1. **Магические числа.** Особо нет. Разве что в файле запуска самого приложения в функции make\_server. Думаю, более чем достаточно будет передать аргументы именованными.
2. **Спагетти код.** Не обнаружено. Всё у нас более или менее разбито по модулям.
3. **Лазанья код.** Ну тут сложно судить, так как приложение совсем маленькое, но в будущем, если развивать приложение, наверняка это возникнет.
4. **Слепая вера.** Все формы. В шаблонных классах вьюшек и в их наследников никакой валидации. Так же в нашей ORM. У нас нет никакого разделения на типы данных. В базу данные заносятся без проверки типа данных. Выход касательно вьюшек – написать валидацию данных, касательно базы – использовать готовую ORM, не изобретать велосипед))
5. **Шифрованный код.** Не обнаружено.
6. **Жёсткое кодирование.** Хост и порт при запуске приложения.Решение – запросить аргументами хост и порт, но при этом оставить значения по умолчанию.
7. **Мягкое кодирование.** Не обнаружено.
8. **Поток лавы.** Опять же отсутствие валидации в формах. При росте приложения и последующих обработках возникнут проблемы. Решение – обязательная валидация и при росте приложения своевременное написание тестов.
9. **Боязнь размещать логику в объектах предметной области.** Большинство вьюшек уже переделаны правильно, и вся логика перемещена в фабрики и в модели. Но остались ещё вью, которые содержат слишком много логики. Решение – из вью ItemCopy и ItemCreate перенести логику в Item и ItemFactory.
10. **Божественные классы.** В первую очередь – это наш класс Engine. Решение – все методы в нём нужно сделать более универсальными, создать какие-то дополнительные классы-фабрики для объектов. В меньшей степени – класс Application. При добавлении нового функционала к нашему фреймворку, нужно не дописывать методы в нём, а писать классы с новым функционалом и подключать их к главному классу Application.
11. **Полтергейст.** Слава богу в этом проекте пока не появлялся. Но чтобы в дальнейшем этого не происходило, нужно обязательно постоянно дебажить новый функционал со строгим контролем от куда, куда и что у нас приходит. Расстановка правильных эксепшинов также поможет быстрее понимать эти неожиданно всплывающие ошибки.
12. **Сплошное одиночество.** Наш класс Logger правильно использует этот паттерн.
13. **Приватизация.** Некоторые функции в моём приложении названы как защищённые. Это сделано только ради того, чтобы и я со временем, и другие разработчики, которым может придётся читать мой код, просто понимали, что изначально использование этих объектов рассчитано только внутри текущего модуля. Даже завёл себе привычку базовые шаблоны называть как защищённые, типа \_base.html.
14. **Python антипаттерны.** В некоторых классах в инициализации есть что-то типа того: self.buyers = []

self.sellers = []

и т.д.

Наверное можно было бы использовать распаковку: self.buyers, self.sellers = [], []

Не знал, что это антипаттерн.

1. По методологическим паттернам у нас всё более менее хорошо, т.к. курс именно об этом. Можно выделить **Фактор невероятности** т.к. никаких валидаций у нас нет, не хватало на это время. А с **Преждевременной оптимизацией** я немного столкнулся, во время выполнения некоторых дз, т.к. не всегда знал что эти моменты будут рассматриваться на уроках с более профессиональной точки. Итог – потеря времени на то, что потом пришлось переписывать. **Изобретение велосипеда** – это был смысл нашего курса, плюс в том, что по мимо изучения самих паттернов, пришло более глубокое понимание работы известных фреймворков.
2. **Инверсия абстракций.** Не обнаружено.
3. **Большой комок грязи.** Не обнаружено.
4. **Затычка на ввод.** Если я правильно понял, то в нашем проекте это встречается в некоторых местах. Такие, как тип пользователя или тип товара. У нас жёстко прописано ‘buyer’, ‘seller’ и т.д. Нужно создать некий файл с переменными, которые будут содержать эти данные.
5. **Волшебная кнопка.** Вроде бы не обнаружена.
6. **Членовредительство.** Не обнаружено.
7. **Дымоход предприятия и системы.** Ещё не доросли.
8. **Путаница.** Не обнаружено.