
  
 **private function** generateMysqliStub($resultStub, $queryStr)  
 {  
 $this->**mysqliResultStub** = $resultStub;  
 $this->**queryStr** = $queryStr;  
 $this->**mysqliStub** = $this->make(mysqli::**class**,  
 [**"query"** => **function**($queryS)  
 {  
 **if** ($this->**queryStr** == $queryS)  
 {  
 **return** $this->**mysqliResultStub**;  
 }  
 **return null**;  
 }]);  
 }  
  
 **private function** createMysqliQuerySpy($expectedParam)  
 {  
 $this->**expectedParam** = $expectedParam;  
 **return** $this->make(mysqli::**class**,  
 [**"query"** => **function**($actualParam)  
 { $this->assertEquals($this->**expectedParam**, $actualParam); }]);  
 }  
}

OrderRecordDaoTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\util\dao\OrderRecordDao.php'**;  
  
**class** OrderRecordDaoTest **extends** \Codeception\Test\Unit  
{  
 **private $iterate**;  
 **private $obj**;  
 **private $queryStr**;  
 **private $mysqliResultStub**;  
 **private $mysqliStub**;  
 **private $expectedParam**;  
  
 **public function** testAddNotOrderRecordExpectInvalidArgumentException()  
 {  
 $object = **"test"**;  
 $db = $this->createMysqliQuerySpy(**null**);  
 $dao = **new** OrderRecordDao($db);  
 $this->expectException(InvalidArgumentException::**class**);  
 $dao->add($object);  
 }  
  
 **public function** testAddOrderRecordExpectAction()  
 {  
 $user\_id = 14;  
 $product\_id = 16;  
 $count = 2;  
 $address = **"Htlr street, 14/oooo"**;  
 $phone = **"1234567"**;  
 $order\_date = **'2018-01-01'**;  
 $object = **new** OrderRecord(12, **new** User($user\_id, **"aaaaaaa"**,  
 **"aswer@test.se"**, **"12345678"**), **new** Product($product\_id,  
 **"namename"**, **new** Category(123, **"lalala"**), 143,  
 **"lalala"**), $count,  
 $address, $phone, $order\_date);  
 $db = $this->createMysqliQuerySpy(**"INSERT INTO `log` (`user\_id`, `product\_id`, `count`, `address`,  
 `phone`, `order\_date`)   
 VALUES ('**$user\_id**', '**$product\_id**', '**$count**', '**$address**', '**$phone**', '**$order\_date**')"**);  
 $dao = **new** OrderRecordDao($db);  
 $dao->add($object);  
 }  
  
 **public function** testUpdateNotOrderRecordExpectInvalidArgumentException()  
 {  
 $object = **"test"**;  
 $db = $this->createMysqliQuerySpy(**null**);  
 $dao = **new** OrderRecordDao($db);  
 $this->expectException(InvalidArgumentException::**class**);  
 $dao->update($object);  
 }  
  
 **public function** testUpdateOrderRecordExpectAction()  
 {  
 $id = 12;  
 $user\_id = 14;  
 $product\_id = 16;  
 $count = 2;  
 $address = **"Htlr street, 14/oooo"**;  
 $phone = **"1234567"**;  
 $order\_date = **'2018-01-01'**;  
 $object = **new** OrderRecord($id, **new** User($user\_id, **"aaaaaaa"**,  
 **"aswer@test.se"**, **"12345678"**), **new** Product($product\_id,  
 **"namename"**, **new** Category(123, **"lalala"**), 143,  
 **"lalala"**), $count,  
 $address, $phone, $order\_date);  
 $db = $this->createMysqliQuerySpy(**"UPDATE `log` SET `user\_id` = '**$user\_id**', `product\_id` = '**$product\_id**',  
 `count` = '**$count**', `address` = '**$address**', `phone` = '**$phone**',   
 `order\_date` = '**$order\_date**' WHERE `id` = '**$id**'"**);  
 $dao = **new** OrderRecordDao($db);  
 $dao->update($object);  
 }  
  
 **public function** testGetAllFromEmptyTableExpectEmptyArray()  
 {  
 $this->generateMysqliResultStub();  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `log` WHERE `confirm\_date` IS NULL"**);  
 $dao = **new** OrderRecordDao($this->**mysqliStub**);  
 $expected = **array**();  
 $actual = $dao->getAll();  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetAllFromNonEmptyTableExpectArrayWithObject()  
 {  
 $id = 12;  
 $user\_id = 14;  
 $product\_id = 16;  
 $count = 2;  
 $address = **"Htlr street, 14/oooo"**;  
 $phone = **"1234567"**;  
 $order\_date = **'2018-01-01'**;  
 $object = **new** OrderRecord($id, **new** User($user\_id, **"aaaaaaa"**,  
 **"aswer@test.se"**, **"12345678"**), **new** Product($product\_id,  
 **"namename"**, **new** Category(123, **"lalala"**), 143,  
 **"lalala"**), $count,  
 $address, $phone, $order\_date);  
 $this->generateMysqliResultStub($object);  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `log` WHERE `confirm\_date` IS NULL"**);  
 $dao = $this->construct(OrderRecordDao::**class**, [$this->**mysqliStub**],  
 [**"convert"** => **function**($obj) { **return** $obj; }]);  
 $expected = **array**($object);  
 $actual = $dao->getAll();  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetByFromEmptyTableExpectEmptyArray()  
 {  
 $column\_name = **"column\_name"**;  
 $value = **"val"**;  
 $this->generateMysqliResultStub();  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `log` WHERE `**$column\_name**` = '**$value**'   
 AND `confirm\_date` IS NULL"**);  
 $dao = **new** OrderRecordDao($this->**mysqliStub**);  
 $expected = **array**();  
 $actual = $dao->getBy($column\_name, $value);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetByFromNonEmptyTableExpectArrayWithObject()  
 {  
 $column\_name = **"column\_name"**;  
 $value = **"val"**;  
 $id = 12;  
 $user\_id = 14;  
 $product\_id = 16;  
 $count = 2;  
 $address = **"Htlr street, 14/oooo"**;  
 $phone = **"1234567"**;  
 $order\_date = **'2018-01-01'**;  
 $object = **new** OrderRecord($id, **new** User($user\_id, **"aaaaaaa"**,  
 **"aswer@test.se"**, **"12345678"**), **new** Product($product\_id,  
 **"namename"**, **new** Category(123, **"lalala"**), 143,  
 **"lalala"**), $count,  
 $address, $phone, $order\_date);  
 $this->generateMysqliResultStub($object);  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT** *\** **FROM `log` WHERE `**$column\_name**` = '**$value**'   
 AND `confirm\_date` IS NULL"**);  
 $dao = $this->construct(OrderRecordDao::**class**, [$this->**mysqliStub**],  
 [**"convert"** => **function**($obj) { **return** $obj; }]);  
 $expected = **array**($object);  
 $actual = $dao->getBy($column\_name, $value);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetColumnByFromEmptyTableExpectEmptyArray()  
 {  
 $get\_column = **"get\_column"**;  
 $column = **"column"**;  
 $value = **"val"**;  
 $this->generateMysqliResultStub();  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT `**$get\_column**` FROM `log` WHERE `**$column**` = '**$value**'  
 AND `confirm\_date` IS NULL"**);  
 $dao = **new** OrderRecordDao($this->**mysqliStub**);  
 $expected = **array**();  
 $actual = $dao->getColumnBy($column, $value, $get\_column);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetColumnByFromNonEmptyTableExpectArrayWithObject()  
 {  
 $get\_column = **"get\_column"**;  
 $column = **"column"**;  
 $value = **"val"**;  
 $object = **array**($get\_column => **"test"**);  
 $this->generateMysqliResultStub($object);  
 $this->generateMysqliStub($this->**mysqliResultStub**,  
 **"SELECT `**$get\_column**` FROM `log` WHERE `**$column**` = '**$value**'  
 AND `confirm\_date` IS NULL"**);  
 $dao = $this->construct(OrderRecordDao::**class**, [$this->**mysqliStub**],  
 [**"convert"** => **function**($obj) { **return** $obj; }]);  
 $expected = **array**(**"test"**);  
 $actual = $dao->getColumnBy($column, $value, $get\_column);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **private function** generateMysqliResultStub($obj=**false**)  
 {  
 $this->**iterate** = 1;  
 $this->**obj** = $obj;  
 $this->**mysqliResultStub** = $this->make(mysqli\_result::**class**,  
 [**"fetch\_assoc"** => **function**()  
 {  
 **if** ($this->**iterate** == 1) {  
 $this->**iterate** = 2;  
 **return** $this->**obj**;  
 }  
 **return false**;  
 }]);  
 }  
  
 **private function** generateMysqliStub($resultStub, $queryStr)  
 {  
 $this->**mysqliResultStub** = $resultStub;  
 $this->**queryStr** = $queryStr;  
 $this->**mysqliStub** = $this->make(mysqli::**class**,  
 [**"query"** => **function**($queryS)  
 {  
 **if** ($this->**queryStr** == $queryS)  
 {  
 **return** $this->**mysqliResultStub**;  
 }  
 **return null**;  
 }]);  
 }  
  
 **private function** createMysqliQuerySpy($expectedParam)  
 {  
 $this->**expectedParam** = $expectedParam;  
 **return** $this->make(mysqli::**class**,  
 [**"query"** => **function**($actualParam)  
 { $this->assertEquals($this->**expectedParam**, $actualParam); }]);  
 }  
}

ProductDaoTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\util\dao\ProductDao.php'**;  
  
**class** ProductDaoTest **extends** \Codeception\Test\Unit  
{  
 **private $expectedParam**;  
  
 **public function** testAddNotProductExpectInvalidArgumentException()  
 {  
 $dummy = $this->make(mysqli::**class**);  
 $productDao = **new** ProductDao($dummy);  
 $notProductObject = **"test"**;  
 $this->expectException(InvalidArgumentException::**class**);  
 $productDao->add($notProductObject);  
 }  
  
 **public function** testAddProductExpectQuery()  
 {  
 $name = **"Motherboard"**;  
 $price = 231;  
 $description = **"lalala"**;  
 $category\_id = 14;  
 $category = **new** Category($category\_id, **"test"**);  
 $product = **new** Product(11, $name, $category, $price, $description);  
 $table\_name = **'products'**;  
 $query = **"INSERT INTO `**$table\_name**` (`name`, `price`, `category\_id`, `description`)   
 VALUES ('**$name**', '**$price**', '**$category\_id**', '**$description**')"**;  
 $mysqliSpy = $this->createMysqliQuerySpy($query);  
 $productDao = **new** ProductDao($mysqliSpy);  
 $productDao->add($product);  
 }  
  
 **public function** testUpdateNotProductExpectInvalidArgumentException()  
 {  
 $dummy = $this->make(mysqli::**class**);  
 $productDao = **new** ProductDao($dummy);  
 $notProductObject = **"test"**;  
 $this->expectException(InvalidArgumentException::**class**);  
 $productDao->update($notProductObject);  
 }  
  
 **public function** testUpdateProductExpectQuery()  
 {  
 $id = 11;  
 $name = **"Motherboard"**;  
 $price = 231;  
 $description = **"lalala"**;  
 $category\_id = 14;  
 $category = **new** Category($category\_id, **"test"**);  
 $product = **new** Product($id, $name, $category, $price, $description);  
 $table\_name = **'products'**;  
 $query = **"UPDATE `**$table\_name**` SET `name` = '**$name**', `price` = '**$price**',  
 `category\_id` = '**$category\_id**', `description` = '**$description**' WHERE `id` = '**$id**'"**;  
 $mysqliSpy = $this->createMysqliQuerySpy($query);  
 $productDao = **new** ProductDao($mysqliSpy);  
 $productDao->update($product);  
 }  
  
 **private function** createMysqliQuerySpy($expectedParam)  
 {  
 $this->**expectedParam** = $expectedParam;  
 **return** $this->make(mysqli::**class**,  
 [**"query"** => **function**($actualParam)  
 { $this->assertEquals($this->**expectedParam**, $actualParam); }]);  
 }  
}

UserDaoTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\util\dao\UserDao.php'**;  
  
**class** UserDaoTest **extends** \Codeception\Test\Unit  
{  
 **private $expectedParam**;  
  
 **private function** createMysqliQuerySpy($expectedParam)  
 {  
 $this->**expectedParam** = $expectedParam;  
 **return** $this->make(mysqli::**class**,  
 [**"query"** => **function**($actualParam)  
 { $this->assertEquals($this->**expectedParam**, $actualParam); }]);  
 }  
  
 **public function** testAddNotUserExpectInvalidArgumentException()  
 {  
 $dummy = $this->make(mysqli::**class**);  
 $userDao = **new** UserDao($dummy);  
 $notUserObject = **"test"**;  
 $this->expectException(InvalidArgumentException::**class**);  
 $userDao->add($notUserObject);  
 }  
  
 **public function** testAddCategoryExpectQuery()  
 {  
 $login = **"loginp"**;  
 $email = **"lalala@lalal.la"**;  
 $password = **"qwertytest"**;  
 $user = **new** User(11, $login, $email, $password);  
 $table\_name = **'users'**;  
 $query = **"INSERT INTO `**$table\_name**` (`login`, `email`, `password`)   
 VALUES ('**$login**', '**$email**', '**$password**')"**;  
 $mysqliSpy = $this->createMysqliQuerySpy($query);  
 $userDao = **new** UserDao($mysqliSpy);  
 $userDao->add($user);  
 }  
  
 **public function** testUpdateNotUserExpectInvalidArgumentException()  
 {  
 $dummy = $this->make(mysqli::**class**);  
 $userDao = **new** UserDao($dummy);  
 $notUserObject = **"test"**;  
 $this->expectException(InvalidArgumentException::**class**);  
 $userDao->update($notUserObject);  
 }  
  
 **public function** testUpdateCategoryExpectQuery()  
 {  
 $id = 22;  
 $login = **"loginp"**;  
 $email = **"lalala@lalal.la"**;  
 $password = **"qwertytest"**;  
 $user = **new** User($id, $login, $email, $password);  
 $table\_name = **'users'**;  
 $query = **"UPDATE `**$table\_name**` SET `login` = '**$login**', `email` = '**$email**',  
 `password` = '**$password**' WHERE `id` = '**$id**'"**;  
 $mysqliSpy = $this->createMysqliQuerySpy($query);  
 $userDao = **new** UserDao($mysqliSpy);  
 $userDao->update($user);  
 }  
}

AuthenticationVerifyerTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\logic\AuthenticationVerifyer.php'**;  
  
**class** AuthenticationVerifyerTest **extends** \Codeception\Test\Unit  
{  
 **private $errors\_msg**;  
 **private $iterate**;  
 **private $user**;  
  
 **protected function** \_before()  
 {  
 $this->**errors\_msg** = **array**();  
 $this->**errors\_msg**[**"incorrect\_login"**] = **"Некорректно введен логин"**;  
 $this->**errors\_msg**[**"incorrect\_password"**] = **"Некорректно введен пароль"**;  
 $this->**errors\_msg**[**"not\_user"**] = **"Неправильный логин или пароль"**;  
 }  
  
 **public function** testAuthenticationErrorsIncorrectLoginExpectMessage()  
 {  
 $login = **"ы"**;  
 $password = **"124345665"**;  
 $verifyer = **new** AuthenticationVerifyer($this->**errors\_msg**);  
 $expected = $this->**errors\_msg**[**"incorrect\_login"**];  
 $actual = $verifyer->authenticationErrors($login, $password, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testAuthenticationErrorsIncorrectPasswordExpectMessage()  
 {  
 $login = **"truelogin"**;  
 $password = **"12"**;  
 $verifyer = **new** AuthenticationVerifyer($this->**errors\_msg**);  
 $expected = $this->**errors\_msg**[**"incorrect\_password"**];  
 $actual = $verifyer->authenticationErrors($login, $password, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testAuthenticationErrorsNotExistUserExpectMessage()  
 {  
 $login = **"truelogin"**;  
 $password = **"12123425"**;  
 $verifyer = $this->construct(AuthenticationVerifyer::**class**,  
 [$this->**errors\_msg**], [**"isTrueUser"** => **false**]);  
 $expected = $this->**errors\_msg**[**"not\_user"**];  
 $actual = $verifyer->authenticationErrors($login, $password, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testAuthenticationErrorsExistUserExpectFalse()  
 {  
 $login = **"truelogin"**;  
 $password = **"12123425"**;  
 $verifyer = $this->construct(AuthenticationVerifyer::**class**,  
 [$this->**errors\_msg**], [**"isTrueUser"** => **true**]);  
 $actual = $verifyer->authenticationErrors($login, $password, **null**);  
 $this->assertFalse($actual);  
 }  
  
 **public function** testIsTrueUserUserNotExistExpectFalse()  
 {  
 $login = **"truelogin"**;  
 $verifyer = $this->construct(AuthenticationVerifyer::**class**,  
 [$this->**errors\_msg**], [**"isExistUser"** => **false**]);  
 $actual = $verifyer->isTrueUser($login, **""**, **null**);  
 $this->assertFalse($actual);  
 }  
  
 **public function** testIsTrueUserUserExistExpectTrue()  
 {  
 $this->**iterate** = 1;  
 $login = **"truelogin"**;  
 $password = **"12345678"**;  
 $email = **"lalala@lalala.la"**;  
 $password\_hash = *password\_hash*($password, ***PASSWORD\_DEFAULT***);  
 $this->**user** = [ **"id"** => 1, **"login"** => $login, **"email"** => $email,  
 **"password"** => $password\_hash];  
 $res = $this->make(mysqli\_result::**class**, [**"fetch\_assoc"** => **function**() {  
 **if** ($this->**iterate** & 1 == 1) {  
 $this->**iterate**++;  
 **return** $this->**user**;  
 }  
 $this->**iterate**++;  
 **return false**;  
 }]);  
 $mysqliStub = $this->make(mysqli::**class**,[**"query"** => $res]);  
 $verifyer = $this->construct(AuthenticationVerifyer::**class**,  
 [$this->**errors\_msg**]);  
 $actual = $verifyer->isTrueUser($login, $password, $mysqliStub);  
 $this->assertTrue($actual);  
 }  
}

AuthenticatorTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\logic\Authenticator.php'**;  
  
**class** AuthenticatorTest **extends** \Codeception\Test\Unit  
{  
 **protected function** \_before()  
 {  
 $\_SESSION = **array**();  
 }  
  
 **public function** testAuthenticateWithErrorExpectNoActin()  
 {  
 $login = **"login"**;  
 $error = **true**;  
 $authenticator = **new** Authenticator();  
 $expected = $\_SESSION;  
 $authenticator->authenticate($login, $error);  
 $actual = $\_SESSION;  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testAuthenticateWithoutErrorExpectAuthenticate()  
 {  
 $login = **"login"**;  
 $error = **false**;  
 $authenticator = **new** Authenticator();  
 $expected = **array**(**"logged\_user"** => **"login"**);  
 $authenticator->authenticate($login, $error);  
 $actual = $\_SESSION;  
 $this->assertEquals($expected, $actual);  
 }  
}

CashierTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\logic\Cashier.php'**;  
  
**class** CashierTest **extends** \Codeception\Test\Unit  
{  
 **protected function** \_after()  
 {  
 **if** (**isset**($\_SESSION[**"logged\_user"**]))  
 {  
 **unset**($\_SESSION[**"logged\_user"**]);  
 }  
 }  
  
 **public function** testGetTotalPriceCartNotArrayInvalidArgumentException()  
 {  
 **self**::*expectException*(InvalidArgumentException::**class**);  
 $cashier = Cashier::*getCashier*();  
 $cart = 2;  
 $cashier->getTotalPrice($cart);  
 }  
  
 **public function** testGetTotalPriceCartEmptyArrayZero()  
 {  
 $cashier = Cashier::*getCashier*();  
 $cart = **array**();  
 $expected = 0;  
 $actual = $cashier->getTotalPrice($cart);  
 **self**::*assertEquals*($expected, $actual);  
 }  
  
 **public function** testGetTotalPriceCartArrayWithoutCartRecordsZero()  
 {  
 $cashier = Cashier::*getCashier*();  
 $cart = **array**(**"lalala"**, 34);  
 $expected = 0;  
 $actual = $cashier->getTotalPrice($cart);  
 **self**::*assertEquals*($expected, $actual);  
 }  
  
 **public function** testGetTotalPriceCartArrayWithFiveProductsPriceThirteenResultSixtyFive()  
 {  
 $cashier = Cashier::*getCashier*();  
 $cart = **array**(**new** CartRecord  
 (**new** Product(12, **"i5 3570"**, **new** Category(11, **"CPU"**),  
 13, **"3-th gen"**), 5));  
 $expected = 65;  
 $actual = $cashier->getTotalPrice($cart);  
 **self**::*assertEquals*($expected, $actual);  
 }  
}

CategoryVerifyerTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\logic\CategoryVerifyer.php'**;  
  
**class** CategoryVerifyerTest **extends** \Codeception\Test\Unit  
{  
 **private $verifyer**;  
 **protected $tester**;  
  
 **protected function** \_before()  
 {  
 $errors\_msg = **array**();  
 $errors\_msg[**"incorrect\_name"**] = **"1"**;  
 $errors\_msg[**"double\_name"**] = **"2"**;  
 $this->**verifyer** = **new** CategoryVerifyer($errors\_msg);  
 }  
  
 **public function** testCategoryErrorsIncorrectNameMessageOne()  
 {  
 $res = $this->make(mysqli\_result::**class**, [**'fetch\_assoc'** => **false**]);  
 $db = $this->make(mysqli::**class**, [**'query'** => $res]);  
 $name = **"V"**;  
 $expected = **"1"**;  
 $actual = $this->**verifyer**->categoryErrors($name, $db);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testCategoryErrorsExistNameMessageTwo()  
 {  
 $errors\_msg = **array**();  
 $errors\_msg[**"incorrect\_name"**] = **"1"**;  
 $errors\_msg[**"double\_name"**] = **"2"**;  
 $verifyer = $this->construct(CategoryVerifyer::**class**, [$errors\_msg],  
 [**'isExistCategoryName'** => **true**]);  
 $name = **"VR"**;  
 $expected = **"2"**;  
 $actual = $verifyer->categoryErrors($name, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testCategoryErrorsValidNameFalse()  
 {  
 $name = **"VR"**;  
 $res = $this->make(mysqli\_result::**class**, [**'fetch\_assoc'** => **false**]);  
 $db = $this->make(mysqli::**class**, [**'query'** => $res]);  
 $actual = $this->**verifyer**->categoryErrors($name, $db);  
 $this->assertFalse($actual);  
 }  
}

FaceControlTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\logic\FaceControl.php'**;  
  
**class** FaceControlTest **extends** \Codeception\Test\Unit  
{  
 **protected function** \_after()  
 {  
 **if** (**isset**($\_SESSION[**"logged\_user"**]))  
 {  
 **unset**($\_SESSION[**"logged\_user"**]);  
 }  
 }  
  
 **public function** testGetOneOfGuestFortyTwo()  
 {  
 $faceControl = FaceControl::*getFaceControl*();  
 $expected = 42;  
 $actual = $faceControl->getOneOf($expected, 1, 2);  
 **self**::*assertEquals*($expected, $actual);  
 }  
  
 **public function** testGetOneOfUserTwentyOne()  
 {  
 $faceControl = FaceControl::*getFaceControl*();  
 $\_SESSION[**"logged\_user"**] = **"username"**;  
 $expected = 21;  
 $actual = $faceControl->getOneOf(1, $expected, 2);  
 **self**::*assertEquals*($expected, $actual);  
 }  
  
 **public function** testGetOneOfIsAdminFiftyFive()  
 {  
 $faceControl = FaceControl::*getFaceControl*();  
 $\_SESSION[**"logged\_user"**] = **"admin"**;  
 $expected = 55;  
 $actual = $faceControl->getOneOf(0, 1, $expected);  
 **self**::*assertEquals*($expected, $actual);  
 }  
}

OrderVerifyerTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\logic\OrderVerifyer.php'**;  
  
**class** OrderVerifyerTest **extends** \Codeception\Test\Unit  
{  
 **private $errors\_msg**;  
  
 **protected function** \_before()  
 {  
 $this->**errors\_msg** = **array**();  
 $this->**errors\_msg**[**"incorrect\_address"**] = **"Некорректный адрес"**;  
 $this->**errors\_msg**[**"incorrect\_phone"**] = **"Некорректный телефон"**;  
 }  
  
 **public function** testOrderErrorsIncorrectAddressExpectMessage()  
 {  
 $address = **"Htlr"**;  
 $phone = **"1234567"**;  
 $verifyer = **new** OrderVerifyer($this->**errors\_msg**);  
 $expected = $this->**errors\_msg**[**"incorrect\_address"**];  
 $actual = $verifyer->orderErrors($address, $phone);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testOrderErrorsIncorrectPhoneExpectMessage()  
 {  
 $address = **"Htlr street"**;  
 $phone = **"1234"**;  
 $verifyer = **new** OrderVerifyer($this->**errors\_msg**);  
 $expected = $this->**errors\_msg**[**"incorrect\_phone"**];  
 $actual = $verifyer->orderErrors($address, $phone);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testOrderErrorsCorrectOrderExpectFalse()  
 {  
 $address = **"Htlr street"**;  
 $phone = **"1234567"**;  
 $verifyer = **new** OrderVerifyer($this->**errors\_msg**);  
 $actual = $verifyer->orderErrors($address, $phone);  
 $this->assertFalse($actual);  
 }  
}

ProductVerifyerTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\logic\ProductVerifyer.php'**;  
  
**class** ProductVerifyerTest **extends** \Codeception\Test\Unit  
{  
 **private $errors\_msg**;  
  
 **protected function** \_before()  
 {  
 $this->**errors\_msg** = **array**();  
 $this->**errors\_msg**[**"incorrect\_name"**] = **"1"**;  
 $this->**errors\_msg**[**"incorrect\_price"**] = **"2"**;  
 $this->**errors\_msg**[**"big\_description"**] = **"3"**;  
 $this->**errors\_msg**[**"double\_name"**] = **"4"**;  
 }  
  
 **public function** testProductErrorsIncorrectNameExpectMessage()  
 {  
 $name = **"a"**;  
 $price = 23;  
 $description = **"good good"**;  
 $verifyer = **new** ProductVerifyer($this->**errors\_msg**);  
 $expected = $this->**errors\_msg**[**"incorrect\_name"**];  
 $actual = $verifyer->productErrors($name, $price, $description, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testProductErrorsIncorrectPriceExpectMessage()  
 {  
 $name = **"intel core i3"**;  
 $price = -23;  
 $description = **"good good"**;  
 $verifyer = **new** ProductVerifyer($this->**errors\_msg**);  
 $expected = $this->**errors\_msg**[**"incorrect\_price"**];  
 $actual = $verifyer->productErrors($name, $price, $description, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testProductErrorsIncorrectDescriptionExpectMessage()  
 {  
 $name = **"intel core i3"**;  
 $price = 23;  
 $description = **""**;  
 $verifyer = $this->construct(ProductVerifyer::**class**,  
 [$this->**errors\_msg**], [**"isCorrectDescription"** => **false**]);  
 $expected = $this->**errors\_msg**[**"big\_description"**];  
 $actual = $verifyer->productErrors($name, $price, $description, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testProductErrorsExistNameExpectMessage()  
 {  
 $name = **"intel core i3"**;  
 $price = 23;  
 $description = **"lalala"**;  
 $verifyer = $this->construct(ProductVerifyer::**class**,  
 [$this->**errors\_msg**], [**"isExistName"** => **true**]);  
 $expected = $this->**errors\_msg**[**"double\_name"**];  
 $actual = $verifyer->productErrors($name, $price, $description, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testProductErrorsCorrectProductExpectFalse()  
 {  
 $name = **"intel core i3"**;  
 $price = 23;  
 $description = **"lalala"**;  
 $verifyer = $this->construct(ProductVerifyer::**class**,  
 [$this->**errors\_msg**], [**"isExistName"** => **false**]);  
 $actual = $verifyer->productErrors($name, $price, $description, **null**);  
 $this->assertFalse($actual);  
 }  
}

RegistrationVerifyerTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\logic\RegistrationVerifyer.php'**;  
  
**class** RegistrationVerifyerTest **extends** \Codeception\Test\Unit  
{  
 **private $errors\_msg**;  
  
 **protected function** \_before()  
 {  
 $this->**errors\_msg** = **array**();  
 $this->**errors\_msg**[**"incorrect\_login"**] = **"1"**;  
 $this->**errors\_msg**[**"incorrect\_password"**] = **"2"**;  
 $this->**errors\_msg**[**"incorrect\_email"**] = **"3"**;  
 $this->**errors\_msg**[**"double\_login"**] = **"4"**;  
 $this->**errors\_msg**[**"double\_email"**] = **"5"**;  
 $this->**errors\_msg**[**"incorrect\_repeat"**] = **"6"**;  
 }  
  
 **public function** testRegistrationErrorsIncorrectLoginExpectMessage()  
 {  
 $login = **"la"**;  
 $password = **"123456789"**;  
 $email = **"lalala@lala.la"**;  
 $password2 = $password;  
 $verifyer = **new** RegistrationVerifyer($this->**errors\_msg**);  
 $expected = $this->**errors\_msg**[**"incorrect\_login"**];  
 $actual = $verifyer->registrationErrors($login, $email,  
 $password, $password2, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testRegistrationErrorsIncorrectPasswordExpectMessage()  
 {  
 $login = **"lalalalal"**;  
 $password = **"12"**;  
 $email = **"lalala@lala.la"**;  
 $password2 = $password;  
 $verifyer = **new** RegistrationVerifyer($this->**errors\_msg**);  
 $expected = $this->**errors\_msg**[**"incorrect\_password"**];  
 $actual = $verifyer->registrationErrors($login, $email,  
 $password, $password2, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testRegistrationErrorsIncorrectEmailExpectMessage()  
 {  
 $login = **"lalalalal"**;  
 $password = **"123456789"**;  
 $email = **"lalalalala.la"**;  
 $password2 = $password;  
 $verifyer = **new** RegistrationVerifyer($this->**errors\_msg**);  
 $expected = $this->**errors\_msg**[**"incorrect\_email"**];  
 $actual = $verifyer->registrationErrors($login, $email,  
 $password, $password2, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testRegistrationErrorsExistLoginExpectMessage()  
 {  
 $login = **"lalalalal"**;  
 $password = **"123456789"**;  
 $email = **"lalala@lala.la"**;  
 $password2 = $password;  
 $verifyer = $this->construct(RegistrationVerifyer::**class**,  
 [$this->**errors\_msg**], [**"isExistUser"** => **true**]);  
 $expected = $this->**errors\_msg**[**"double\_login"**];  
 $actual = $verifyer->registrationErrors($login, $email,  
 $password, $password2, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testRegistrationErrorsExistEmailExpectMessage()  
 {  
 $login = **"lalalalal"**;  
 $password = **"123456789"**;  
 $email = **"lalala@lala.la"**;  
 $password2 = $password;  
 $verifyer = $this->construct(RegistrationVerifyer::**class**,  
 [$this->**errors\_msg**], [**"isExistEmail"** => **true**, **"isExistUser"** => **false**]);  
 $expected = $this->**errors\_msg**[**"double\_email"**];  
 $actual = $verifyer->registrationErrors($login, $email,  
 $password, $password2, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testRegistrationErrorsIncorrectRepeatExpectMessage()  
 {  
 $login = **"lalalalal"**;  
 $password = **"123456789"**;  
 $email = **"lalala@lala.la"**;  
 $password2 = **"12345789"**;  
 $verifyer = $this->construct(RegistrationVerifyer::**class**,  
 [$this->**errors\_msg**], [**"isExistEmail"** => **false**, **"isExistUser"** => **false**]);  
 $expected = $this->**errors\_msg**[**"incorrect\_repeat"**];  
 $actual = $verifyer->registrationErrors($login, $email,  
 $password, $password2, **null**);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testRegistrationErrorsCorrectUserExpectFalse()  
 {  
 $login = **"lalalalal"**;  
 $password = **"123456789"**;  
 $email = **"lalala@lala.la"**;  
 $password2 = $password;  
 $verifyer = $this->construct(RegistrationVerifyer::**class**,  
 [$this->**errors\_msg**], [**"isExistEmail"** => **false**, **"isExistUser"** => **false**]);  
 $actual = $verifyer->registrationErrors($login, $email,  
 $password, $password2, **null**);  
 $this->assertFalse($actual);  
 }  
}

RegistratorTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\logic\Registrator.php'**;  
  
**class** RegistratorTest **extends** \Codeception\Test\Unit  
{  
 **public function** testRegistrateUserWithErrorExpectNoAction()  
 {  
 $mysqliMock = $this->make(mysqli::**class**,  
 [**"query"** => \Codeception\Stub\Expected::*never*()]);  
 $login = **"lalalal"**;  
 $email = **"lalalal@lalala.la"**;  
 $password = **"12345678"**;  
 $registrator = **new** Registrator();  
 $registrator->registrateUser($login, $email, $password, **true**, $mysqliMock);  
 }  
  
 **public function** testRegistrateUserWithoutErrorExpectAction()  
 {  
 $mysqliMock = $this->make(mysqli::**class**,  
 [**"query"** => \Codeception\Stub\Expected::*once*()]);  
 $login = **"lalalal"**;  
 $email = **"lalalal@lalala.la"**;  
 $password = **"12345678"**;  
 $registrator = **new** Registrator();  
 $registrator->registrateUser($login, $email, $password, **false**, $mysqliMock);  
 }  
}

URIResolverTest.php

**<?php  
  
require\_once *\_\_DIR\_\_*** . **'\..\..\model\logic\URIResolver.php'**;  
  
**class** URIResolverTest **extends** \Codeception\Test\Unit  
{  
 **public function** testClearURIWithValuesClearURI()  
 {  
 $uri = **"http://localhost:1234/index.php?value1=4&lala=2&value3=abc"**;  
 $expected = **"http://localhost:1234/index.php"**;  
 $actual = URIResolver::*getURIResolver*()->clearURI($uri);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testClearURIWithoutValuesSameURI()  
 {  
 $uri = **"http://localhost:1234/index.php"**;  
 $expected = **"http://localhost:1234/index.php"**;  
 $actual = URIResolver::*getURIResolver*()->clearURI($uri);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetValueNoValueExpectFalse()  
 {  
 $name = **"name"**;  
 $actual = URIResolver::*getURIResolver*()->getValue($name);  
 $this->assertFalse($actual);  
 }  
  
 **public function** testGetValueHasValueExpectValue()  
 {  
 $name = **"name"**;  
 $value = **"value"**;  
 $\_GET[$name] = $value;  
 $expected = $value;  
 $actual = URIResolver::*getURIResolver*()->getValue($name);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testGetPOSTValueNoValueExpectFalse()  
 {  
 $name = **"name"**;  
 $actual = URIResolver::*getURIResolver*()->getPOSTValue($name);  
 $this->assertFalse($actual);  
 }  
  
 **public function** testGetPOSTValueHasValueExpectValue()  
 {  
 $name = **"name"**;  
 $value = **"value"**;  
 $\_POST[$name] = $value;  
 $expected = $value;  
 $actual = URIResolver::*getURIResolver*()->getPOSTValue($name);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testSetToURINoValuesExpectAddToURI()  
 {  
 $name = **"name"**;  
 $value = **"value"**;  
 $uri = **"http://localhost:1234/index.php"**;  
 $expected = **"http://localhost:1234/index.php?**$name**=**$value**"**;  
 $actual = URIResolver::*getURIResolver*()->setToURI($uri, $name, $value);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testSetToURIHasOnlyOneValueExpectAddToURI()  
 {  
 $name = **"name"**;  
 $value = **"value"**;  
 $uri = **"http://localhost:1234/index.php?val1=1"**;  
 $expected = **"http://localhost:1234/index.php?val1=1&**$name**=**$value**"**;  
 $actual = URIResolver::*getURIResolver*()->setToURI($uri, $name, $value);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testSetToURIHasNameValueExpectSetToURI()  
 {  
 $name = **"name"**;  
 $value = **"value"**;  
 $uri = **"http://localhost:1234/index.php?val1=1&**$name**=666&val2=2"**;  
 $expected = **"http://localhost:1234/index.php?val1=1&**$name**=**$value**&val2=2"**;  
 $actual = URIResolver::*getURIResolver*()->setToURI($uri, $name, $value);  
 $this->assertEquals($expected, $actual);  
 }  
  
 **public function** testSetToURINoNameExpectInsertToURI()  
 {  
 $name = **"name"**;  
 $value = **"value"**;  
 $uri = **"http://localhost:1234/index.php?val1=1&val2=2"**;  
 $expected = **"http://localhost:1234/index.php?val1=1&val2=2&**$name**=**$value**"**;  
 $actual = URIResolver::*getURIResolver*()->setToURI($uri, $name, $value);  
 $this->assertEquals($expected, $actual);  
 }  
}