# Проектирование

Целью проектирования является определение того, как будет реализована функциональность логической модели, построенной на этапе анализа. Анализ предметной области завершён, создана аналитическая модель.

## Проектные классы

Проектные классы – это классы, описание которых настолько полно, что они могут быть реализованы [умл2]. Описание формируется путем уточнения классов анализа, которое включает в себя добавлений деталей реализации. На рисунке 1 представлена часть диаграммы проектных классов, реализуемых в системе. Эта часть диаграммы отображает группу классов, относящихся к «Заведениям».

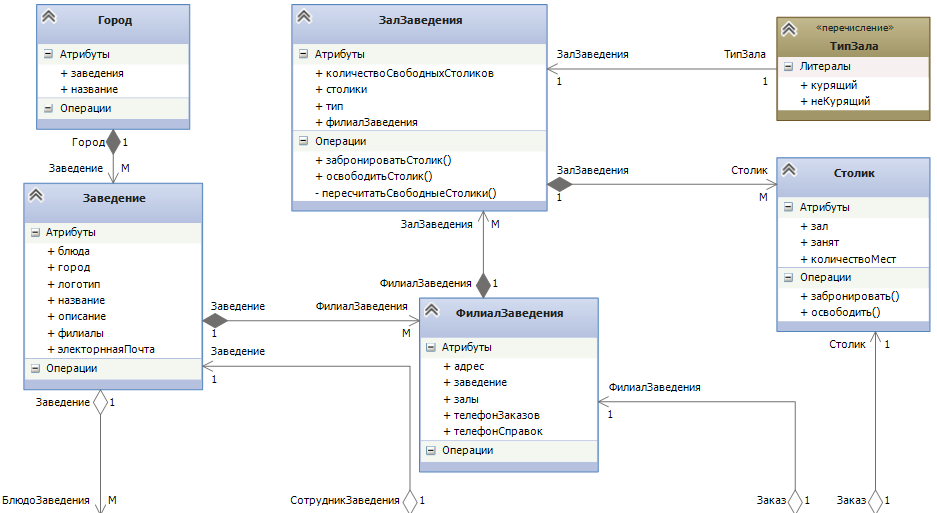


Рисунок 1 – Диаграмма проектных классов

Если сравнивать эту часть диаграммы с аналогичной частью диаграммы классов анализа, то очевидно, что изменения минимальны, а именно, объекты класса «Город» приобрели новое свойство – «Заведения», в котором содержится список заведений города. Еще одним изменением является уточнение отношений «Заведение» - «Блюдо Заведения» (агрегация, один ко многим), «Заведение» - «Сотрудник Заведения» (агрегация, один к одному), «Филиал» - «Заказ» (агрегация, один ко многим), «Заказ» - «Столик» (агрегация, один к одному).

На рисунке 2 представлена часть диаграммы, которая отображает группу классов, относящихся к «Блюдам».

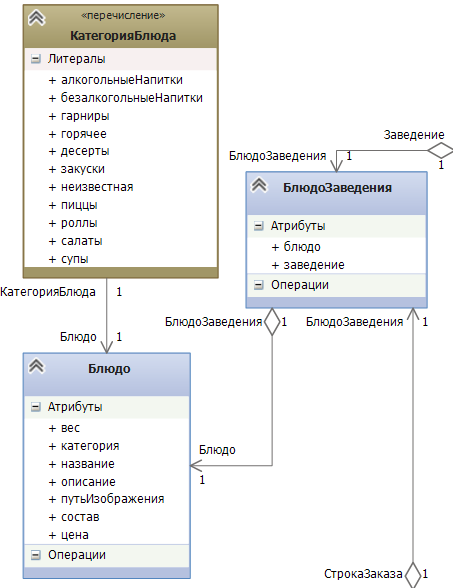


Рисунок 2 – Диаграмма проектных классов

Эта часть также не претерпела сильных изменений, помимо уже указанных были изменены отношения «Блюдо Заведения» - «Блюдо» (агрегация, один к одному), «Строка заказа» - «Блюдо Заведения» (агрегация, один к одному). Здесь необходимо сказать о том, что классы «Корзина Заказов» и «Строка Корзины Заказов» были заменены на «Строка Заказа». Это связано с тем, что было принято решение хранить «Корзину» в сессии.

На рисунке 3 представлена часть диаграммы, которая отображает группу классов, относящихся к «Заказам».

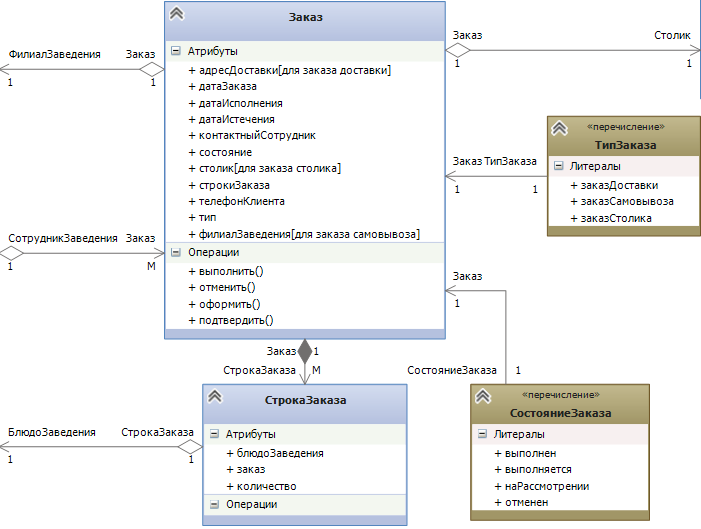


Рисунок 3 – Диаграмма проектных классов

Изменения в этой части более существенны, так как, как уже указывалось, «Корзина Заказов» была вынесена в сессию. Поэтому добавился класс «Строка Заказов», а также добавились и изменились отношения: «Заказ» - «Строка Заказов» (композиция, один ко многим), «Сотрудник Заведения – «Заказ» (композиция, один ко многим).

На рисунке 4 представлена последняя часть диаграммы, которая отображает группу классов, относящихся к «Управлению Заведений».

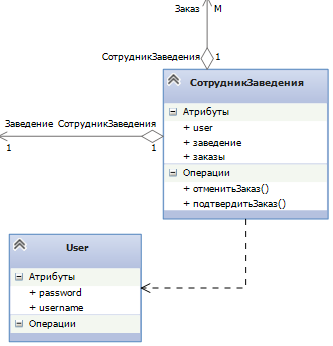


Рисунок 4 – Диаграмма проектных классов

Здесь было решено добавить класс «User». Это класс Django, который необходим для авторизации, от него зависят все пользователи системы.

## Диаграмма пакетов системы

Пакет (package) – это инструмент группирования, который позволяет взять любую конструкцию UML и объединить ее элементы в единицы высокого уровня. При этом каждый элемент (класс) может принадлежать только одному пакету. Пакеты же могут входить в состав других пакетов [http://www.planerka.info/item/Diagrammy-paketov].

На рисунке 5 представлена диаграмма пакетов. Она состоит из пакетов: «Заведения», «Блюда», «Заказы», «Управление Заведений».

Пакет «Заведения» содержит все классы, содержащие информацию о заведении (местоположение и планировка) – для ориентации в заведениях, бронирования столиков, самовывоза. Этот пакет зависит от пакета «Блюда». От этого пакета зависят пакеты «Заказы» и «Управление Заведений».

Пакет «Блюда» содержит классы, содержащие информацию о блюдах, а также класс, связывающий блюда с заведением. От этого пакета зависят пакеты «Заведения» и «Заказы».

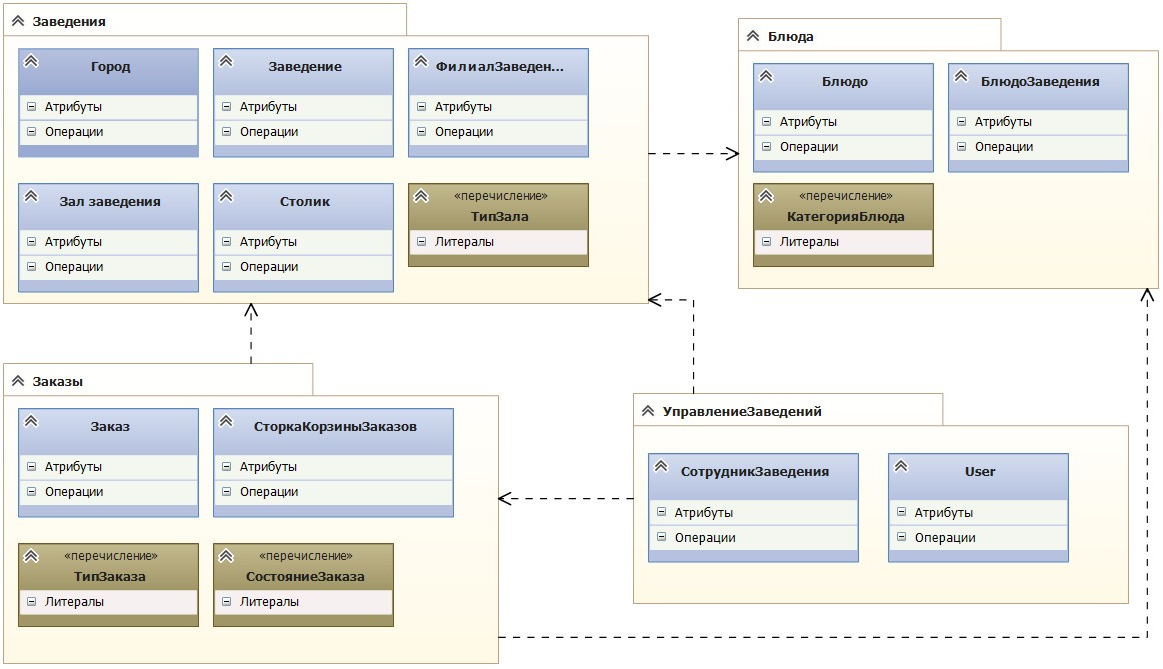


Рисунок 5 – Диаграмма пакетов

Пакет «Заказы» содержит классы для создания и хранения заказов. Этот пакет зависит от пакетов «Заведения» и «Блюда».

Пакет «Управление заведений» содержит класс, объект которого – аккаунт сотрудника, который может просматривать, подтверждать или отменять заказы. Этот пакет зависит от пакета «Заведения».

Пакет SQL database является базой данных, используемой для хранения информации, необходимой для работы системы.

## Диаграммы последовательностей для операций проектных классов

Диаграмма последовательности (англ. sequence diagram) — диаграмма, на которой показано взаимодействие объектов (обмен между ними сигналами и сообщениями), упорядоченное по времени, с отражением продолжительности обработки и последовательности их проявления [https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0\_%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8]. Неотъемлемой частью объекта на диаграмме последовательности является линия жизни объекта. Линия жизни показывает время, в течение которого объект существует в Системе. Периоды активности объекта в момент взаимодействия показываются с помощью фокуса управления. Временная шкала на диаграмме направлена сверху вниз [http://it-gost.ru/articles/view\_articles/94].

В ходе выполнения курсового проекта были спроектированы диаграммы последовательностей для нескольких функций. На рисунке 6 изображена диаграмма последовательности для метода get\_context\_data(self, \*\*kwargs) класса DishesList во views приложения Dishes.

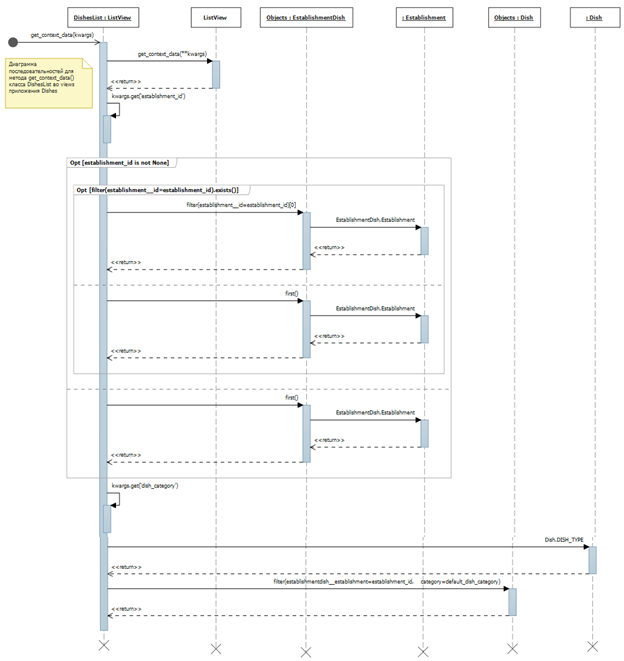


Рисунок 6 – Диаграмма последовательности get\_context\_data(self, \*\*kwargs) в DishesList

При вызове get\_context\_data(self, \*\*kwargs) происходит создание нового context, которому передается результат вызова super().get\_context\_data(\*\*kwargs). После этого задается параметр establishment\_id=self.kwargs.get('establishment\_id'). Далее осуществляется двойная проверка: первая – что в establishment\_id есть значение, вторая – фильтрует заведения по id и возвращает True, если заведение найдено. Если обе проверки пройдены успешно, то переменной current\_establishment присваивается найденный объект-заведение, иначе присваивается первое заведение из списка. Далее задается переменная dish\_category=self.kwargs.get('dish\_category'), после чего происходит получение списка Dish.DISH\_TYPE и осуществляется фильтрация по establishment\_id и default\_dish\_category, с целью получения списка блюд указанной категории в определенном заведении.

На рисунке 7 изображена диаграмма последовательности для метода get\_context\_data(self, \*\*kwargs) класса EstablishmentList во views приложения Establishments.

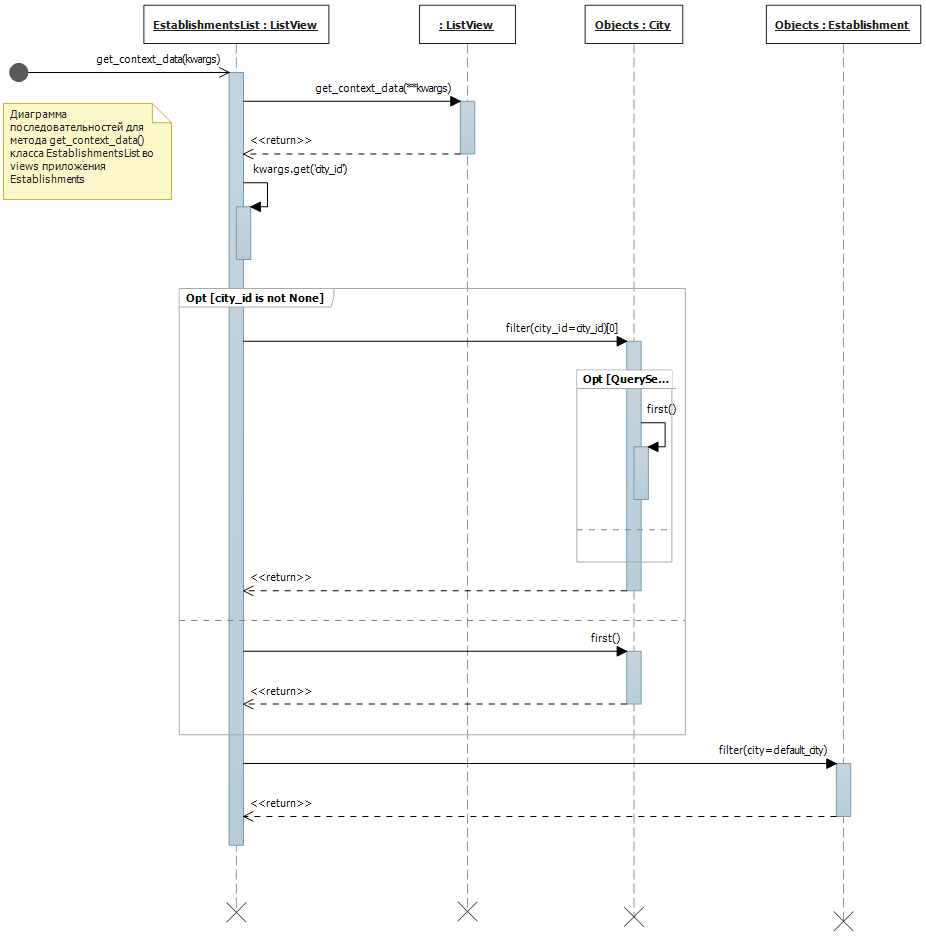


Рисунок 7 – Диаграмма последовательности get\_context\_data(self, \*\*kwargs) в EstablishmentList

При вызове get\_context\_data(self, \*\*kwargs) происходит создание нового context, которому передается результат вызова super().get\_context\_data(\*\*kwargs). После этого задается параметр city\_id=self.kwargs.get(city\_id'). Далее осуществляется проверка того, есть ли в city\_id значение. Если значение есть, то переменной default\_city присваивается найденный по фильтру или первый объект-город, иначе присваивается первый город из списка. Затем по фильтру city=default\_city возвращается список заведений города.

На рисунке 8 изображена диаграмма последовательности для метода increment\_dish(request) во views приложения Orders.

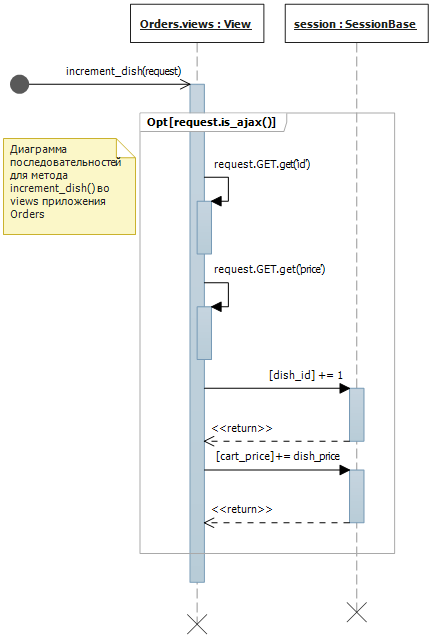


Рисунок 8 – Диаграмма последовательности increment\_dish(request) в Orders

Сценарий описываемой последовательности сообщений между объектами программной системы вызывается при вызове метода increment\_dish файла «views.py» приложения «orders». Вызов данной функции осуществляется по запросу ajax со страницы просмотра корзины заказов, отсылаемому серверу по событию нажатия на кнопку увеличения количества порций блюда.

Сценарий осуществляется только в том случае, если переданный запрос является запросом ajax. Это проверяется с помощью функции is\_ajax() переданного в функцию объекта request. После этого происходит получение значений ключей, переданный в GET запросе с помощью функции get() объекта request.GET типа QueryDict. Также из объекта сессии (session: SessionBase) извлекается значение ключа – полученного из объекта request идентификатора блюда (dish\_id) – которое является количеством порций данного блюда в заказе. С помощью объекта модели класса Dish, соответствующего полученному идентификатору, извлекается цена блюда. Если количество блюд в корзине стало равным нулю, то данный ключ удаляется из объекта сессии с помощью метода pop(dish\_id, None). В противном случае значение ключа, соответствующее количеству порций блюда, уменьшается на величину переданного в запросе значения количества с помощью метода \_\_setitem\_\_. Затем из объекта сессии извлекается цена корзины заказов до нажатия на кнопку уменьшения порций блюд с помощью метода get(‘cart\_price’). Если уменьшение цены корзины на величину цены выбранного блюда приводит к тому, что цена корзины становится равной нулю, то значение, содержащее в себе цену корзины, в объекте сессии удаляется с помощью метода pop(‘cart\_price’, None). В противном случае значение данного ключа уменьшается на величину цены блюда с помощью метода \_\_setitem\_\_ объекта сессии, устанавливающего новое значение для указанного ключа.

На рисунке 9 изображена диаграмма последовательности для метода get\_context\_data(self, \*\*kwargs) класса EmployeePage во views приложения Employees.

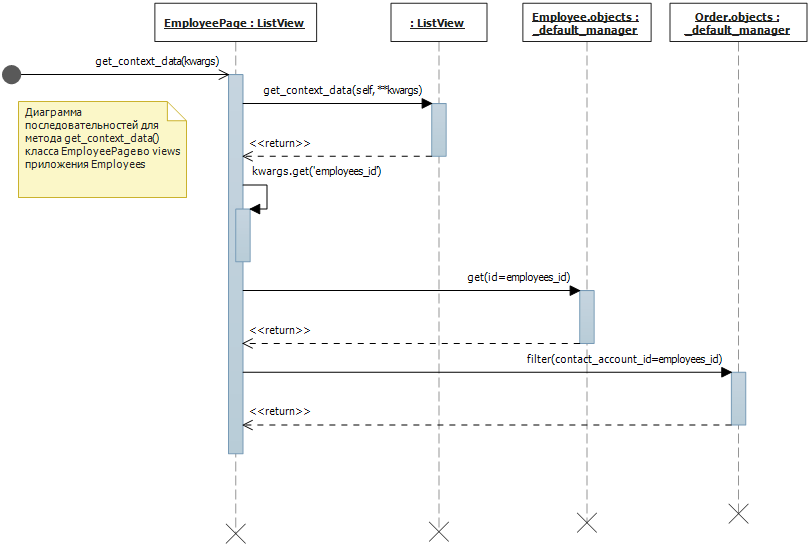


Рисунок 9 – Диаграмма последовательности get\_context\_data(self, \*\*kwargs) в Employees

Сценарий описываемой последовательности сообщений между объектами программной системы вызывается при вызове метода get\_context\_data файла «views.py» приложения «employees». Вызов данной функции осуществляется при переходе приложения по следующему адресу: url(r'(?P<employees\_id>\d+)/$', EmployeePage.as\_view(), name='employees'). При вызове метода get\_context\_data(self, \*\*kwargs) создается объект context, которому передается результат вызова super().get\_context\_data(\*\*kwargs). После этого извлекается идентификатор сотрудника employees\_id с помощью функции self.kwargs.get('employees\_id'). С помощью менеджера по умолчанию объектов Employee извлекается сотрудник заведения, соответствующий полученному идентификатору из строки запроса. В конце с помощью менеджера по умолчанию объектов Order по идентификатору сотрудника создается объект order\_list.

На рисунке 10 изображена диаграмма последовательности для метода login\_view(request, login\_state) во views приложения Emloyees.

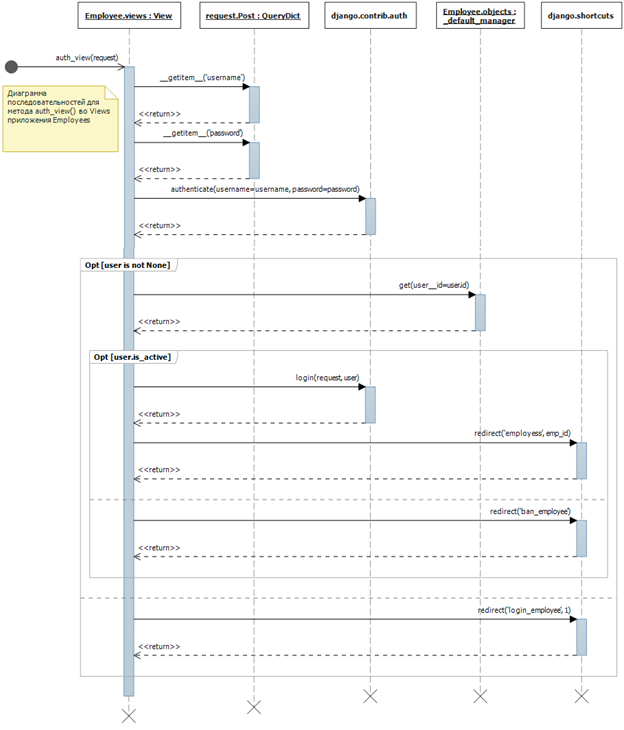


Рисунок 10 – Диаграмма последовательности login\_view(request, login\_state)в Employees

Сценарий описываемой последовательности сообщений между объектами программной системы вызывается при вызове метода auth\_view файла «views.py» приложения «employees». Вызов данной функции осуществляется при переходе приложения по ссылке: url(r'^accounts/after\_login/$', views.auth\_view, name='after\_login'), после нажатия кнопки «Войти» на странице authentication.html. После этого происходит получение значений ключей, переданных в POST запросе с помощью функции \_\_getitem\_\_() объекта request.POST типа QueryDict. Далее объекту user присваивается результат аутентификации по связке «логин-пароль» при помощи auth.authenticate(username=username, password=password). После этого объект user проверяется по двум параметрам: во-первых – есть ли такой пользователь, а если нет, то происходит переход по ссылке url(r'^accounts/login/(?P<login\_state>\d+)?', views.login\_view, name='login\_employee') на страницу 'employees/authentication.html' и выводится предупреждение о неверном вводе имени или пароля. Если первая проверка прошла успешно, то пользователь проверяется на предмет наличия блокировки (user.is\_active). Если пользователь не заблокирован, происходит его авторизация auth.login(request, user) и переход по ссылке url(r'(?P<employees\_id>\d+)/$', EmployeePage.as\_view(), name='employees') на страницу employees.html. Если пользователь не прошел проверку происходит переход по ссылке url(r'^accounts/ban/$', views.ban\_view, name='ban\_employee') на страницу ban.html.