

Customer - the customer model. The User model technically holds the account-specific information, but convenience accessors exist in the customer model.

User - account information, username, password, etc.

Address - The user's address. Convenience methods allow access to building's attribute Building - contains addresses. Addresses are otherwise just unit numbers and designation

Everything here is pretty much a straight one-to-one mapping. The only trick is that you need to create the related objects before you can call the new ones. So, if you were to create an Address, for example, you would need to create a Building object if it did not already exist. That is:

a = Address.new(:label => "home", :unit designation => "apt", :unit number => "1-b", :building => Building.find or create(args), :customer => Customer.find(me))

which is simple enough, I guess. By and large, the fields in the form correspond 1-1 with some attribute of either Customer (which has the same relationship with User that Address has with Building). The only way to really inform yourself would be to look at the data backend of the models, so here goes:

mysql> DESCRIBE addresses;

	Field	Туре	Null	Key	Default	Extra	l		1 11000	1 120		11022	1	ļ	
	+	19PC 	Nuii 	KC <u>y</u> 	Deruure		! -	16 rows in set (0.01 s	+====== sec)		тт				
	id	int(11)	NO	PRI	NULL	auto_increment									
	label	varchar(20)	YES		NULL			+ Field	+		+ Null	+	Default		,+
	building_id	int(11)	YES		NULL			rieid	1	ype	- Null	Key	Deraurt	Extra	1
	customer_id	int(11)	YES		NULL			id	i	nt(11)	NO	PRI	NULL	auto increment	i
	unit_designation	varchar(12)	YES		NULL			first_name	1	archar(255)	YES	j	NULL	_	į
	unit_number	varchar(8)	YES		NULL			last_name	7	archar(255)	YES		NULL		ĺ
-	+	- 	+	- 	⊦	- 	 -	login	7	archar(255)	YES		NULL		

6 rows in set (0.02 sec)

mysgl> DESCRIBE buildings;

+-----

+-----

Field

city

state

zip

doorman

schedule

serviced

RIBE	E buildings;					crypted_password
 +	Туре	Null	Key	Default	Extra	salt created_at updated at
 	<pre>int(11) varchar(255) varchar(255) varchar(255) varchar(255)</pre>	NO YES YES YES YES	PRI	NULL NULL NULL NULL NULL	auto_increment	remember_token_e user_class account_id account_type referrer
į	varchar(9)	YES		NULL	į	16 rows in set (0.
 id 	<pre>tinyint(1) int(11) tinyint(1)</pre>	YES YES YES		NULL NULL NULL		The same principles their account information

9 rows in set (0.01 sec)

Once both objects are instantiated, you can reach one through the other. (ie, customer.addr1 is set to point to the instance's building's addr1 attribute). I will try to simplify the creation so this caveat need not be kept in mind, but for now it is a necessity.

Field	Туре	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
title	varchar(255)	YES	İ	NULL	į –
company	varchar(255)	YES		NULL	
sex	varchar(1)	YES		NULL	
dob	date	YES		NULL	
\mid primary_address_id \mid	int(11)	YES		NULL	
work	varchar(10)	YES		NULL	
home	varchar(10)	YES		NULL	
cell	varchar(10)	YES		NULL	
active	tinyint(1)	YES		NULL	
accepted_terms	tinyint(1)	YES		NULL	
points	int(11)	YES		NULL	
created_on	datetime	YES		NULL	
updated_on	datetime	YES		NULL	
carbon_credits	float	YES		NULL	
water_credits	float	YES		NULL	

email

maiden

remember token

	Type	Null	Key	Default	Extra	my_fresh_air
_name name	int(11) varchar(255) varchar(255) varchar(255)	NO YES YES YES	PRI 	NULL NULL NULL	auto_increment	wants_updates wants_promotion email_format wants_minor_re
	varchar(255)	YES	j	NULL		23 rows in set (
ber_token	varchar(255)	YES		NULL		20 10 11 200 (
n	varchar(255)	YES		NULL		which should all be
ed_password	varchar(40)	YES		NULL		Willett Stibula all De
	varchar(40)	YES		NULL		
ed_at	datetime	YES		NULL		
ed_at	datetime	YES		NULL		
ber_token_expires_at	datetime	YES		NULL		
class	varchar(20)	YES		NULL		
nt_id	int(11)	YES		NULL		
nt_type	varchar(255)	YES		NULL		
rer	int(11)	YES		NULL		

count information. The only real difference would be that the User model has other subclasses, (such as driver, washer, etc.). Also, the access methods available to user through customer are limited to those fields we associate with a customer (ie, a first name, last name, etc.) while account-like attributes are accessible through customer.account.attribute name., which includes things like email address, login/password, etc. You'll have to refer to the model code, but it's

Lastly we have the customer preferences

Field	Type	Null	Key	Defaul
id	int(11)	NO	PRI	NULL
customer_id	int(11)	YES	ĺ	NULL
wf_temperature	varchar(255)	YES	ĺ	NULL
wf_fabric_softener	tinyint(1)	YES	ĺ	NULL
wf_bleach	tinyint(1)	YES	ĺ	NULL
ls_starch	varchar(255)	YES	ĺ	NULL
ls_press	varchar(255)	YES	ĺ	NULL
ls_packaging	varchar(255)	YES	ĺ	NULL
dc_starch	varchar(255)	YES	ĺ	NULL
dc_press	varchar(255)	YES	ĺ	NULL
permanent_tags	tinyint(1)	YES	ĺ	NULL
day_before_email	tinyint(1)	YES	ĺ	NULL
day_before_sms	tinyint(1)	YES		NULL
day_of_email	tinyint(1)	YES		NULL
day_of_sms	tinyint(1)	YES		NULL
preferred_contact	varchar(255)	YES		NULL
doorman_permission	tinyint(1)	YES		NULL
my fresh water	tinyint(1)	YES		NULL

updates minor repairs | tinyint(1) | YES | NULL in set (0.02 sec)

s in set (0.02 sec)

me principles apply in this case. Each customer is backed by a user class, which houses all of just a simple passthrough for convenience's sake.

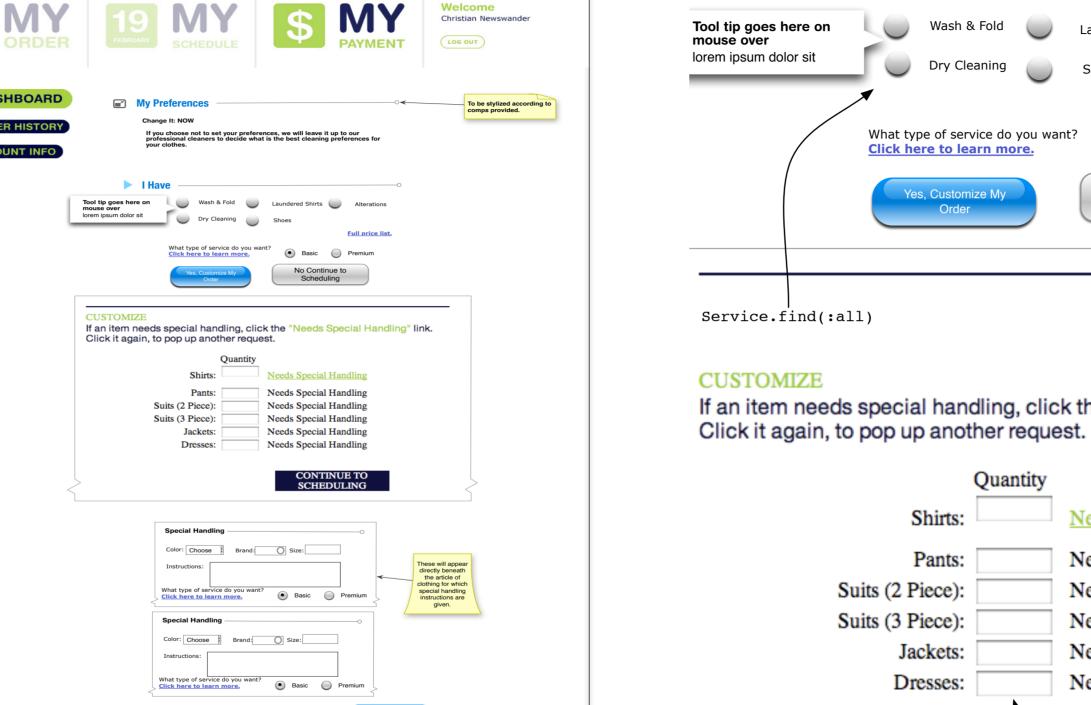
••	p				
m	er_preferences;				
-· -·	+	+ Null +	+ Key +	+ Default +	+
	int(11)	NO	PRI	NULL	auto_increment
	int(11)	YES	İ	NULL	<u> </u>
	varchar(255)	YES	İ	NULL	İ
	tinyint(1)	YES	ĺ	NULL	
	tinyint(1)	YES	ĺ	NULL	
	varchar(255)	YES	ĺ	NULL	
	varchar(255)	YES		NULL	
	varchar(255)	YES		NULL	
	varchar(255)	YES		NULL	
	varchar(255)	YES		NULL	
	tinyint(1)	YES		NULL	
	tinyint(1)	YES	ļ	NULL	
	tinyint(1)	YES	ļ	NULL	
	tinyint(1)	YES	ļ	NULL	
	tinyint(1)	YES	ļ	NULL	
	varchar(255)	YES	ļ	NULL	
	tinyint(1)	YES	ļ	NULL	
	tinyint(1)	YES	ļ	NULL	
	tinyint(1)	YES	ļ	NULL	
	tinyint(1)	YES	ļ	NULL	
	tinyint(1)	YES		NULL	

order_build.graffle > order_build

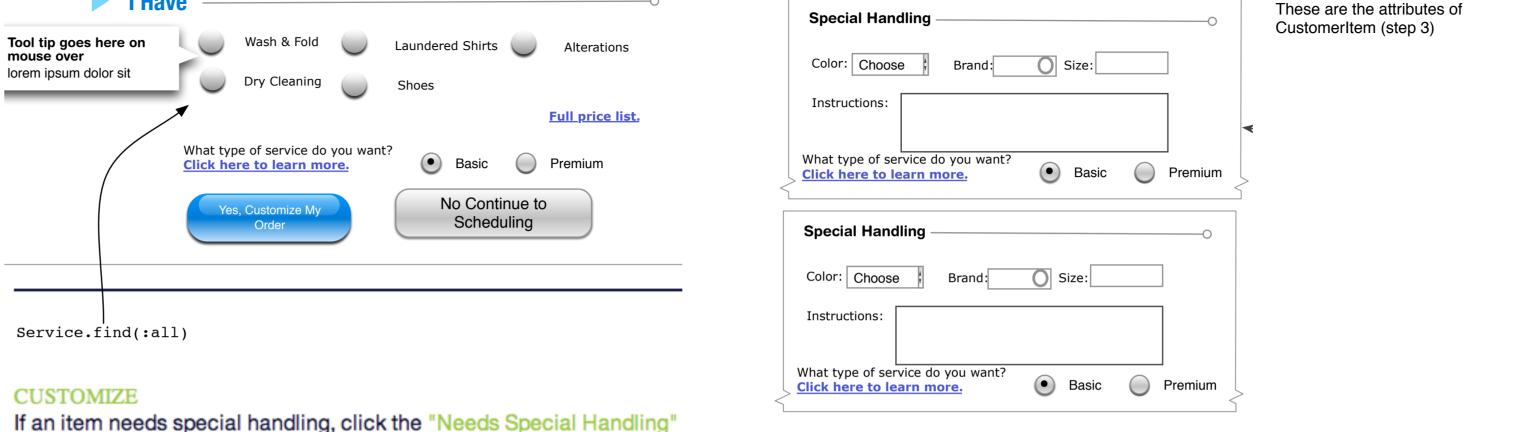
Models of Interest:

Service - This is how we organize this part

hould all be rather straightforward



Thu Mar 13 2008 Modified: Wed Mar 19 2008 ItemTypes - the classes of items available for service. Order - Hold the general order information OrderItem - A particular item in an order. CustomerItem - The detail about a particular item of clothing



So this should happen in the following stages:

Quantity

Needs Special Handling

Needs Special Handling

Needs Special Handling

Needs Special Handling

Needs Special Handling

COLUMN TITE MO

create x new OrderItems of the appropriate.

an attribute of CustomerItem. (see step 2 examples or

next page)

type where x =the input of these fields "type" is actually

Shirts:

service.applicable item types()

1) The user selects the service(s) he or she is interested in.

2) If the user wants to customize (clicks on the blue button), then a Customize block appears fo each service chosen. The customize block allows the user to define how many of each item the order will contain. (These fields should only allow numeric input). The controller should create a new OrderItem-CustomerItem pair for each article of clothing input. For example, if the customer specifies 1 shirt and 2 pairs of pants, the controller should do something like:

OrderItem.create(:customer item => CustomerItem.create(:type => "pants") OrderItem.create(:customer item => CustomerItem.create(:type => "pants") OrderItem.create(:customer item => CustomerItem.create(:type => "shirt")

3) If the user wants to specify special instructions, they click on the Needs Special handling link This allows them to fill out the rest of the field in the customer items for one of the OrderItems you just defined. These inputs need to appear under the input field in the Customize section described in step 2 (and illustrated on the previous page) so the customer can see at a glance what type of item it is (shirt, pants, 2 pc suit, etc.) 4) When the user is done, they can:

Your building has been registered with My Fresh Shirt Your building has a special pick up date: Your building has a special delivery date: Picking these times will help us be more efficient and friendly to the nvironment by allowing us to make less trips and less stops, as a reward, ou will receive 100 points. Learn More About our Doorman Service THU FRI SAT SUN MON TUE WED Feb 21 Feb 22 Feb 23 Feb 24 Feb 25 Feb 26 Feb 27

THU FRI SAT SUN MON TUE WED
Feb 21 Feb 22 Feb 23 Feb 24 Feb 25 Feb 26 Feb 27 x 12-2 pm ○ 8-10 8-10 0 8-10 10-12 0 10-12 0 10-12 0 10-12 0 10-12 SCHEDULE MY DELIVERY LATE

My Order Schedule Summary My Pick Up TUESDAY 10am-12pm My Delivery THURSDAY 2pm-4pm

ontinue to Payment

*May incur a fee, please read more info!

Return to Order Descri

Models of Interest:

Schedule - an array of Stops that are available to address. If no stop is available, nil is returned. Window - an array of Windows (time ranges) against which an order may be placed Request - a customer request for pickup or delivery.

My Pick Up schedule.to s(format) [2] x 12-2 pm [3] x 2-4 [5] 0 6-8 [6] 08-10

Schedule #array of Stops

4) Now the customer has to pick out *when* he wants to accept pickups and deliveries. This wor exactly the same way as the administrative schedule (please refer to mfs admin schedule for more information; but in this case, since we don't have an axis of windows, you can just use Stop's window attribute to get the window's start and end times). The radio button sets for "My Pick Up" and "My Delivery" should be distinct. If a slot is available, you make requests against the calling make request(order, type), with type being either: pickup or: delivery. This will return a request object of the appropriate type (which will need to be saved). Note that an error condition may occur due to the last of a Stop's available slots being consumed while it the page. The page will have to ask the customer to reschedule, and present an updated grid.

(highlight if stop.concordant with(address))

x appears if no slots available

(if stop.available? == false)

radio button if slots available

(if stop.available? == true)