Radosav Radovanović

jul, 2022., Lazarevac

Jam.py programiranje

Contents

[Jam.py programiranje 5](#_Toc109586253)

[Task stablo 5](#_Toc109586254)

[Tok procesa 7](#_Toc109586255)

[Rad sa modulima 8](#_Toc109586256)

[Programiranje na strani klijenta 9](#_Toc109586257)

[Index.html 9](#_Toc109586258)

[Inicijalizacija aplikacije 9](#_Toc109586259)

[Forme 10](#_Toc109586260)

[Šabloni formi 11](#_Toc109586261)

[Eventi forme 12](#_Toc109586262)

[Opcije forme 16](#_Toc109586263)

[Kontrole svesne podataka 17](#_Toc109586264)

[Programiranje podataka 18](#_Toc109586265)

[Dataset 18](#_Toc109586266)

[Navigacija dataseta 19](#_Toc109586267)

[Modifikovanje dataseta 20](#_Toc109586268)

[Polja 21](#_Toc109586269)

[Lookup polja 22](#_Toc109586270)

[Filtriranje zapisa 23](#_Toc109586271)

[Detalji 25](#_Toc109586272)

[Programiranje na strani servera 28](#_Toc109586273)

[Programiranje izveštaja 30](#_Toc109586274)

[Šabloni izveštaja 30](#_Toc109586275)

[Kreiranje izveštaja 30](#_Toc109586276)

[Parametri izveštaja 30](#_Toc109586277)

[Programiranje izveštaja na klijentu 31](#_Toc109586278)

[Programiranje izveštaja na serveru 32](#_Toc109586279)

[Jam.py FAQ 34](#_Toc109586280)

[Koja je razlika izmedju kataloga i žurnala? 34](#_Toc109586281)

[Kako podići već kreirani projekat na novu verziju jampy frejmvorka? 34](#_Toc109586282)

[Šta su strani ključevi? 34](#_Toc109586283)

[Mogu li koristiti druge biblioteke u mojoj aplikaciji? 34](#_Toc109586284)

[Kada štampam izveštaj dobijam ods umesto pdf fajla? 34](#_Toc109586285)

[Kako da …? 36](#_Toc109586286)

[Kako da migriram razvoj na produkciju? 36](#_Toc109586287)

[Migracija novog projekta 36](#_Toc109586288)

[Migracija postojećeg projekta 36](#_Toc109586289)

[Uvoz meta podataka sa zaustavljanjem servera 36](#_Toc109586290)

[Uvoz meta podataka bez zaustavljanja servera 36](#_Toc109586291)

[Kako migrirati na drugu bazu podataka? 37](#_Toc109586292)

[Kako da napišem funkciju vidljivu u globalnom opsegu? 37](#_Toc109586293)

[Kako validirati vrednost polja? 38](#_Toc109586294)

[Kako dodati dugme na formu? 39](#_Toc109586295)

[Kako izvršiti serverski skript sa klijenta 39](#_Toc109586296)

[Kako da promenimo stil i atribute elemenata forme? 40](#_Toc109586297)

[Kako dodati zapis koristeći edit formu bez otvaranja view forme? 42](#_Toc109586298)

[Kako zabraniti promenu zapisa? 43](#_Toc109586299)

[Kako da povežem dva objekta? 43](#_Toc109586300)

[Kako promeniti vrednost polja izabranih zapisa? 46](#_Toc109586301)

[Kako da upišem edit formu bez zatvaranja iste? 48](#_Toc109586302)

[Kako da sačuvam promene u dve tabele u istoj transakciji na serveru? 48](#_Toc109586303)

[Kako sprečiti duplicirane vrednosti u polju tabele 49](#_Toc109586304)

[Kako implementirati osnovni multi-tenancy? 49](#_Toc109586305)

[Kao mogu koristiti podatke iz tabele druge baze podataka? 50](#_Toc109586306)

[Kako mogu procesirati zahtev ili dobiti neke podatke iz druge aplikacije ili servisa? 52](#_Toc109586307)

[Kako mogu izvršiti kalkulacije u pozadini? 53](#_Toc109586308)

[Da li su podržani detalji unutar detalja? 53](#_Toc109586309)

[Izvoz u/Uvoz iz CSV fajlova 55](#_Toc109586310)

[Autentifikacija 56](#_Toc109586311)

[Kako autentifikovati iz prilagodjene korisničke tabele? 56](#_Toc109586312)

[Kako kreirati registracionu formu? 57](#_Toc109586313)

[Kako dati korisniku mogućnost da menja lozinku? 59](#_Toc109586314)

[Integracija sa postojećom bazom podataka 61](#_Toc109586315)

[Zaključavanje zapisa 62](#_Toc109586316)

[Čuvanje istorije promena koje su napravli korisnici 62](#_Toc109586317)

[Jam.py reference klasa 63](#_Toc109586318)

[Klijentska strana - javascript reference 63](#_Toc109586319)

[*class* AbstractItem() 63](#_Toc109586320)

[*class* Task() 69](#_Toc109586321)

[*class* Group() 76](#_Toc109586322)

[*class* Item() 78](#_Toc109586323)

[*class* Detail() 117](#_Toc109586324)

[*class* Reports() 117](#_Toc109586325)

[*class* Report() 117](#_Toc109586326)

[*class* Field() 121](#_Toc109586327)

[*class* Filter() 124](#_Toc109586328)

[Serverska strana - python reference 126](#_Toc109586329)

[class App 126](#_Toc109586330)

[*class* AbstractItem 126](#_Toc109586331)

[class Task 127](#_Toc109586332)

[*class* Group 133](#_Toc109586333)

[class Item 133](#_Toc109586334)

[*class* Detail 147](#_Toc109586335)

[*class* Reports 147](#_Toc109586336)

[*class* Report 147](#_Toc109586337)

[*class* Field 150](#_Toc109586338)

[*class* Filter 152](#_Toc109586339)

# Jam.py programiranje

## Task stablo

Svi objekti okvira predstavljaju stablo objekata. Svi *objekti* stabla imaju zajedničkog pretka, klasu *AbstractItem* (klijent/server) i zajedničke atribute:

* ID – jedinstveni objekt id u okviru
* owner – neposredni roditelj i vlasnik objekta
* task – koren stabla objekata
* items - lista objekata dece objekta
* item\_type – tip objekta
* item\_name – ime objekta koje će biti korišćeno u kodu
* item\_caption – ime objekta koje će se prikazivati korisnicima

U korenu stabla je objekat – task. Task sadrži grupe objekata. Postoje tri tipa grupe objekata koje imaju sledeće vrednosti *item\_type* atributa:

* items – ove grupe imaju objekte koje mogu imati pridružene tabele u *db*.
* details – ove grupe takodje sadrže objekte sa pridruženim tabelama u db, ali se one koriste za kreiranje objekata detalja za druge objekte.
* reports – ove grupe sadrže objekte izveštaje.

Možete kreirati vašu vlastitu grupu objekata.

Objekti kojima mogu biti pridružene db tabele mogu imati svoje detalje, koje se koriste za smeštanje zapisa povezanih sa zapisom vodećeg objekta.

Na primer task stablo Demo projekta je:

/demo/

catalogs/

customers

tracks

albums

artists

genres

media\_types

journals/

invoices/

invoice\_table

details/

invoice\_table

reports/

invoice

purchases\_report

customers\_report

U korenu task stabla je task objekat sa *item\_name =* demo. Ovaj objekat ima četiri grupe: *catalogs, journals, details* i *reports*. *Catalogs* i *journals* grupe imaju *item\_type* = items. Objekti koje one sadrže su kao omotači oko odgovarajućih db tabela. Tu je jedan objekat u *details* grupi - sa *item\_name* invoice\_table, koji takodje ima svoju db tabelu, i tri izveštaja u grupi *reports*. Invoices *journal* ima invoice\_table kao *details* objekat, koji sadrži listu pesama jedne korisničke fakture. Dakle dva su objekta istog imena, invoice\_table (*detail\_item* i *detail*). Oni dele istu db tabelu.

Svaki objekat je atribut svog vlasnika i svi objekti, detalji i izveštaji su atributi objekta taska isto tako (svi oni imaju jedinstveno *item\_name*).

Task je globalni objekat na klijentu. Da bi mu pristupili, samo otkucajte task bilo gde u kodu. Na serveru task nije globalni objekat.

Jam.py je event-driven environment. Svaki event ima kao parametar *item* (ili *field*) – objekat koji je pokrenuo event. Funkcije definisane u server modulu objekta mogu biti izvršene iz klijentskog modula korišćenjem *server* metoda, imaju odgovarajući *item* - objekat kao prvi parametar. Znajući objekat, možemo pristupiti bilo kome drugom objektu u task stablu. Na primer, da pristupimo customers katalogu možemo pisati:

def on\_apply(item, delta, params):

customers = item.task.catalogs.customers.copy()

ili samo:

def on\_apply(item, delta, params):

customers = item.task.customers.copy()

Hijerarhijska struktura projekta je jedna od osnova DRY (don’t repeat yourself) principa okvira. Na primer, neke metode objekta, kada su izvršene, sukcesivno generišu evente za *task*, *grupu* i *objekat*. Na taj način možemo definisati osnovno ponašanje za sve objekte u event hendleru taska, koji može biti proširen u event hendleru grupe i na kraju, ako je potrebno, može biti spec. u event hendleru samog objekta.

[Task stablo](https://youtu.be/hsSKqEh6vL4) je video tutorijal koji demonstrira korišćenje taska stabla u Demo projektu.

## Tok procesa

U Jam.py okviru, dva taska rade u isto vreme: *Application* *builder* i *Project*. Svaki od njih predstavlja jedno stablo objekata – postoje *Application* *builder* *task* *stablo* i *Project* *task* *stablo*. Kako bilo, pre nego se upoznate sa Jam.py tokom procesa, trebalo bi da se familijarizujete sa konceptom task stabla.

Jam.py tok procesa je kao što sledi:

* Kada je *server*.*py* pokrenut, on kreira *WSGI* aplikaciju koja kreira *AppBuilder* *task* *stablo*.
* *Project* *task* stablo je kreirano na serveru od strane *AppBbuilder*-a pošto server primi prvi zahtev od klijenta. Da bi to uradio *AppBuilder* koristi meta podatke smeštene u *admin.sqlite* bazi podataka iz root foldera projekta. Posle kreiranja task stabla server aplikacije pokreće *on*\_*created* event na serveru, koji može biti korišćen za inicijalizaciju serverskog task stabla.
* Kada aplikacija na klijentu (*AppBuilder* ili *Project*) je prvi put pokrenuta u brauzeru (pošto je *builder.html* ili *index.html* učitan) prazni task objekat je izgradjen, on šalje zahtev na server za sopstvenu inicijalizaciju.
* Ako je projektni *safe mode* parametar postavljen, okvir proverava da li je korisnik prijavljen, pre izvršenja zahteva. Ako nije, aplikacija kreira na kljentu *login* formu, i posle korisničkog unosa korisničkog imena i lozinke, klijentski task šalje zahtev za prijavu.
* Posle uspešne prijave ili ako projektni *safe mode* parametar nije postavljen, server šalje klijentu informacije o zahtevanom tasku. Task na klijentu gradi stablo, zasnovano na dobijenim informacijama, pridružuje event hendlere na njihove objekte i izvršava *on\_page\_loaded* event hendler.
* U tom event hendleru programer bi trebao da pridruži JQuery event handler funkcije na HTML elemente DOM-a, definisane u *index.html* fajlu. U tim funkcijama programer može da koristi metode objekata task stabla da izvrši neki spec. zadatke. Te metode, kada su izvršene, pokreću različite evente u kojima druge metode mogu biti pozvane itd.
* Objekti task stabla, koji imaju odgovarajuće tabele u db, imaju metode za čitanje i pisanje podataka na bazi podataka servera.
* Objekti izveštaja generišu izveštaje na serveru, zasnovane na LibreOffice šablonima.
* Svi objekti, čiji metodi generišu zahteve serveru, rade to na sledeći način: one pozivaju metod taska koji šalje na server:
  + ID taska,
  + ID objekta,
  + Tip zahteva,
  + Njegove parametre.

Server, kada primi zahtev, nalazi odgovarajući task (*Project* ili *AppBuilder* task) i objekat na serveru, izvršava odgovarajući metod sa prosledjenim parametrima i vraća rezultat izvršenja klijentu. Ovi serverski metodi mogu pokrenuti njihove sopstvene evente koji mogu nadjačati podrazumevano ponašnje.

[Form eventi](https://youtu.be/DY463lcv0R4) i [Klijent-server interakcije](https://youtu.be/nLOhdA2FX0I) su video tutorijali koji ilustruju Jam.py tok procesa.

## Rad sa modulima

Za svaki objekat task stabla programer može da piše kod koji će se izvršiti na klijentu ili serveru. U AppBuider-u za svaki objekat postoje dva gore-desno dugmeta – *klijent modul* i *server* *modul*. Klikom na njih otvoriće se *editor* *koda*. Svaki objekat ima unapred definisan skup *eventa* koji bi se mogao pokrenuti aplikacijom. *Event hendler* je funkcija definisana u modulu objekta, čije ime počinje sa *on\_* prefiksom. Svi objavljeni događaji navedeni su u *kartici* *eventa* informativnog okna *editora* *koda*. U editoru koda programer može da piše kod za ove događaje kao i definiše neke svoje funkcije.

Na primer, sledeći kod znači da odmah neposredno posle dodavanja novog zapisa u Invoices žurnalu Demo projekta, vrednost polja datuma fakture biće jednaka trenutnom datumu.

function on\_after\_append(item) {

item.invoicedate.value = new Date();

}

Napomena:Ovi dogadjaji i funkcije postaju atributi objekta i može im se pristupiti bilo gde u kodu projekta.

Na primer, sledeći kod definisan u klijent modulu objekta će izvršiti *on\_edit\_form\_created* event hendler definisan u Customers objektu.

function on\_edit\_form\_created(item) {

task.customers.on\_edit\_form\_created(item);

}

## Programiranje na strani klijenta

### Index.html

Kada korisnik otvara Jam.py aplikaciju u veb pretraživaču, pretraživač prvo učitava fajl *index.html*. Ovaj fajl se nalazi u root folderu projekta. To je uobičajeni HTML fajl koja sadrži veze do *CSS* i *JS* fajlova, koje aplikacija klijenta koristi. Fajlovi sa čijm učitavanjem Jam.py počinje nalaze se u folderu Jam na serveru. Na primer::

<link href=jam/css/jam.css rel=stylesheet>

Ako je potrebno, drugi fajlovi mogu biti dodani ovde. Na primer neka biblioteka grafikona. Bolje je smestiti ih u *js* i *css* foldere *static* foldera projekta. Na primer:

<script src=static/js/Chart.min.js></script>

*index.html* *body* tag ima div sa klasom *templates*, koji sadrži *html* šablone projektnih formi. Na kraju fajla je sledeći kod:

<script>

$(document).ready(function(){

task.load()

});

</script>

U ovom kodu *load* metod task-a, koji je kreiran kada je *jam.js* fajl bio učitan, je pozvan da učita informacije o stablu task-a i bazirano na tim informacijama, izgradi stablo, učita module, pridruži event hendlere objektima i pokrene *on\_page\_loaded* event hendler.

### Inicijalizacija aplikacije

*on\_page\_loaded* event hendler je prvi event pokrenut od aplikacije na klijentu. Novi projekat koristi *on\_page\_loaded* event handler da dinamički izgradi aplikacioni meni i pridruži *click* event hendler meniju objekta koristeći JQuery.

function on\_page\_loaded(task) {

$(title).text(task.item\_caption);

$(#title).text(task.item\_caption);

if (task.safe\_mode) {

$(#user-info).text(task.user\_info.role\_name + ' ' + task.user\_info.user\_name);

$('#log-out')

.show()

.click(function(e) {

e.preventDefault();

task.logout();

});

}

if (task.full\_width) {

$('#container').removeClass('container').addClass('container-fluid');

}

$('#container').show();

task.create\_menu($(#menu), $(#content), {view\_first: true});

}

Ovaj event handler koristi JQuery da izabere elemente iz *index.html* da postavi njihove atribute i pridruži evente.

<div id=container class=container style=display: none>

<div class=row-fluid>

<div class=span6>

<h3 id=title class=muted></h3>

</div>

<div class=span6 logging-info>

<span id=user-info></span>

<a id=log-out href=# style=display: none>Log out</a>

</div>

</div>

<div class=container>

<div id=taskmenu class=navbar>

<div class=navbar-inner>

<ul id=menu class=nav>

</ul>

</div>

</div>

</div>

<div id=content>

</div>

</div>

Na kraju, *create\_menu* metod task-a je pozvan da dinamički kreira glavni meni projekta.

### Forme

Jedan od ključnih koncepata okvira je koncept forme. Kada korisnik klikne na objekat glavnog menija, *view* metod odgovarajućeg menija je izvršen, koji kreira *view* formu. Ova view forma ima *New* i *Edit* dugmad, klikom na koje će se izvršiti *insert*\_*record* i *edit*\_*record* metode. Ove metode stvaraju formu za editovanje objekta.

Forme su zasnovane na HTML šablonima koji određuju njihov izgled. Šabloni su definisani u *index*.*html* fajlu, smeštenom u root folderu projekta. Aplikacija već ima podrazumevane šablone za *view,* *edit, filter i param* forme. Na primer, sve *edit* forme demo projekta koriste sledeći HTML obrazac:

<div class=default-edit>

<div class=form-body>

<div class=edit-body></div>

<div class=edit-detail></div>

</div>

<div class=form-footer>

<button type=button id=ok-btn class=btn expanded-btn>

<i class=icon-ok></i> OK<small class=muted>&nbsp;[Ctrl+Enter]</small>

</button>

<button type=button id=cancel-btn class=btn expanded-btn>

<i class=icon-remove></i> Cancel

</button>

</div>

</div>

Možete definisati vaš šablon forme za kreiranje prilagodjenih formi.

Kada neki metod kreira formu, aplikacija pronalazi odgovarajući html šablon. Ako je *kontejner* (Jquery objekat) parametar spec., metod prazni i dodaje html šablon unutar kontejnera, inače, kreira praznu modalnu formu i dodaje šablon forme. Posle toga pridružuje šablon na objektov *prefix\_form* atribut, pokreće *on\_prefix\_form\_created* event, prikazuje formu, i pokreće *on\_prefix\_form\_shown* event, gde je prefix tip forme (*view*, *edit*, *filter*, *param*). Ispod je primer *on\_edit\_form\_created* event handler task-a:

function on\_edit\_form\_created(item) {

item.edit\_form.find(#ok-btn).on('click.task',

function() { item.apply\_record() });

item.edit\_form.find(#cancel-btn).on('click.task',

function(e) { item.cancel\_edit(e) }

);

if (!item.master && item.owner.on\_edit\_form\_created) {

item.owner.on\_edit\_form\_created(item);

}

if (item.on\_edit\_form\_created) {

item.on\_edit\_form\_created(item);

}

item.create\_inputs(item.edit\_form.find(.edit-body));

item.create\_detail\_views(item.edit\_form.find(.edit-detail));

return true;

}

U ovom primeru, *find* JQuery metod se koristi za pronalaženje elemenata forme. Prvo pronalazimo OK i Cancel dugmad i pridružujemo JQuery klik evente, tako *cancel\_edit* i *apply\_record* metode će biti izvršene kada korisnik klikne na dugmad. Ove metoda odustaju od ili primenjuju promene napravljene na zapisu i pozivaju *close\_edit\_form* metod za zatvaranje forme.

Ako forma nije forma detalja i ima event hendler *on\_edit\_form\_created*, definisan u vlasnikovom klijentskom modulu, ovaj hendler je izvršen. Posle toga, ako objekat ima event handler *on\_edit\_form\_created*, definisan u objetovom klijent modulu, taj event je izvršen.

U ovim event hendlerima neke dodatne akcije mogu biti izvršene. Na primer, možete pridružiti klik event na dugme ili neki drugi element koji se nalazi u vašem edit šablonu, promeniti *edit\_options*, kreirati tabele koristeći create\_table metod itd.

Potom je *create\_inputs* metod pozvan da kreira ulazne elemente forme unutar elementa klase *edit-body*. Na kraju, *create\_detail\_views* metod je pozvan da kreira detalje u elementu sa klasom *edit-detail*.

Napomena: Ako neki elementi nedostaju u šablonu forme, *exception* neće biti podignut.

*close\_prefix\_form*, gde je *prefix* tip forme, zatvara formu tog tipa. Pre nego što je forma zatvorena, *on\_prefix\_form\_close\_query* i *on\_prefix\_form\_closed* eventi su pokrenuti. Posle toga forma je zatvorena i uklonjena iz DOM-a.

### Šabloni formi

Šabloni formi projekta su locirani u *div*-u sa klasom *templates* unutar *body* taga u *index*.*html* fajlu. Kada je *load* metod izvršen, on odseca *div* sa klasom *templattes* iz *body* taga i smešta šablon u *templates* *atribut* kao *JQuery* objekat.

Da dodate šablon forme za objekat treba da dodate *div* sa *name-suffix* klasom u *templates* div, gde je *name* objektov *item\_name* a *suffix* je tip forme: *view, edit, filter, param*. Na primer:

<div class=invoices-edit>

...

</div>

je šablon edit forme za invoices objekat.

Za objekte detalja, pre imena objekta, trebalo bi da bude ime master objekta, razdvojeno crticom:

<div class=invoices-invoice\_table-edit>

...

</div>

Ako objekat nema svoj šablon forme, tada će se za šablon forme objekta uzeti šablon forme objektovog vlasnika, ako je definisan. Tako šablon

<div class=journals-edit>

...

</div>

će biti korišćen za kreiranje edit forme objekta kome je vlasnik Journals grupa ako taj objekat nema svoj sopstveni šablon edit forme. Ako posle pretrage na taj način šablon nije pronadjen za objekat, šablon sa *default-suffix* klasom će biti korišćen za kreranje forme. Tako šablon

<div class=default-edit>

...

</div>

će biti korišćen za kreiranje edit forme za objekte koji nemaju definisan svoj šablon kao ni njihovi vlasnici. Kada je novi projekat kreiran *index.html* već sadrži takve šablone. Ispod je primer podrazumevanog šablona edit forme iz *index.html* fajla:

<div class=default-edit>

<div class=form-body>

<div class=edit-body></div>

<div class=edit-detail></div>

</div>

<div class=form-footer>

<button type=button id=ok-btn class=btn expanded-btn>

<i class=icon-ok></i> OK<small class=muted>&nbsp;[Ctrl+Enter]</small>

</button>

<button type=button id=cancel-btn class=btn expanded-btn>

<i class=icon-remove></i> Cancel

</button>

</div>

</div>

Postoji više primera šablona u sekciji Primeri šablona forme.

### Eventi forme

Pošto je forma kreirana i HTML šablon forme dodn u DOM, aplikacija pokreće sledeće dogadjaje forme za vreme njenoh životnog veka:

* on\_view\_form\_created - event je pokrenut kada je forma kreirana, ali pre prikazivanja
* on\_view\_form\_shown - event je pokrenut kada je forma prikazana
* on\_view\_form\_close\_query - event je pokrenut kada je napravljen pokušaj zatvaranja forme
* on\_view\_form\_closed - event je pokrenut kada je forma zatvorena
* on\_view\_form\_keydown - event je pokrenut kada se keydown event pojavio u formi
* on\_view\_form\_keyup - event je pokrenut kada se keyup event pojavio u formi

Za druge tipove formi - *edit*, *filter* i *param*, zamenite *view* sa tipom forme, na primer *on\_edit\_form\_created* za edit formu.

Sada ćemo objasniti kako koristiti *on\_view\_form\_created* event.

Kada korisnik klikne na stavku u meniju aplikacija izvršava *view* metod odgovarajućeg objekta task stabla, ovaj metod kreira formu koristeći HTML šablon forme i pokreće *on\_view\_form\_created* event task-a. Kada kreirate novi projekat, task klijent modul već sadrži kod, uključujući *on\_view\_form\_created* event hendler. Ovaj event hendler se izvrši svaki put kada je *view* forma kreirana i definiše podrazumevano ponašanje forme.

Možete otvoriti task klijent modul da vidite ovaj event hendler. Ako je potrebno da promenite podrazumevano ponašanje za sve view forme projekta, trebalo bi to ovde uraditi.

Ispod je opis glavnih koraka da to napravite:

* Inicijalizacija *view* forme i *table\_options*.
* Pridruživanje JQuery event hendlera za podrazumevanu dugmad na metode objekta, zavisno od korisničkih prava. U primeru ispod *delete* dugme je inicijalizovano:

if (item.can\_delete()) {

item.view\_form.find(#delete-btn).on('click.task', function(e) {

e.preventDefault();

item.delete\_record();

});

}

else {

item.view\_form.find(#delete-btn).prop(disabled, true);

}

* Izvršavanje *on\_view\_form\_created* objektove grupe i *on\_view\_form\_created* objekta ako su definisani:

if (!item.master && item.owner.on\_view\_form\_created) {

item.owner.on\_view\_form\_created(item);

}

if (item.on\_view\_form\_created) {

item.on\_view\_form\_created(item);

}

* Kreiranje tabele za prikaz podataka objekta i tabela detalja ako su spec. pozivanjem *create\_view\_tables* metoda.
* Izvršavanje *open* metoda, koji dobija objektov dataset sa servera.
* Na kraju, vrati *true* da sprečiš pozivanje *on\_view\_form\_created* event hendlera vlasnika objekta i samog objekta jer je već pozvan *\_process\_event* metod.

Pošto smo inicijalizovali dugmad i pre ngo što smo kreirali tabelu pozivmo *on\_view\_form\_created* event hendler samog objekta.

Na primer, u klijent modulu tracks objekta demo aplikacije sledeći *on\_view\_form\_created* event hendler je definisan. U njemu menjamo *height* atribut *table\_options*, kreiramo kopiju *invoice\_table* objekta i postavljamo njegov *paginate* atribut i pozivamo *create\_table* metod koji kreira tabelu za prikazivanje podataka.

function on\_view\_form\_created(item) {

item.table\_options.height -= 200;

item.invoice\_table = task.invoice\_table.copy();

item.invoice\_table.paginate = false;

item.invoice\_table.create\_table(item.view\_form.find('.view-detail'), {

height: 200,

summary\_fields: ['date', 'total'],

});

item.alert('Double-click the record in the bottom table to see track sales.');

}

Modul takodje ima *on\_after\_scroll* event hendler koji će biti izvršen kda korisnik predje na drugu pesmu i dobija prodaju te pesme.

Ovaj primer objašnjava principe korišćenja eventa forme. Redosled pokretanja eventa zavisi od tipa eventa.

#### Close query eventi

Kada korisnik pokuša da zatvori formu *on\_close\_query* event je prvo pokrenut (ako je definisan) za objekat. Ako event hendler vrati *true* aplikacija zatvara formu inače ako event hendler vrati *false* aplikacija ostavlja formu otvorenu, inače *on\_close\_query* event je pokrenut (ako je definisan) na isti način za grupu objekata a posle toga za task.

Na primer, podrazumevano *on\_edit\_form\_close\_query* event hendler u task klijent modulu:

function on\_edit\_form\_close\_query(item) {

var result = true;

if (!item.virtual\_table && item.is\_changing()) {

if (item.is\_modified()) {

item.yes\_no\_cancel(task.language.save\_changes,

function() {

item.apply\_record();

},

function() {

item.cancel\_edit();

}

);

result = false;

}

else {

item.cancel\_edit();

}

}

return result;

}

Ovaj kod proverava dali je zapis promenjen i otvara Da Ne Odustani dijalog. Ako želimo da zatvorimo formu bez ovog dijaloga možemo definisati event hendler u klijent modulu objekta na sledeći način:

function on\_edit\_form\_close\_query(item) {

item.cancel()

return true;

}

#### Keydown, keyup eventi

Ovi eventi su pokrenuti na isti način kao *close query eventi*, počinjući od objekta, ako event handler vraća *true*, event hendleri grupe i taska neće biti izvršeni

Na primer, podrazuevano *on\_edit\_form\_keyup* event hendler u task klijent modulu:

function on\_edit\_form\_keyup(item, event) {

if (event.keyCode === 13 && event.ctrlKey === true){

item.edit\_form.find(#ok-btn).focus();

item.apply\_record();

}

}

Ovaj kod upisuje promene na zapisu u tabelu baze podataka kada korisnik pritisne *Ctrl+Enter*.

Pretpostavimo da želimo da upiše promene kada korisnik pritisne *Enter*. Tada ćemo napisati sledeći event hendler u objektovom klijent modulu:

function on\_edit\_form\_keyup(item, event) {

if (event.keyCode === 13){

item.edit\_form.find(#ok-btn).focus();

item.apply\_record();

return true;

}

}

U ovom slučaju event hendler taska neće biti pozvan kada korisnik pritisne *Enter*.

#### Svi drugi eventi

Za druge evente, event hendler *taska* će prvo biti pozvan, ako ne vrati *true,* event hendler *grupe* će biti pozvan*,*  ako ne vrati *true,* event hendler obekta će biti pozvan.

Ovaj mehanizam je implementiran u *\_process\_event* metodu *Item* klase u *jam.js* modulu.

\_process\_event: function(form\_type, event\_type, e) {

var event = 'on\_' + form\_type + '\_form\_' + event\_type,

can\_close;

if (event\_type === 'close\_query') {

if (this[event]) {

can\_close = this[event].call(this, this);

}

if (!this.master && can\_close === undefined && this.owner[event]) {

can\_close = this.owner[event].call(this, this);

}

if (can\_close === undefined && this.task[event]) {

can\_close = this.task[event].call(this, this);

}

return can\_close;

}

else if (event\_type === 'keyup' || event\_type === 'keydown') {

if (this[event]) {

if (this[event].call(this, this, e)) return;

}

if (!this.master && this.owner[event]) {

if (this.owner[event].call(this, this, e)) return;

}

if (this.task[event]) {

if (this.task[event].call(this, this, e)) return;

}

}

else {

if (this.task[event]) {

if (this.task[event].call(this, this)) return;

}

if (!this.master && this.owner[event]) {

if (this.owner[event].call(this, this)) return;

}

if (this[event]) {

if (this[event].call(this, this)) return;

}

}

}

### Opcije forme

Za svaki tip forme objekta postoji atribut koji kontroliše ponašanje modalne forme:

* view\_options
* edit\_options
* filter\_options
* param\_options

Ovaj objekat ima sledeće atribute, spec. parametre modalne forme:

* width – širina modalne forme, podrazmevano 560 px,
* title – naslov modalne forme, podrzumevano to je vrednost *item\_caption* atributa,
* close\_button – ako je *true*, dugme za zatvaranje forme biće kreirano u gornjem desnom uglu forme, podrazumevano *true*,
* close\_caption – ako je true, i *close\_button* je *true*, prikazaće *Close - [Esc]* blizu *close* dugmeta
* close\_on\_escape – ako je true, pritiskom na Escape taster će pokrenuti odgovarajući *close\_form* metod.
* close\_focusout – ako je *true*, odgovarajući *close\_form* metod će biti pozvan kada forma izgubi fokus
* template\_class – ako je spec., div sa ovom klasom će biti tražen u task *templates* atributu i korišćen kao html šablon forme kada se forma kreira.

*edit\_options* ima *fields* atribut, koji spec. listu imena polja koju će *create\_inputs* metod koristiti, ako *fields* atribut *edit*\_*options* parametra nije spec., podrazumevana vrednost biće lista imena polja postavljena u Edit Form Dialog u AppBuilder-u.

*view\_options* ima *fields* atribut, koji specificira listu imena polja koju će koristiti *create\_table* metod, ako *fields* atribut *view*\_*options* parametra nije spec., podrazumevana vrednost biće lista imena polja postavljena u View Form Dialog u AppBuilderu.

Širina modalne edit forme, kreirana u sledećem primeru biće 700 px.

function on\_edit\_form\_created(item) {

item.edit\_options.width = 700;

}

### Kontrole svesne podataka

Za kreiranje tabele za prikazivanje objektovog dataseta koristimo *create\_table* metod:

item.create\_table(item.view\_form.find(.view-table), table\_options);

Za kreiranje kontrola za editovanje polja dataseta koristimo *create\_inputs* metod:

item.create\_inputs(item.edit\_form.find(.edit-body), input\_options);

Ove metode imaju dva parametra, *container* i *options*. Prvi parametar je JQuery kontejner u kome će kontrole biti smeštene. Drugi definiše na koji način će podaci biti prikazani. Za detaljen informacije vidi njihove API reference.

Metode se obično koriste u *on\_view\_form\_created* i o*n\_edit\_form\_created* event hendlerima. Sve vizualne kontrole (tabele, ulazi, čekboksovi), kreirane ovim metodama su svesne podataka. To znači da one trenutno reflektuju bilo koju promenu objektovog dataseta. Ponekad je neophodno onemogućiti tu interakciju. Da bi to uradili koristimo *disable\_controls* i *enable\_controls* metode respektivno.

[Data aware controls](https://youtu.be/fMTq8P4XdGw)

## Programiranje podataka

### Dataset

Jam.py okvir koristi dataset koncept koji je vrlo blizu koncepta *Embercadero Delphi* dataseta.

Napomena: Mogući su i drugi načini čitanja i modifikovanja db podataka. Možete koristiti *connect* metod taska da dobijete konekciju iz konekcionog pula i koristiti konekciju da dobijete pristup bazi podataka koristeći Python Database API.

Svi objekti sa *item\_type* = *item* ili *table* isto kao i njihovi detalji (vidi Task stablo) mogu pristupiti podacima iz pridruženih tabela iz projektne baze podataka i pisati promene unutra. Svi oni su objekti *Item* klase:

* Item class (na klijentu)
* Item class (na serveru)

Obe klase imaju iste atribute, metode i evente pridružene upravljanju podacima. Da bi dobili dataset iz tabele projektne baze podataka, koristimo *open* metod. Ovaj metod, baziran na parametima, generiše sql upit za dobijanje dataseta. Pošto je dataset otvoren, aplikacija može navigovati, promeniti njegove zapise ili insertovati nove i pisati promene u objektovu tabelu baze podataka. Na primer, sledeće funkcije će postaviti *support\_rep\_id* vrednost polja na vrednost *id* polja na klijentu i na serveru respektivno:

function set\_support\_id(customers) {

customers.open();

while (!customers.eof()) {

customers.edit();

customers.support\_rep\_id.value = customers.id.value;

customers.post();

customers.next();

}

customers.apply();

}

def set\_support\_id(customers):

customers.open()

while not customers.eof():

customers.edit()

customers.support\_rep\_id.value = customers.id.value

customers.post()

customers.next()

customers.apply()

Ove funkcije uzimaju Customers objekat kao parametar. Posle toga, *open* metod je korišćen za dobijanje liste zapisa iz customers tabele i svaki zapis je modifikovan. Na kraju, promene su sačuvane u tabeli baze podataka korišćenjem *apply* metod.

Napomena: Postoji kraći put za navigaciju dataseta. Na primer, u pajtonu, sledeće petlje su ekvivalentne:

while not customers.eof():

print customers.firstname.value

customers.next()

for c in customers:

print c.firstname.value

[Datasets](https://youtu.be/gHTYj7h9ljI) i [Datasets Part 2](https://youtu.be/1bUGmgBfrNw) demonstriraju skoro sve metode rada sa datasetovima na spec. primerima.

### Navigacija dataseta

Svaki aktivni dataset ima kursor ili pokazivač, na trenutni zapis u datasetu. Trenutni zapis u datasetu je onaj čijim se vrednostima može manipulisati *edit*, *insert* i *delete* metodama, i onaj čije se vrednosti polja trenutno prikazuju u kontrolama osetljivm na podatke. Trenutni zapis možete da promenite pomeranjem kursora da biste ukazali na drugi. Sledeća tabela lista metode koje možete koristiti u aplikacionom kodu da biste prešli na različite zapise:

|  |  |  |
| --- | --- | --- |
| **Klijent metod** | **Server metod** | **Opis** |
| first | First | Pomera kurzor na prvi zapis u objektovom datasetu. |
| last | Last | Pomera kurzor na poslednji zapis u objektovom datasetu. |
| next | Next | Pomera kurzor na sledeći zapis u objektovom datasetu. |
| prior | Prior | Pomera kursor na prethodni zapis u objektovom datasetu. |

Kao dodatak ovim metodama, sledeća tabela opisuje dve metode koje obezbedjuju korisne informacije kada iterirate kroz zapise dataseta:

|  |  |  |
| --- | --- | --- |
| **Klijent metod** | **Server metod** | **Opis** |
| Bof | Bof | Ako metoda vraća *true*, kurzor je na prvom zapisu dataseta, inače kurzor nije na prvom zapisu dataseta. |
| Eof | Eof | Ako metoda vraća *true*, kurzor je na poslednjem zapisu dataseta, inače kurzor nije na poslednjem zapisu dataseta. |

Svaki put kada se kurzor pomeri na drugi zapisu datasetu sledeći eventi su pokrenuti:

|  |  |  |
| --- | --- | --- |
| **Klijent metod** | **Server metod** | **Opis** |
| on\_before\_scroll | on\_before\_scroll | Pojavljuje se pre nego što aplikacija skroluje sa jednog na drugi zapis. |
| on\_after\_scroll | on\_after\_scroll | Pojavljuje se pošto aplikacija skroluje sa jednog zapisa na drugi zapis. |

Korišćenjem ovih metoda možemo iterirati dataset. Na primer, na klijentu:

function get\_customers(customers) {

customers.open();

while (!customers.eof()) {

console.log(customers.firstname.value, customers.lastname.value);

customers.next();

}

}

Na serveru:

def get\_customers(customers):

customers.open()

while not customers.eof():

print customers.firstname.value, customers.lastname.value

customers.next()

#### Kraći put za iteriranje dataseta

Imamo *each* metod na klijentu koji možemo koristiti za iteriranje dataseta:

function get\_customers(customers) {

customers.open();

customers.each(function(c) {

if (c.rec\_no === 10) {

return false;

}

console.log(c.rec\_no, c.firstname.value, c.lastname.value);

});

}

Na serveru dataset možemo iterirati na sledeći način:

def get\_customers(customers):

customers.open()

for c in customers:

if c.rec\_no == 10:

break

print c.firstname.value, c.lastname.value

Obe funkcije će imati za izlaz korisnička imena prvih 10 zapisa dataseta. U oba slučaja c i customers su pokazivači na isti objekat.

### Modifikovanje dataseta

Kada aplikacija otvara objektov dataset, dataset automatski ulazi u *browse* stanje. Ovo stanje dataseta omogućava vam da pregledate zapise u datasetu, ali ne možete da uređujete ili unosite nove zapise. Uglavnom koristite *browse* stanje da skrolujete sa zapisa na zapis dataseta. Iz browse stanja mogu se postaviti sva ostala stanja dataseta. Na primer, pozivanjem *insert* ili *append* metoda dataset menja stanje iz *browse* u *insert*.

Dve metode mogu vratiti dataset u *browse* stanje. *Cancel* otkazuje trenutnu promenu ili insert zapisa i vraća dataset u *browse* stanje. *Post* upisuje promene u dataset i ako je uspešan, vraća dataset u *browse* stanje. Ako nije uspešan, trenutno stanje ostaje nepromenjeno. Da biste proverili objektov dataset status upotrebite *item\_state* atribut ili *is\_new*, *is\_edited* ili *is\_changing* metode:

|  |  |  |
| --- | --- | --- |
| **Klijent** | **Server** | **Opis** |
| item\_state | item\_state | Indicira trenutno operativno stanje objektovog dataseta. |
| is\_new | is\_new | Vraća *true* ako je objektov dataset u insert stanju. |
| is\_edited | is\_edited | Vraća *true* ako je objektov dataset u edit stanju. |
| is\_changing | is\_changing | Vraća *true* ako je dataset u insert ili edit stanju. |

Možete koristite sledeće metode da uradite *insert*, *update*, i *delete* podataka dataseta:

|  |  |  |
| --- | --- | --- |
| **Klijent** | **Server** | **Opis** |
| edit | edit | Postavlja objektov dataset u edit stanje. |
| append | apend | Dodaje zapis na kraj dataseta, i postavlja datast u insert stanje. |
| insert | insert | Insertuje zapis na početak dataseta i postavlja dataset u insert stanje. |
| post | post | Upisuje novi ili promenjeni zapis, i postavlja dataset u browse stanje. |
| cancel | cancel | Odustaje od tekuće operacije i postavlja dataset u browse stanje. |
| delete | delete | Briše tekući zapis i postavlja datset u browse stanje. |

Sve promene napravljene na datasetu su smeštene *post* metodom u change log memoriju. Posle svih promena koje su napravljene, one mogu biti smeštene u povezane db tabele pozivanjem *apply* metoda. *apply* metod generiše i izvršava sql upit da sačuva promene u bazi podataka.

|  |  |  |
| --- | --- | --- |
| **Klijent** | **Server** | **Opis** |
| log\_changes | log\_changes | Indicira da li trajno upisuje change log podataka u bazu podataka. |
| apply | apply | Šalje promenjene, insertovane i obrisane zapise iz objektovog dataseta na server za unos u bazu podataka. |

### Polja

Svi objekti, koji rade sa tabelama baze podataka imaju *fields* atribut - listu objekata polja, koji se koriste za predstavljanje zapisa tabele objekta. Svako polje ima sledeće atribute:

|  |  |  |
| --- | --- | --- |
| **Klijent** | **Server** | **Opis** |
| owner | owner | Objekat koji je vlasnik polja. |
| field\_name | field\_name | Ime polja koje se koristi u kodu. |
| field\_caption | field\_caption | Ime polja koje se pojavljuje korisniku. |
| field\_type | field\_type | Tip polja, jedan od: *text*, *integer*, *float*, *currency*, *date*, *datetime*, *boolean*, *blob*. |
| field\_size | field\_size | Veličina polja tipa *text*. |
| required | required | Spec. da je nonblank vrednost za polje zahtevana. |

Da bi pristupili podacima dataseta, *Field* klasa ima sledeća svojstva:

|  |  |  |
| --- | --- | --- |
| **Klijent** | **Server** | **Opis** |
| value | value | Koristite da bi dobili ili postavili vrednosti polja. Kada čita, vrednost polja je konvertovana u tip polja. Ako je polje tipa integer, float i currency, i ako je vrednost polja u zapisu tabele baze podataka NULL, vrednost svojstva je 0. Da bi dobili nekonvertovanu vrednost koristite *raw\_value* svojstvo polja. |
| text | text | Koristite da bi dobili ili postavili vrednost polja kao tekst. |
| lookup\_value | lookup\_value | Koristite da bi dobili ili postavili lookup vrednost polja. |
| lookup\_text | lookup\_text | Koristite da bi dobili ili postavili lookup vrednost polja kao tekst. |
| display\_text | display\_text | Predstavlja vrednost polja kako je prikazana u kontrolama osetljivim na podatke. Kada je polje lookup njegova vrednost je *lookup\_text*, inače to je *text* vrednost, sa respektom projektnih locale parametra. Ovo ponašanje može biti nadjačano sa *on\_field\_get\_text* event hendlerom objekta koji je vlasnik polja. |
| raw\_value | raw\_value | Koristite da bi dobili vrednost polja kao što je smešten u bazu podataka. Konverzija se ne koristi. |

Pored toga, svako polje je atribut objekta koji ga poseduje. Dakle, da biste dobili pristup polju objekta, koristite sledeću sintaksu: *item.field\_name*. invoices.total je referenca na total polje objekta invoices i invoices.total.value je vrednost ovog polja. Ispod su vrednosti atributa polja objekta invoices u demo projektu:

customer integer

value: 2

text: 2

lookup\_value: Köhler

lookup\_text: Köhler

display\_text: Leonie Köhler

firstname integer

value: 2

text: 2

lookup\_value: Leonie

lookup\_text: Leonie

display\_text: Leonie

billing\_address integer

value: 2

text: 2

lookup\_value: Theodor-Heuss-Straße 34

lookup\_text: Theodor-Heuss-Straße 34

display\_text: Theodor-Heuss-Straße 34

id integer

value: 1

text: 1

lookup\_value: None

lookup\_text:

display\_text: 1

date date

value: 2014-01-01

text: 01/01/2014

lookup\_value: None

lookup\_text:

display\_text: 01/01/2014

total currency

value: 2.08

text: $2.08

lookup\_value: None

lookup\_text:

display\_text: $2.08

#### Zajednička polja

Objekti koji imaju pristup podacima baze podataka mogu imati zajednička polja. Ona su definisana u *grupi* kojoj pripadaju. Ovde se definišu dva polja: *id* i *deleted*. *id* polje je postavljeno kao *primarni* *ključ* i čuvaće jedinstveni ident. za svaki zapis u tabeli baze podataka. Ovu vrednost automatski generiše okvir kada se insertuje novi zapis u tabelu. *Deleted* polje je postavljeno kao fleg izbrisanosti zapisa. Kada je *Soft delete* čekboks postavljen u Item Editor Dialog-u, *delete* metod ne briše fizički zapis iz tabele, već koristi ovo polje da bi označio zapis kao obrisan. *Open* metod to uzima u obzir kada stvara sql upit da dobije zapise iz tabele baze podataka.

Za grupu detalja mogu se definisati još dva zajednička polja - *master\_id* i *master\_rec\_id*. Koriste se za povezivanje zapisa objekta detalja na zapis u master tabeli.

### Lookup polja

Lookup polje prikazuje za korisnika prihvatljiviju vrednost izabane lookup vrednosti. Na primer, lookup polje može prikazati ime kupca koje je vezano za izabrani broj kupaca iz druge tabele ili liste. Kada unosi vrednost u lookup polje, korisnik bira sa liste vrednosti. Ovo može učiniti unos podataka bržim i tačnijim. Dve vrste lookup polja koje možete da kreirate su:

* lookup polje na osnovu lookup objekta i
* lookup polje na osnovu lookup liste.

#### Lookup polje zasnovano na lookup objektu

U okviru možete dodati lookup polje objektu da biste potražili informacije u tabeli drugog objekta. Na primer, u katalogu Album Demo applikacije nalazi se polje za unos/pretragu kataloga Artist. Da bi podesio vrednost polja korisnik mora kliknuti na dugme desno od polja unosa i izabrati zapis iz kataloga Artist koji će se pojaviti. Tada će vrednost ovog polja biti *id* izabranog zapisa lookup objekta - Artist. Samo polje će prikazivati vrednost lookup polja – *artists.name*. Drugi način da postavite vrednost polja je da se koristi *typehead*, ako je *Typehead* atribut je postavljen u Field Editor Dialog-u. Za takva polja lookup objekat i lookup polje mora biti postavljeno u Field Editor Dialog.

Sql upit koji se generiše na serveru, kada je pozvan *open* metod i *expanded* parametar je postavljen na *true* (podrazumevano), koristi *JOIN* klauzulu da biste dobili lookup vrednosti takvih polja. Stoga svako takvo polje ima par vrednosti: prva vrednost čuva referencu na *id* zapisa tabele lookup objekta (vrednost njegovog polja primarnog ključa), a druga vrednost je *vrednost lookup polja* u ovom zapisu. Da biste dobili pristup ovim vrednostima, koristite sledeća svojstva lookup polja:

|  |  |  |
| --- | --- | --- |
| **Klijent** | **Server** | **Opis** |
| Value | value | Vrednost smeštena u tabeli objekta, koja referencira zapis u tabeli lookup objekta. |
| lookup\_value | lookup\_value | Vrednost lookup polja u tabeli lookup objekta. |

Ponekad postoji potreba da ima dve ili više vrednosti iz istog zapisa tabele lookup objekta. Na primer, Invoices journal na fakturi u Demo projektu ima nekoliko lookup polja (kupac, adresa, grad, i tako dalje) koji imaju informacije o kupcu, sve smeštene u jednom zapisu u kupacima. Da bi se izbeglo stvaranje nepotrebnih polja u tabeli Invoices objekta, smeštanjem iste reference na zapis u lookup objektu i kreiranje *JOIN* za svako takvo polje, sva lookup polja osim Customers imaju *master field* vrednost koja ukazuje na polje Customers. Ova polja nemaju odgovarajuća polja u tabeli Invoices objekta. Njihova vrednost je uvek jednaka vrednosti *master field* i sql upit koji se generiše na serveru, kada se poziva *open* metod, koristi jednu *JOIN* klauzulu za sva ova polja.

Kada korisnik klikne na dugme na desno od lookup polja ili koristi *typeahead*, aplikacija stvara kopiju lookup objekta vezanog za lookup polje, postavlja svoje *lookup\_field* atribute i pokreće *on\_field\_select\_value* event hendler. Napišite ovaj event hendler da odredite polja lookup objekta koja će se prikazati, da postavite filtere lookup objekta, pre nego što će se otvoriti i prikazati korisniku da odabere lookup vrednost.

Lookup polje u lookup objektu takodje može biti lookup polje. Dakle da bi ste postavili lookup lookup-a koristite atribute lookup polje 2 i lookup polje 3.

#### Lookup polje zasnovano na lookup listi

Ponekad izvor za lookup polje može biti definisan kao lookup lista. Na primer, MediaType polje u Tracks katalogu Demo projekta ima lookup list atribut postavljen na MediaTypes lookup listu. Koristite Lookup List Dialog taska da definišete takve lookup liste.

### Filtriranje zapisa

Postoje tri načina za definiciju koje zapise u objektovom datasetu dobiti iz tabele baze podataka kada je open metod pozvan:

* spec. where parametra open metoda,
* pozivom set\_where metoda, pre poziva open metoda,
* korišćenjem filter-a.

Kada je *where* parametar spec., on se uvek koristi čak iako je *set\_where* metod bio pozvan ili objekat ima postavljene *filtere*. Kada je *where* parametar izostavljen parametar prosledjen *set\_where* metodu se koristi.

Na primer, na klijentu u sledećem kodu u prvom pozivu *open* metoda biće korišćena *where* opcija za filtritanje zapisa, u drugom pozivu koriste se prosledjeni parametri *set*\_*where* metodu i only the third time the value of invoicedate1 *filter* will be used:

function test(invoices) {

var date = new Date(new Date().setYear(new Date().getFullYear() - 1));

invoices.clear\_filters();

invoices.filters.invoicedate1.value = date;

invoices.open({where: {invoicedate\_\_ge: date}});

invoices.set\_where({invoicedate\_\_ge: date});

invoices.open();

invoices.open();

}

The same code on the server looks the following way:

from datetime import datetime

def test(invoices):

date = datetime.now()

date = date.replace(year=date.year-1)

invoices.clear\_filters()

invoices.filters.invoicedate1.value = date

invoices.open(where={'invoicedate\_\_ge': date})

invoices.set\_where(invoicedate\_\_ge=date)

invoices.open()

invoices.open()

U okviru, sledeći simboli i odgovarajuće konstante su definisane za filtriranje zapisa:

|  |  |  |  |
| --- | --- | --- | --- |
| **Filter type** | **Filter symbol** | **Constant** | **SQL Operator** |
| EQ | eq | FILTER\_EQ | = |
| NE | ee | FILTER\_NE | <> |
| LT | lt | FILTER\_LT | < |
| LE | le | FILTER\_LE | <= |
| GT | gt | FILTER\_GT | > |
| GE | ge | FILTER\_GE | >= |
| IN | in | FILTER\_IN | IN |
| NOT IN | not\_in | FILTER\_NOT\_IN | NOT IN |
| RANGE | range | FILTER\_RANGE | BETWEEN |
| ISNULL | isnull | FILTER\_ISNULL | IS NULL |
| EXACT | exact | FILTER\_EXACT | = |
| CONTAINS | contains | FILTER\_CONTAINS | koristi LIKE sa *%string%* da nadje zapise čija vrednost polja sadrži traženi stringom |
| STARTWITH | startwith | FILTER\_STARTWITH | koristi LIKE sa *string%* da nadje zapise čija vrednost polja počinje sa traženim stringom |
| ENDWITH | endwith | FILTER\_ENDWITH | koristi LIKE sa *%string* da nadje zapise čija vrednost polja završava sa traženim stringom |
| CONTAINS ALL | contains\_all | FILTER\_CONTAINS\_ALL | Koristi LIKE sa *%str1% AND %str2%* da nadje zapise čija vrednost polja sadrži sve reči traženog stringa |

*where* parametar *open* metoda je rečnik, čiji ključevi su imena polja koja su praćena, posle duple podcrte, sa filterskim simbolom. Za EQ filter simbol filtriranja \_\_eq može biti izostavljen. Na primer {'id': 100} je ekvivalentno kao {'id\_\_eq': 100}.

#### Filteri

Za svaki objekat koji ima pristup tabeli baze podataka lista filtera može biti kreirana. Za kreiranje filtera koristite *Filters Dialog AppBuilder*-a. Filteri omogućuju pogodan način da korisnici vizuelno određuju parametre filtera.

Svaki filter ima sledeće atribute:

* owner - objekat koji je vlasnik filtera,
* filter\_name - ime filteara koje može biti korišćeno u kodu
* filter\_caption - ime filtera koje se koristi za vizuelno predstavljanje filtera,
* filter\_type - tip filtera,
* visible - ako je *true*, vizuelno predstavljanje filtera će biti kreirano *create\_filter\_inputs* metodom, kada *filter* opcija *open* metoda nije spec.,
* value – vrednost filtera,

Svi filteri objekta su atributi *filters* atributa objekta. Korišćenjem *filter\_name* možemo pristupiti filter objektu:

invoices.filters.invoicedate1.value = new Date()

Drugi način da pristupimo filteru je korišćenje *filter\_by\_name* metoda:

invoices.filter\_by\_name('invoicedate').value = new Date()

### Detalji

Detalji se koriste u okviru za rad sa tabelarnim podacima. Na primer, Invoices journal u Demo aplikaciji, ima InvoiceTable detalj, koji sadrži listu pesama u fakturisanu kupcu.

Detalji i objekti detalja dele istu tabelu baze podataka.

Da biste kreirali detalj, prvo morate da kreirate detalj objekat (izaberite Details Group projekta i kliknite na New dugme), a zatim koristite Detail Dialog (izaberite objekat u stablu projekta i kliknite na dugme Detalji) da biste dodali detalje na objekat. Na primer sledeći kod:

def on\_created(task):

task.invoice\_table.open()

print task.invoice\_table.record\_count()

task.invoices.open(limit=1)

task.invoices.invoice\_table.open()

print task.invoices.invoice\_table.record\_count()

će štampati:

2259

6

Detalj imaju dva zajednička polja - *master\_id* i *master\_rec\_id*, koji se koriste za smeštanje informacija o *id*-u *master* *objekta* (svaki objekat ima svoj jedinstveni *id*) i vrednost primarnog ključa zapisa *master* objekta. Na taj način svaki objekat detalj može biti povezan na nekoliko master objekata. Isto tako svaki objekat može imati nekoliko objekata detalja. Da bi dobili pristup detaljima objekta koristi *details* atribut. Da bi dobili pristup master objektu koristite *master* atribut. Detail klasa, korišćena za kreiranje detalja, je naslednik *Item* klase i nasledjuje sve njene atribute, metode i evente.

Napomena: *apply* metod *Detail* klase ne radi ništa. Da bi upisali promene napravljene na detalju koristite apply metod master objekta.

Da bi radili sa detaljima master objekat mora biti aktivan. Da bi napravili promene na detaljima master objekat mora biti u *edit* ili *insert* stanju.

*Primeri*

U ovom primeru iz klijent modula Invoices objekat Demo project, Invoice\_table detalj je ponovo otvoren svaki put kada kursor master objekta se premesti na drugi zapis.

var ScrollTimeOut;

function on\_after\_scroll(item) {

clearTimeout(ScrollTimeOut);

ScrollTimeOut = setTimeout(

function() {

item.invoice\_table.open(function() {});

},

100

);

}

I kao primer:

from datetime import datetime, timedelta

def on\_created(task):

invoices = task.invoices.copy()

invoices.set\_where(invoicedate\_\_gt=datetime.now()-timedelta(days=1))

invoices.open()

for i in invoices:

i.invoice\_table.open()

i.edit()

for t in i.invoice\_table:

t.edit()

t.sales\_id.value = '101010'

t.post()

i.post()

invoices.apply()

Isti kod na klijentu:

function on\_page\_loaded(task) {

var date = new Date(),

invoices = task.invoices.copy();

invoices.set\_where({invoicedate\_\_gt: date.setDate(date.getDate() - 1)});

invoices.open();

invoices.each(function(i) {

i.invoice\_table.open();

i.edit();

i.invoice\_table.each(function(t) {

t.edit();

t.sales\_id.value = '101010';

t.post();

});

i.post();

});

invoices.apply();

}

## Programiranje na strani servera

Najčešće, klijent šalje zahtev serveru kada prateće metode objekta su izvršene:

* open
* apply
* print
* server

U tim slučajevima klijent šalje serveru:

* ID taska,
* ID objekta,
* type zahteva i
* njegovi parametri.

Kada primi zahtev, baziran na prosledjenim parametrima, server pronalazi task (može biti Project task ili AppBuilder task) i objekat na serveru, izvršava odgovarajući metod sa prosledjenim parametrima i vraća rezultat izvršavanja na klijent. *server* metod može pokrenuti evente koji mogu modifikovati njegovo podrazumevano ponašanje.

Svaki objekat task stabla ima *environ* i *session* atribute koji sadrže kontekst trenutnog zahteva. Najčešći serverski eventi su:

* on\_created – Event je pokrenut od strane taska kada je kreiran od serverske aplikacije. Može biti korišćeno za inicijalizaciju projekta.
* on\_apply events – Ovi eventi su pokrenuti kada je *apply* metod objekta pozvan na klijentu ili serveru.
* on\_open events – Ovi eventi su pokrenuti kada je *open* metod objekta pozvan na klijentu ili serveru.
* on\_generate – Event je pokrenut kada je print metod report objekta pozvan na klijentu.

Napomena: Primetimo da je objekat task stabla na serveru nepromenljiv, ne možemo promeniti atribute objekata u task stablu. Morate koristiti *copy* metod da kreirate kopiju objekta. Ova kopija je tačna kopija objekta u vreme izgradnje task stabla. Kopija se ne dodaje u task stablo i biće uništena Python skupljačem smeća kada više nije potrebna.

##### on\_apply eventi

Kada je *apply* metod objekta pozvan na klijentu ili serveru, serverska aplikacija, podrazumevano, generiše sql upit, zasnovan na datasetu i izvrši ga. Ovo ponašanje može biti promenjen pisanjem *on\_apply* event hendler u objektovom server modulu.

Ponekad postaje neophodno izvršiti neki kod, kada se promene upisuju, za sve objekte. U tom slučaju *on\_apply* event handler taska (deklarisan u task server modulu) može biti korišćen.

Sledeći kod opisuje kako se tim eventima upravlja:

#...

result = None

if self.task.on\_apply:

result = self.task.on\_apply(self, delta, params, connection)

if result is None and self.on\_apply:

result = self.on\_apply(self, delta, params, connection)

if result is None:

result = self.apply\_delta(delta, params, connection)

#...

return result

Ovaj kod proverava da li *task* ima *on\_apply* event hendler. Ako ga ima, izvršava ga. Ako *on\_apply* event hendler nije deklarisan za *task* server objekat ili je vratio *None*, metod proverava da li objekat ima *on*\_*apply* event hendler. Ako je definisan u server modulu objekta, izvršava ga. Ako je vraćeni rezultat *None* *apply*\_*delta* metod objekta je pozvan da generiše sql upit, izvršava ga i vraća rezultat.

*Primeri*

Ovde je primer kako *on\_apply* može biti korišćen: [Kako implementirati osnovni multi-tenancy?](https://jam-py.com/docs/how_to/how_to_multitenancy.html)

##### on\_open eventi

Kada je *open* metod objekta pozvan na klijentu ili serveru, server aplikacija izvršava sledeći kod:

result = None

if self.task.on\_open:

result = self.task.on\_open(self, params)

if result is None and self.on\_open:

result = self.on\_open(self, params)

if result is None:

result = self.execute\_open(params)

Ovaj kod provera da li *task* ima *on*\_*open* event handler. Ako ga ima, izvršava ga. Ako *on\_open* event handler nije deklarisan za *task* server objekat ili je vratio *None*, metod proverava da li objekta ima *on\_open* event handler. Ako je definisan u server modulu objekta, izvršava ga. Ako je vraćeni rezultat *None*, *execute\_open* metod objekta je pozvan da generiše sql query, izvršava ga i vraća rezultat.

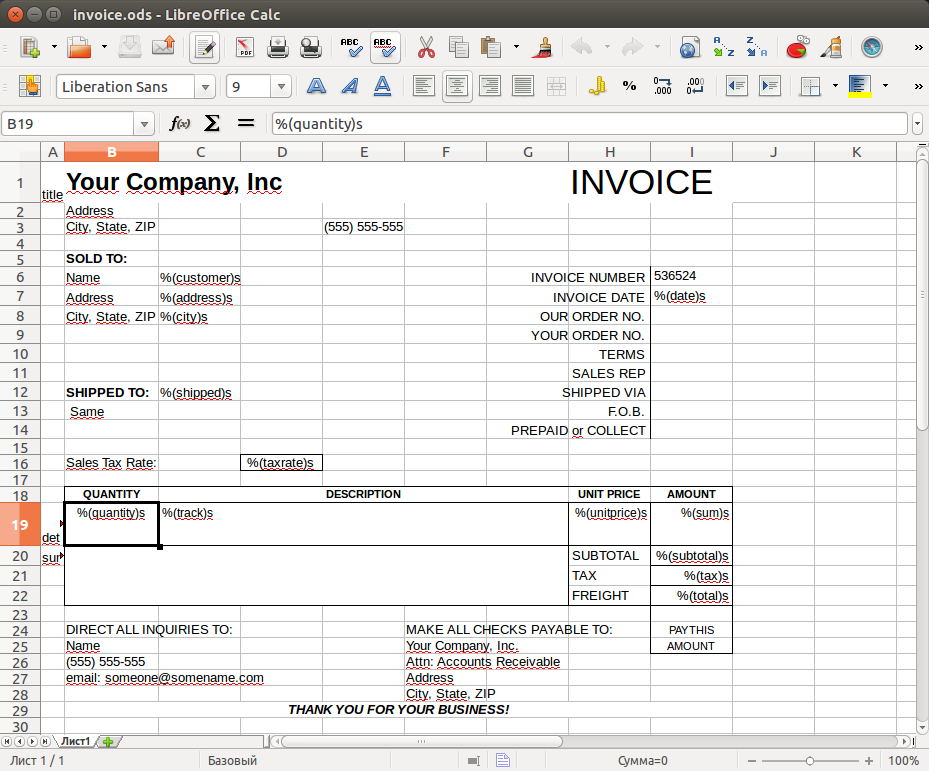
*Primeri*

Ovde je primer kako *on\_open* može biti korišćen: [Kako implementirati osnovni multi-tenancy?](https://jam-py.com/docs/how_to/how_to_multitenancy.html)

## Programiranje izveštaja

### Šabloni izveštaja

Da kreirate izveštaj, prvo morate pripremiti šablon izveštaja u LibreOffice Calc-u. Fajlovi šablona nalaze se u folderu reports projekta. Sledeća slika prikazuje šablon Invoice izveštaja.



Izveštaji u Jam.py su band orijentisani. Svaki šablon izveštaja podeljen je na bandove. Da biste postavili band koristite krajnje levu kolonu šablona. U šablonu izveštaja Invoice postoje tri banda: title, detail i summary.

Pored toga, šabloni mogu da imaju programibilne ćelije. Na primer, u šablonu izveštaja Invoice ćelija *i7* sadrži tekst *%(date)s*. Programabilna ćelija započinje sa *%*, a zatim sledi ime varijable u zagradama koju sledi karakter *s*.

### Kreiranje izveštaja

Da dodate izveštaj u Jam.py projekat, izaberite *Reports group* u projektnom stablu, kliknite na *New* dugme i popunite *caption*, *name* i *name of template file* izveštaja. Ako je čekboks *visible* postavljen, podrazumevani kod dodaje izveštaj u Reports meni projekta.

### Parametri izveštaja

Možete spec. prametre izveštaja. Na primer, Customerpurchases report Demo projekta ima tri parametra. Da dodate ili promenite parametar izveštaja kliknite na Reportparams dugme u desnom panelu AppBuilder-a. Forma sa prikazanom listom postojećih parametara će se pojaviti. Tada klikom na *New* ili *Edit* dugme forme možete dodati ili promeniti parametar.

U dijalog boksu popunite:

* caption – ime parametra koje se pojavljuje korisnicima
* name – ime parametra koje će se koristiti u kodu
* type – tip podatka parametra
* visible – klijent aplikacija kreira fornu za spec. parametara pre štampe izveštaja. Ako je ovaj čekboks postavljen, ulazni element za dati parametar će se pojaviti na formi.
* required – ako je čekboks postavljen i *visible* je postavljen, klijentska aplikacija će zahtevati da korisnik unese parametar pre štampe izveštaja.
* align – spec. kako je vrednost parametra poravnata unutar ulaznog elementa.

Možete kreirati lookup parametar, na primer, Customerpurchases report ima Customer parametar čija vrednost može biti izabrana iz Customers kataloga. U ovom slučaju trebalo bi da postavite:

* Lookupitem – lookup objekat,
* Lookupfield – lookup polje.

### Programiranje izveštaja na klijentu

Za štampanje izveštaja na klijentu koristite *print* metod. Kao rezultat pozivanja *print* metoda, klijent poziva *create\_param\_form* metod za kreiranje forme za editovanje parametara izveštaja, zasnovanog na *html* šablonu definisanom u *index*.*html* fajlu. Ovaj metod, posle kreiranja forme, pokreće sledeće event hendlere:

* on\_param\_form\_created taska,
* on\_param\_form\_created report grupe koja je vlasnik izveštaja, ako je definisan,
* on\_param\_form\_created izveštaja, ako je definisan.

Podrazumevani kod je u *on\_param\_form\_created* event hendleru, definisan na tasku. U ovom eventu, klik na *Print* dugme je pokreće *process\_report* metod izveštaja.

function on\_param\_form\_created(item) {

item.create\_param\_inputs(item.param\_form.find(.edit-body));

item.param\_form.find(#ok-btn).on('click.task', function() {

item.process\_report()

});

item.param\_form.find(#cancel-btn).on('click.task', function() {

item.close\_param\_form()

});

}

Nadalje se pokreću sledeći event hendleri:

* on\_before\_print\_report event handler report grupe
* on\_before\_print\_report event handler izveštaja

U njima programer može definisati neke zajedničke (event hendler report grupe) ili spec. (event hendler izveštaja) atribute izveštaja. Na primer, u podrazumevanom kodu, postoji *on\_before\_print\_report* event hendler report grupe, u kojem su definisane ekstenzije izveštaja:

function on\_before\_print\_report(report) {

var select;

report.extension = 'pdf';

if (report.param\_form) {

select = report.param\_form.find('select');

if (select && select.val()) {

report.extension = select.val();

}

}

}

U sledećem event hendleru, definisanom u klijent modulu Invoice izveštaja Demo aplikacije, vrednost *id* parametra je postavljena:

function on\_before\_print\_report(report) {

report.id.value = report.task.invoices.id.value;

}

Posle toga *process\_report* metod šalje asinhroni zahtev na server za generisanje izveštaja. Server vraća u metod jedan *url* fajla sa generisanim izveštajem. Metod potom proverava *on\_open\_report* event handler report grupe da li je definisan. Ako je taj event hendler definisan poziva ga, inače proverava *on\_open\_report* izveštaja. Ako je definisan poziva ga. Ako nijedan od eventa nije definisan, (zavisno od atributa *extension* izveštaja) otvara izveštaj u brauzeru i čuva ga na disk.

### Programiranje izveštaja na serveru

Kada server dobije zahtev sa klijenta da generiše izveštaj, on prvo kreira kopiju izveštaja i tada kopija poziva *generate* metod. Ovaj metod pokreće *on\_before\_generate* event. U ovom eventu programer bi trebao da piše kod koji generiše sadržaj izveštaja.

Na primer za Invoice izveštaj Demo aplikacije ovaj event je:

def on\_generate(report):

invoices = report.task.invoices.copy()

invoices.set\_where(id=report.id.value)

invoices.open()

customer = invoices.firstname.display\_text + ' ' + invoices.customer.display\_text

address = invoices.billing\_address.display\_text

city = invoices.billing\_city.display\_text + ' ' +

invoices.billing\_state.display\_text + ' ' + \

invoices.billing\_country.display\_text

date = invoices.invoicedate.display\_text

shipped = invoices.billing\_address.display\_text + ' ' +

invoices.billing\_city.display\_text + ' ' + \

invoices.billing\_state.display\_text + ' ' +

invoices.billing\_country.display\_text

taxrate = invoices.taxrate.display\_text

report.print\_band('title', locals())

tracks = invoices.invoice\_table

tracks.open()

for t in tracks:

quantity = t.quantity.display\_text

track = t.track.display\_text

unitprice = t.unitprice.display\_text

sum = t.amount.display\_text

report.print\_band('detail', locals())

subtotal = invoices.subtotal.display\_text

tax = invoices.tax.display\_text

total = invoices.total.display\_text

report.print\_band('summary', locals())

Prvo, koristimo *copy* metod da kreiramo kopiju Invoice journal objekta.

invoices = report.task.invoices.copy()

Kreiramo kopiju jer više korisnika može istovremeno pokrenuti isti izveštaj u paralelnim nitima. Potom pozivamo *set\_where* metod kopije:

invoices.set\_where(id=report.id.value)

gde je *report.id.value* je id parameta izveštaja, vrednost koju postavljamo u *on\_before\_print\_report* event hendlere na klijentu i koji je jednak tekućoj vrednosti id polja invoice journal-a. Potom koristeći *open* metod, dobijamo zapise na serveru. Posle toga *print\_band* metod se koristi za štampanje title banda:

report.print\_band('title', locals())

Zatim pridružimo vrednosti u četiri lokalne varijable: customer, address, city i date koje odgovaraju programibilnim ćelijama u title bandu u šablonu izveštaja. Potom na isti način definišemo lokalne varijable za detail i summary bandove. Kada je izveštaj generisan i vrednost atributa *extension* izveštaja postavljena na klijentu nije jednaka pdf, server konvertuje *ods* fajl koristeći LibreOffice. Pošto je izveštaj generisan on se smešta u report folder static foldera i server šalje klijentu url fajla izveštaja.

# Jam.py FAQ

## Koja je razlika izmedju kataloga i žurnala?

Kada je novi projekat kreiran, njegovo task drvo ima sledeće grupe: *Catalogs*, *Journals*, *Details* i *Reports*. *Catalogs* i *Journals* idu ispod grupe *Item* tipa i imaju istu funkcionalnu namenu. Mi smo ih kreirali da pravimo razliku izmedju dva tipa objeka podataka:

* Catalogs – objekti podataka koji sadrže informacije kataloškog tipa, kao korisnici, organizacije, pesme itd.
* Journals – objekti podataka koji smeštaju informacije o dogadjajima zapisanim u nekim dokumentima, kao što su fakture, nalozi za plaćanje itd.

## Kako podići već kreirani projekat na novu verziju jampy frejmvorka?

Ako je potrebno podići verziju paketa, možete to učiniti koristeći *pip*. Na Linux, Mac OS X ili nekom drugom kao Unix OS-u, izvršite komandu:

sudo pip install --upgrade jam.py

Ako koristite Windows, izvršite u komandnom šelu, sa admin privilegijama:

pip install --upgrade jam.py

## Šta su strani ključevi?

*Strane* *ključeve* (*foreign* *keys*) možete kreirati u AppBuilder. Oni sprečavaju brisanje zapisa lookup tabele ako je referenca na zapis smeštena u lookup polje.

Na primer, kada je strani ključ kreiran na Customer polju Invoices objekta, korisnik neće moći da obriše zapis u Customers katalogu ako je referenca na taj zapis smeštena u Invoices.

Pri tome *Soft* *delete* atribut lookup objekta mora biti postavljen na *false*.

## Mogu li koristiti druge biblioteke u mojoj aplikaciji?

Možete dodati javascript biblioteke. Najbolje mesto da ih smestite je *js* *folder* *static* *foldera* projekta. Referencirajte ih koristeći *src* atribut u <*script*> tagu *index*.*html* fajla.

Na primer, Demo projekat koristi Chart.js biblioteku za kreiranje dešborda:

<script src=/static/js/Chart.min.js></script>

Na strani servera možete importovati python biblioteku u vaše module.

Na primer za uvoz smtplib biblioteke za slanje mejlova:

import smtplib

## Kada štampam izveštaj dobijam ods umesto pdf fajla?

Kada se izveštaj generiše, serverska aplikacija prvo kreira *ods* file. Ako je *ext* atribut izveštaja postavljen na *pdf* ili bilo koji drugi format osim *ods*, aplikacija prvo kreira *ods* fajl a zatim koristi LibreOffice u *headless* modu da konvertuje *ods* fajl u dati foramt. Ako je LibreOffice trenutno pokrenut na serveru konverzija se možda neće desiti. Morate zatvoriti LibreOffice na serveru da bi konverzija uspela.

# Kako da …?

Ovde su korisni saveti i kodovi koje možete koristiti u vašim aplikacijama.

## Kako da migriram razvoj na produkciju?

Migriranje razvoja na produkciju je vrlo jednostavno u Jampy zbog mogućnosti exporta i importa njegovih meta podataka. Proces importovanja meta podataka zavisi od tipa baze podataka projekta.

### Migracija novog projekta

* Kreiraj praznu bazu podatka u produkcionoj okolini.
* Pokreni jam*-*project*.*py skript za kreiranje novog projekta.
* Podesi server.
* U brauzeru pokreni *AppBuilder* i završi kreiranje projekta sa praznom bazom podataka.
* Otvori *Parameters* dijalog za podešavanje projekta. Podesi parametre:
  + *Production* na tačno
  + *Safe* *mode*
  + *Debugging* na netačno
* Izvezi meta podatke razvojnog projekta u zip faj.
* Uvezi meta podatke u novi projekat.

Napomena: Za projekte sa *SQLite* bazom podataka možete jednostavno kopirati razvojni folder u produkcionu okolinu.

### Migracija postojećeg projekta

* Izvezi meta podatke razvojnog projekta u zip fajl.
* Uvezi podatke u produkcioni projekat.

Napomena: Za *SQLite* baze podataka, Jampy ne podržava uvoz meta podataka u postojeći projekat (projekat sa tabelama u bazi podataka). Možete importovati meta podatke samo u novi projekat.

### Uvoz meta podataka sa zaustavljanjem servera

* Zaustavi server i kopiraj zip fajl sa meta podacima u migracioni (*migration*) folder u projektnom folderu. Ako folder ne postoji kreiraj ga.
* Pokreni server.
* Web aplikacija, dok inicijalizuje sebe, će uvesti meta podatke. Možete videti kako je prošao uvoz meta podataka u log fajlu u *logs* folderu projektnog foldera. Ako je uvoz uspešan, zip fajl će biti obrisan.

### Uvoz meta podataka bez zaustavljanja servera

Klikni na Import dugme u *AppBuilder*.

Napomena: Podrazumevno, web aplikacija u u procesu uvoza meta podataka čeka 5 minuta ili sve dok svi prethodni zahtevi aplikaciji ne budu procesuirani, pre nego krene u promene baze podataka. Za projekte koji pokreću više procesa možete podesiti *Import**delay* parametar u *Parameters* dialogu projekta, ili koristiti *Importing metadata* sa zaustavljenim serverom.

## Kako migrirati na drugu bazu podataka?

Možete migrirati na drugu bazu podataka. Na primer, razvijali ste vaš projekat na *SQLite* bazi podataka, i želite da migrirate na *PostgreSQL*.

Da to uradite, pratite sledeće korake:

1. Kreiraj praznu *PostgreSQL* bazu podataka
2. Kreiraj novi projekat sa tom bazom podataka
3. Izvezi meta podatke iz *SQLite* projekta u zip fajl.
4. Uvezi meta podatke u *PostgreSQL* projekat. Web aplikacija će kreirati strukturu u *PostgreSQL* bazi podataka.
5. Kopiraj podatke iz *SQlite* u *PostgreSQL* bazu podataka koristeći *copy\_database* metod task objekta:
   * Kreiraj u serverskom modulu task objekta PostgreSQL projekta kod:

from jam.db.db\_modules import SQLITE

def copy\_db(task):

task.copy\_database(SQLITE, '/home/work/demo/demo.sqlite')

* + Tada izvršite funkciju *copy\_database* na jedan od sledećih načina:
    1. Pozovi gore kreiranu funkciju u *on\_created* event hendleru:

from jam.db.db\_modules import SQLITE

def copy\_db(task):

task.copy\_database(SQLITE, '/home/work/demo/demo.sqlite')

def on\_created(task):

copy\_db(task)

* + 1. Kreiraj CopyDB dugme u nekoj formi i koristi task server metod da bi izvršio kopiranje:

function on\_view\_form\_created(item) {

item.add\_view\_button('Copy DB').click(function() {

task.server('copy\_db')

});

}

* + 1. Pokreni iz debbuging konzole brauzera:

task.server('copy\_db')

1. Ukloni kod korišćen za kopiranje podataka.

Napomena: Ne možete migrirati u *SQLite* bazu podataka ako tekuća baza podataka ima strane ključeve.

## Kako da napišem funkciju vidljivu u globalnom opsegu?

Svaka funkcija definisana u server ili klijent modulu nekog objekta postaje njegov atribut. Koristeći *task* drvo, moguće je pristupiti bilo kojoj funkciji definisanoj u server ili klijent modulu iz bilo kog modula projekta.

Na primer, ako imamo funkciju *some\_func* deklarisanu u *Customers* klijent modulu, možemo je izvršiti u bilo kom modulu projekta. Primetite da je *task* globalna varijabla na klijentu:

task.customers.some\_func()

Na serveru, task nije globalni objekat, ali objekat koji okida/poziva je prosledjen u svaki event hendler i funkciju pozvanu od strane server metoda. Nadalje, ako je *some*\_*func* funkcija deklarisana u *Customers* server modulu, može biti izvršena u funkciji ili event hendleru kao:

def on\_apply(item, delta, params):

item.task.customers.some\_func()

Primetite da event hendler je samo funkcija i može biti pozvana iz drugih modula.

## Kako validirati vrednost polja?

Napiši *on\_field\_validate* event hendler da validiraš vrednost polja.

Na primer, event će biti okinut, kada je *post* metod pozvan, koji upisuje zapis u memoriju ili kada korisnik napušta polje koje je menjao:

function on\_field\_validate(field) {

if (field.field\_name === 'unitprice' && field.value <= 0) {

return 'Unit price must be greater that 0';

}

}

Ispod je kod koji ne koristi *on\_field\_validate* metod i proverava vrednost *unitprice* polja i sprečava da korisnik napusti polje kada je vrednost manja ili jednaka nuli:

function on\_edit\_form\_shown(item) {

item.each\_field( function(field) {

var input = item.edit\_form.find('input.' + field.field\_name);

input.blur( function(e) {

var err;

if ($(e.relatedTarget).attr('id') !== cancel-btn) {

err = check\_field\_value(field);

if (err) {

item.alert\_error(err);

input.focus();

}

}

});

});

}

function check\_field\_value(field) {

if (field.field\_name === 'album' && !field.value) {

return 'Album must be specified';

}

if (field.field\_name === 'unitprice' && field.value <= 0) {

return 'Unit price must be greater that 0';

}

}

Deklarisali smo *on\_edit\_form\_shown* event hendler u objektovom modulu, tako da će raditi samo u tom modulu. Možemo deklarisati sledeći event hendler u task klijent modulu tako da možemo pisati *check\_field\_value* fukciji u bilo kom modulu po potrebi da bi omogućili ovu validaciju polja.

*on\_edit\_form\_shown* task objekta se poziva prvo za svaki objekat kada je forma prikazana.

function on\_edit\_form\_shown(item) {

if (item.check\_field\_value) {

item.each\_field( function(field) {

var input = item.edit\_form.find('input.' + field.field\_name);

input.blur( function(e) {

var err;

if ($(e.relatedTarget).attr('id') !== cancel-btn) {

err = item.check\_field\_value(field);

if (err) {

item.alert\_error(err);

input.focus();

}

}

});

});

}

}

U ovom event hendleru proveravamo da li objekat ima *check\_field\_value* atribut. Svaka funkcija deklarisana u modulu objekta postaje njegov atribut.

## Kako dodati dugme na formu?

Najjednostavniji način za dodavanje dugmeta na edit/view formu je korišćenje *add\_edit\_button* / *add\_view\_button metoda* objekta. Možete pozvati ove funkcije u *on\_edit\_form\_created* / *on\_view\_form\_created event* hendlerima.

Na primer, Customers objekat koristi ovaj kod za dodavanje dugmeta na view formu:

function on\_view\_form\_created(item) {

item.table\_options.multiselect = false;

if (!item.lookup\_field) {

var print\_btn = item.add\_view\_button('Print', {image: 'icon-print'}),

email\_btn = item.add\_view\_button('Send email', {image: 'icon-pencil'});

email\_btn.click(function() { send\_email() });

print\_btn.click(function() { print(item) });

item.table\_options.multiselect = true;

}

}

U ovom kodu objektov *lookup\_field* atribut se proverava i ako nije definisan (view forma nije kreirana zbog izbora vrednosti za lookup polje) dva dugmeta se kreiraju i za njih JQuery klik eventi su pridruženi, koji pokreću *send\_email* i *print* funkcije deklarisane u ovom modulu.

## Kako izvršiti serverski skript sa klijenta

Možete koristiti server metod na klijentu da pošaljete ahtev na server za izvršenje funkcije definisane u serverskom modulu objekta.

Dat je primer dole. Kreiramo *btn* dugme koje je JQuery objekat. Tada koristimo klik metod da pridružimo funkciju koja poziva server metod objekta da pokrene *calculate* funkciju definisanu u serverskom modulu objekta. Kod u klijentskom modulu objekta:

function on\_view\_form\_created(item) {

var btn = item.add\_view\_button('Calculate', {type: 'primary'});

btn.click(function() {

item.server('calulate', [1, 2, 3], function(result, error) {

if (error) {

item.alert\_error(error);

}

else {

console.log(result);

}

})

});

}

Kod u serverskom modulu objekta:

def calculate(item, a, b, c):

return a + b + c

## Kako da promenimo stil i atribute elemenata forme?

Možete pristupiti bilo kom DOM elementu na formi koristeći Jquery. U sledećem primeru, u *on\_edit\_form\_created* event hndleru definisanom u klijent modulu mi nalazimo *OK* dugme, skrivamo ga i menjamo tekst *Cancel* dugmeta na Close u edit formi:

function on\_edit\_form\_created(item) {

item.edit\_form.find(#ok-btn).hide();

item.edit\_form.find(#cancel-btn).text('Close');

}

Kada aplikacija kreira ulazne kontrole, ona dodaje klasu sa imenom field\_name atributa odgovarajućeg polja na svaki ulaz. Zatim koristeći jQuery [selektore](https://www.w3schools.com/jquERY/jquery_ref_selectors.asp), možemo pronaći svako polje, za customer polje kao što sledi (mi biramo ulaz sa customer klasom u edit formi):

item.edit\_form.find(input.customer)

Kada smo pronašli element forme možemo koristiti JQuery metode da ga promenimo. Kako su ulazi za polja kreirani sa *create\_inputs,* *on\_edit\_form\_created* event je pokrenut i morate napisati *on\_edit\_form\_shown* event hendler za promenu ulaza. Na primer:

function on\_edit\_form\_shown(item) {

item.edit\_form.find('input.name').css('color', 'red');

item.edit\_form.find('input.name').css('font-size', '24px');

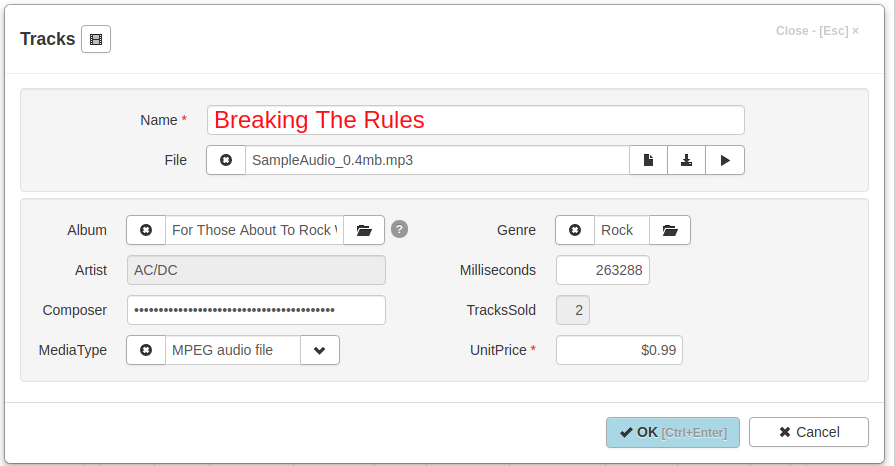
item.edit\_form.find('input.tracks\_sold').width(20);

item.edit\_form.find('input.genre').parent().width('40%');

item.edit\_form.find('input.composer').prop('type', 'password');

}

Kod će promeniti ulaze na ovaj način:



Primetite da ako treba da promenite širinu ulaza kontrole koja ima roditelja ( ulaz za datum, dateumvreme i lookup polja) postavite širinu roditelja:

item.edit\_form.find('input.album').parent().width('50%');

Drugi način da promenite stil DOM elemenata je korišćenje CSS. Kada je task nod selektovan u AppBuilder-u, projektno css dugme je locirano na desnom panelu. Klikom na njega otvara se project.css fajl, koji je lociran u projektnom folderu. Možete ga koristiti za CSS ulaze koje definišu stil DOM elemenata projekta.

Svaka forma objekta kreirana u projektu ima *css* klase koje omogućuju programeru da ident. formu. Svaka forma ima jednu klasu koja ident. tip forme: *view-form*, *edit-form*, *filter-form* or *param-form*. Na primer, sledeći kod će ukloniti slike sa dugmadi sa dna forme:

.view-form .form-footer .btn i {

display: none;

}

Više primera za edit forme:

.edit-form #ok-btn {

font-weight: bold;

background-color: lightblue;

}

.edit-form.invoices input.total {

color: red;

}

Takodje svaka forma ima klasu sa imenom *item\_name* atributa objekta. Sledeći kod će ukloniti slike sa dugmadi samo u Invoices *view* formi:

.view-form.invoices .form-footer .btn i {

display: none;

}

Možete promeniti način na koji su tabele prikazane. Tabele koje su kreirane sa create\_table metodom imaju *css* klasu *dbtable* i klasu sa imenom item\_name attibuta objekta. Svak kolon atabele takodje ima klasu sa imeno polja *field\_name* atributa odgovarajućeg polja. Primer, sledeći kod će prikazati ćelije u Invoices tabeli Customer kolona boldovano:

.dbtable.invoices td.customer {

font-weight: bold;

}

Još jedan način promene kako je kolona prikazana je pisanje on\_field\_get\_html event hendlera. Na primer:

function on\_field\_get\_html(field) {

if (field.field\_name === 'total') {

if (field.value > 10) {

return '<strong>' + field.display\_text + '</strong>';

}

}

}

## Kako dodati zapis koristeći edit formu bez otvaranja view forme?

Prvo morate pozvati *open* metod objekta da inicijalizuje dataset. Na primer, ako želite da dodate *novi* zapis u Invoices u Demo aplikaciji, možete uraditi sledeće:

var invoices = task.invoices.copy();

invoices.open({ open\_empty: true });

invoices.append\_record();

U ovom kodu, kreiramo kopiju objekta sa *copy* metodom tako da ova operacija ne utiče na Invoices view formu ako je otvorena u tabu.

Takodje možete promeniti zapis, ali pre nego što to uradite, morate ga dobiti sa servera. Ispod je kod koji modifikuje zapis sa *id = 411*. Proveravamo da li zapis postoji koristeći *rec\_count* svojstvo, inače prikazujemo upozorenje.

var invoices = task.invoices.copy();

invoices.open({ where: {id: 411} });

if (invoices.rec\_count) {

invoices.edit\_record();

}

else {

invoices.alert\_error('Invoices: record not found.');

}

U primeru gore open metod je izvršen sinhrono.

Kod ispod rati isto ali asinhrono:

var invoices = task.invoices.copy();

invoices.open({ where: {id: 411} }, function() {

if (invoices.rec\_count) {

invoices.edit\_record();

}

else {

invoices.alert\_error('Invoices: record not found.');

}

});

Invoices ima *modeless* atribut postavljen u Edit form dialogu, tako da će edit forma biti otvorena u tabu. Možete promeniti postavku *modeless* atributa *edit\_options* da napravite edit formu modalnom:

var invoices = task.invoices.copy();

invoices.edit\_options.modeless = false;

## Kako zabraniti promenu zapisa?

Pretpostavimo da imamo objekat sa jednim boolean poljem sa imenom *posted*, i ako je vrednost polja *true*, moramo zabraniti promenu ili brisanje zapisa. Možemo to uraditi pisanjem *on\_after\_scroll* event hendlera i koristeći *permissions* svojstvo:

function on\_after\_scroll(item) {

if (item.rec\_count) {

item.permissions.can\_edit = !item.posted.value;

item.permissions.can\_delete = !item.posted.value;

if (item.view\_form) {

item.view\_form.find(#delete-btn).prop(disabled, item.posted.value);

}

}

}

Takodje možemo pisati *on\_apply* event hendler u server modulu objekta:

def on\_apply(item, delta, params, connection):

for d in delta:

if d.posted.old\_value:

raise Exception('Document posted. No change allowed')

## Kako da povežem dva objekta?

Objasnićemo na primeru tracks i invoicetable objekata u demo aplikaciji. Linkovaćemo zapise tracks objekta sa odgovarajućom listom prodatih tracks iz invoicetable objekta. Podrazumeano ponašanje *view\_form* je definisano u *on\_view\_form\_created* event hendleru definisanom u task klijent modulu. Promenićemo ga u *on\_view\_form\_created* event hendleru u tracks klijent modulu:

function on\_view\_form\_created(item) {

item.table\_options.height -= 200;

item.invoice\_table = task.invoice\_table.copy();

item.invoice\_table.paginate = false;

item.invoice\_table.create\_table(item.view\_form.find('.view-detail'), {

height: 200,

summary\_fields: ['date', 'total'],

});

}

Ovde redukujemo visinu za prikaz tracks zapisa za 200 pixela

item.table\_options.height -= 200;

kreiramo kopiju invoice\_table, postavljamo *paginate* atribut na *false* i pozivamo *create\_table* metod da kreira tabelu koja će prikazati prodate tracks:

item.invoice\_table = task.invoice\_table.copy();

item.invoice\_table.paginate = false;

item.invoice\_table.create\_table(item.view\_form.find('.view-detail'), {

height: 200,

summary\_fields: ['date', 'total'],

});

Za tu tabelu postavljamo visinu od 200 pixela i definišemo summary polja. Ova tabela će uvek biti prazna ako ne želimo da definišemo sledeći *on\_after\_scroll* event hendler:

function on\_after\_scroll(item) {

if (item.view\_form.length) {

if (item.rec\_count) {

item.invoice\_table.set\_where({track: item.id.value});

item.invoice\_table.set\_order\_by(['-invoice\_date']);

item.invoice\_table.open(true);

}

else {

item.invoice\_table.close();

}

}

}

*on\_after\_scroll* event je pokrenut kada god je tekući zapis promenjen. Tako da kada je track promenjen mi pozivamo *open* metod, prepodešavajući filter i poredak:

item.invoice\_table.set\_where({track: item.id.value});

item.invoice\_table.set\_order\_by(['-invoice\_date']);

item.invoice\_table.open(true);

Ovaj metod šalje zahtev serveru, koji generiše sql upit, izvršava ga i vraća dataset koji sadrži prodate zapise tih trekova poredjenih u opadajućem poretku polja invoice\_date. Ako je tracks dataset prazan mi čistimo dataset prodatih zapisa pozivanjem *close* metoda. Pošto su kontrole Jam.py osetljive na podatke, svaka promena dataseta prodatih zapisa će biti prikazana u tabeli koju smo kreirali u *on\_view\_form\_created* event hendlere. Sada svaki put kada je track promenjen aplikacija šalje na server da obnovi prodate trekove. To nije efektivno i može voditi čekanju. Da sprečimo to koristimo JavaScript setTimeout funkciju:

var scroll\_timeout;

function on\_after\_scroll(item) {

if (!item.lookup\_field && item.view\_form.length) {

clearTimeout(scroll\_timeout);

scroll\_timeout = setTimeout(

function() {

if (item.rec\_count) {

item.invoice\_table.set\_where({track: item.id.value});

item.invoice\_table.set\_order\_by(['-invoice\_date']);

item.invoice\_table.open(true);

}

else {

item.invoice\_table.close();

}

},

100

);

}

}

Ova funkcija garantuje da će podaci biti apdejtovani ne više od jedan put svakih 100 milisec.

Pošto je invoicetable detalj i ima *master\_rec\_id* polje koje smešta referencu na invoice koji je vlasnik tog zapisa, možemo pokazati korisniku invoice koji sadrži tekući zapis prodaje. Da to uradimo prosledjujemo *create\_table* metodi funkciju koja će biti izvršena kada korisnik duplo klikne na zapis:

item.invoice\_table.create\_table(item.view\_form.find('.view-detail'), {

height: 200,

summary\_fields: ['date', 'total'],

on\_dblclick: function() {

show\_invoice(item.invoice\_table);

}

});

I definišimo funkciju:

function show\_invoice(invoice\_table) {

var invoices = task.invoices.copy();

invoices.set\_where({id: invoice\_table.master\_rec\_id.value});

invoices.open(function(i) {

i.edit\_options.modeless = false;

i.can\_modify = false;

i.invoice\_table.on\_after\_open = function(t) {

t.locate('id', invoice\_table.id.value);

};

i.edit\_record();

});

}

U ovoj funkciji kreiramo kopiju invoices journal-a i tražimo fakturu. Kada je open metod izvršen mi ćemo prikazati fakturu pozivom *edit\_record* metod. Ali pre pozivanja mi ćemo postaviti atribute tako da bude modalna forma i korisnik neće moći da da modifikuje. Pored toga mi dinamički dodeljujemo *on\_after\_open* event hendler da invoice\_table detalj dobijemo. U ovom event hendleru mi ćemo pronaći tekući zapis u zapisim prodatih pozivanjem *locate* metoda.

Na kraju proveramo *lookup\_field* atribut tracks objekta. Ovaj atribut je *true* ako je objekat bio kreiran za izbor vrednosti lookup polja kada korisnik klikne na dugme desno od ulaznog lookup polja. Mi ćemo kreirati tako da prodati trekovi nisu prikazani kada korisnik bira vrednost za lookup polje. Dodatno, dodajemo jednu poruku upozorenja korisniku o mogućnosti da vidi fakturu. Na kraju kod *on\_view\_form\_created* će biti kao:

function on\_view\_form\_created(item) {

if (!item.lookup\_field) {

item.table\_options.height -= 200;

item.invoice\_table = task.invoice\_table.copy();

item.invoice\_table.paginate = false;

item.invoice\_table.create\_table(item.view\_form.find('.view-detail'), {

height: 200,

summary\_fields: ['date', 'total'],

on\_dblclick: function() {

show\_invoice(item.invoice\_table);

}

});

item.alert('Double-click the record in the bottom table ' +

'to see the invoice in which the track was sold.');

}

}

var scroll\_timeout;

function on\_after\_scroll(item) {

if (!item.lookup\_field && item.view\_form.length) {

clearTimeout(scroll\_timeout);

scroll\_timeout = setTimeout(

function() {

if (item.rec\_count) {

item.invoice\_table.set\_where({track: item.id.value});

item.invoice\_table.set\_order\_by(['-invoice\_date']);

item.invoice\_table.open(true);

}

else {

item.invoice\_table.close();

}

},

100

);

}

}

function show\_invoice(invoice\_table) {

var invoices = task.invoices.copy();

invoices.set\_where({id: invoice\_table.master\_rec\_id.value});

invoices.open(function(i) {

i.edit\_options.modeless = false;

i.can\_modify = false;

i.invoice\_table.on\_after\_open = function(t) {

t.locate('id', invoice\_table.id.value);

};

i.edit\_record();

});

}

## Kako promeniti vrednost polja izabranih zapisa?

Pokazaćem kako promeniti Media Type polje Tracks kataloga na istu vrednost za izabrane zapise. Prvo postavljamo *multiselect* atribut *table\_options* na *true* za prikaz ček boksova uz levu marginu Tracks tabele za korisnika da izabere zapise i kreirati *Set media type* dugme u on\_view\_form\_created event hendleru u klijent modulu Tracks objekta.

function on\_view\_form\_created(item) {

item.table\_options.multiselect = true;

item.add\_view\_button('Set media type').click(function() {

set\_media\_type(item);

});

}

Kada se dugme pritisne, *set\_media\_type* funkcija definisana u modulu će biti izvršena.

U ovoj funkciji kreiramo kopiju Tracks objekta. Prosleđujemo copy metodu *handlers* opciju *false*. To značu da sve postavke na objektu napravljene u Form Dialogs u AppBuilder-u i sve funkcije i eventi definisani u klijent modulu objekta će biti neraspoložive za kopiju.

Potom analiziramo *selections* atribut koji je niz vrednosti primarnog ključa zapisa koje je izabrao korisnik.

Posle toga inicijalizujemo dataset kopije pozivajem open metoda sa open\_empty opcijom. Takodje postavljamo *fields options* tako da dataset ima samo jedno polje media\_type. Postavljamo zahtevane atribute tog polja na *true*.

Na kraju, pre pozivanja append\_record metod, mi dinamički pridružujemo *on\_edit\_form\_created* event hendler da promeni klik event OK dugmeta, koji je definisan u klijent modulu taska.

U novom *on\_click* event hendleru, mi prvo pozivamo post metod da proveri da li je media type postavljeno, ako je izuzetak podignut pozivamo *edit* metod da omogućimo korisniku da ga postavi.

function set\_media\_type(item) {

var copy = item.copy({handlers: false}),

selections = item.selections;

if (selections.length > 1000) {

item.alert('Too many records selected.');

}

else if (selections.length || item.rec\_count) {

if (selections.length === 0) {

selections = [item.id.value];

}

copy.open({fields: ['media\_type'], open\_empty: true});

copy.edit\_options.title = 'Set media type to ' + selections.length +

' record(s)';

copy.edit\_options.history\_button = false;

copy.media\_type.required = true;

copy.on\_edit\_form\_created = function(c) {

c.edit\_form.find('#ok-btn').off('click.task').on('click', function() {

try {

c.post();

item.server('set\_media\_type', [c.media\_type.value, selections],

function(res, error) {

if (error) {

item.alert\_error(error);

}

if (res) {

item.selections = [];

item.refresh\_page(true);

c.cancel\_edit();

item.alert(selections.length + '

record(s) have been modified.');

}

}

);

}

finally {

c.edit();

}

});

};

copy.append\_record();

}

}

Kada korisnik klikne na *OK* dugme, objektov *server* metod izvršava *set\_media\_type* funkciju na serveru, koji menja vrednost polja izabranih zapisa. Posle promene zapisa na serveru, mi na klijentu deselektujemo zapise, osvežamo podatke strane, zatvaramo editovanje pozivom *cancel\_edit* metod i informišemo korisnika of rezultatu.

def set\_media\_type(item, media\_type, selections):

copy = item.copy()

copy.set\_where(id\_\_in=selections)

copy.open(fields=['id', 'media\_type'])

for c in copy:

c.edit()

c.media\_type.value = media\_type

c.post()

c.apply()

return True

## Kako da upišem edit formu bez zatvaranja iste?

Možete to uraditi dodavanjem dugmeta koje će upisati zapis bez zatvaranja edit forme. Ispod je primer za sinhroni i asinhroni slučaj:

function on\_edit\_form\_created(item) {

var save\_btn = item.add\_edit\_button('Save and continue');

save\_btn.click(function() {

if (item.is\_changing()) {

item.post();

try {

item.apply();

}

catch (e) {

item.alert\_error(error);

}

item.edit();

}

});

}

function on\_edit\_form\_created(item) {

var save\_btn = item.add\_edit\_button('Save and continue');

save\_btn.click(function() {

if (item.is\_changing()) {

item.disable\_edit\_form();

item.post();

item.apply(function(error){

if (error) {

item.alert\_error(error);

}

item.edit();

item.enable\_edit\_form();

});

}

});

}

## Kako da sačuvam promene u dve tabele u istoj transakciji na serveru?

Ispod su dva primera. U prvom, svaki *apply* metod dobija svoju konekciju iz konekcionog pool-a i komituje posle toga u bazu podataka. U drugom, konekcija je dobijena iz konekcionog pool-a i prosledjena svakom apply metodu tako da se promene komituju na kraju.

import datetime

def change\_invoice\_date(item, invoice\_id):

now = datetime.datetime.now()

invoices = item.task.invoices.copy(handlers=False)

invoices.set\_where(id=invoice\_id)

invoices.open()

invoices.edit()

invoices.invoice\_date.value = now

invoices.post()

invoices.apply()

customer\_id = invoices.customer.value

customers = item.task.customers.copy(handlers=False)

customers.set\_where(id=customer\_id)

customers.open()

customers.edit()

customers.last\_modified.value = now

customers.post()

customers.apply()

import datetime

def change\_invoice\_date(item, invoice\_id):

now = datetime.datetime.now()

con = item.task.connect()

try:

invoices = item.task.invoices.copy(handlers=False)

invoices.set\_where(id=invoice\_id)

invoices.open()

invoices.edit()

invoices.invoice\_date.value = now

invoices.post()

invoices.apply(con)

customer\_id = invoices.customer.value

customers = item.task.customers.copy(handlers=False)

customers.set\_where(id=customer\_id)

customers.open()

customers.edit()

customers.last\_modified.value = now

customers.post()

customers.apply(con)

con.commit()

finally:

con.close()

## Kako sprečiti duplicirane vrednosti u polju tabele

Jedan od načina je da pišemo *on\_apply* event handler. U primeru ispod, delta parametar je dataset koji sadrži promene koje će biti smeštene u users tabelu. Idemo kroz zapise promena i ako zapis nije bio obrisan ili login polje nije promenjeno tražimo zapis u tabeli sa istim loginom i ako postoji podižemo izuzetak. Ako user edituje zapis na klijentu koristeći edit formu neće moći da sačuva promene i videće odgovarajuću poruku upozorenja.

def on\_apply(item, delta, params, connection):

for d in delta:

if not (d.rec\_deleted() or d.rec\_modified() and

d.login.value == d.login.old\_value):

users = d.task.users.copy(handlers=False)

users.set\_where(login=d.login.value)

users.open(fields=['login'])

if users.rec\_count:

raise Exception('There is a user with this login - %s' % d.login.value)

## Kako implementirati osnovni multi-tenancy?

Ako neki objekat ima *user\_id* polje, sledeći kod na server modulu objekta će odraditi posao:

def on\_open(item, params):

if item.session:

user\_id = item.session['user\_info']['user\_id']

if user\_id:

params['\_\_filters'].append(['user\_id', item.task.consts.FILTER\_EQ,

user\_id])

def on\_apply(item, delta, params, connection):

if item.session:

user\_id = item.session['user\_info']['user\_id']

if user\_id:

for d in delta:

if d.rec\_inserted():

d.edit()

d.user\_id.value = user\_id

d.post()

elif d.rec\_modified():

if d.user\_id.old\_value != user\_id:

raise Exception('You are not allowed to change record.')

elif d.rec\_deleted():

if d.user\_id.old\_value != user\_id:

raise Exception('You are not allowed to delete record.')

Ovaj kod koristi *session* atribut objekta da dobijemo jedinstveni *user\_id* i *on\_open* i *on\_apply* event hendlere.

*on\_open* event hendler obezbedjuje da *sql* select naredba koju aplikacija generiše vrati samo one zapise gde je *user\_id* polje isto kao i *user\_id* koji pravi zahtev. I *on\_apply* event hendler postavlja *user\_id* na user\_id korisnika koji dodaje ili menja zapise.

Možete koristiti opštiji pristup i raditi sa sledećim kodom u server modulu taska.Multi-tenancy će biti primenjen na svaki objekat koji ima user\_id polje:

def on\_open(item, params):

if item.field\_by\_name('user\_id'):

if item.session:

user\_id = item.session['user\_info']['user\_id']

if user\_id:

params['\_\_filters'].append(['user\_id', item.task.consts.FILTER\_EQ,

user\_id])

def on\_apply(item, delta, params, connection):

if item.field\_by\_name('user\_id'):

if item.session:

user\_id = item.session['user\_info']['user\_id']

if user\_id:

for d in delta:

if d.rec\_inserted():

d.edit()

d.user\_id.value = user\_id

d.post()

elif d.rec\_modified():

if d.user\_id.old\_value != user\_id:

raise Exception('You are not allowed to change record.')

elif d.rec\_deleted():

if d.user\_id.old\_value != user\_id:

raise Exception('You are not allowed to delete record.')

## Kao mogu koristiti podatke iz tabele druge baze podataka?

Možete koristiti tabele druge baze podataka. Prvo morate spec. ime tabele i informacije o poljima. Možete to uraditi na sledeći način:

* Izaberite *projektni nod* u task stablu i kliknite na *Database* dugme.
* Postavite *db manual mode* i spec. *konekcione atribute* za bazu podataka.
* Uvezite informacije o tabelama kao što je opisano u ’*Integracija sa postojećom baza podataka*’.
* Izaberite *projektni* *nod* task stabla i kliknite na *Database* dugme i *vratite prethodne vrednosti*.

Tada u server modulu novih objekata morate dodati kod da čitate i pišete podatke u bazu podataka. Ispod je kod za MySQL bazu podataka (auto incrementirajući primarni ključ):

import MySQLdb

from jam.db import mysql

def on\_open(item, params):

connection = item.task.create\_connection\_ex(mysql, database='demo', \

user='root', password='111', host='localhost', encoding='UTF8')

try:

sql = item.get\_select\_query(params, mysql)

rows = item.task.select(sql, connection, mysql)

finally:

connection.close()

return rows, ''

def on\_apply(item, delta, params):

connection = item.task.create\_connection\_ex(mysql, database='demo', \

user='root', password='111', host='localhost', encoding='UTF8')

try:

sql = delta.apply\_sql(params, mysql)

result = item.task.execute(sql, None, connection, mysql)

finally:

connection.close()

return result

Ako baza podataka koristi generatore da dobije vrednost primarnog ključa morate ih spec. za nove zapise (Firebird):

import fdb

from jam.db import firebird

def on\_open(item, params):

connection = item.task.create\_connection\_ex(firebird, database='demo.fdb', \

user='SYSDBA', password='masterkey', encoding='UTF8')

try:

sql = item.get\_select\_query(params, firebird)

rows = item.task.select(sql, connection, firebird)

finally:

connection.close()

return rows, ''

def get\_id(table\_name, connection):

cursor = connection.cursor()

cursor.execute('SELECT NEXT VALUE FOR %s FROM RDB$DATABASE' % (table\_name +

'\_SEQ'))

r = cursor.fetchall()

return r[0][0]

def on\_apply(item, delta, params):

connection = item.task.create\_connection\_ex(firebird, database='demo.fdb', \

user='SYSDBA', password='masterkey', encoding='UTF8')

for d in delta:

if not d.id.value:

d.edit()

d.id.value = get\_id(item.table\_name, connection)

for detail in d.details:

for r in detail:

if not r.id.value:

r.edit()

r.id.value = get\_id(r.table\_name, connection)

r.post()

d.post()

try:

sql = delta.apply\_sql(params, firebird)

result = item.task.execute(sql, None, connection, firebird)

finally:

connection.close()

return result

Možete koristiti *on\_open* i *on\_apply* evente taska. Ispod je kod iz task klijent modula:

import MySQLdb

from jam.db import mysql

def on\_open(item, params):

if item.item\_name in ['table1', 'table2']: # or

#if item.table\_name in ['table1', 'table2']:

connection = item.task.create\_connection\_ex(mysql, database='demo', \

user='root', password='111', host='localhost', encoding='UTF8')

try:

sql = item.get\_select\_query(params, mysql)

rows = item.task.select(sql, connection, mysql)

finally:

connection.close()

return rows, ''

def on\_apply(item, delta, params):

if item.item\_name in ['table1', 'table2']:

connection = item.task.create\_connection\_ex(mysql, database='demo', \

user='root', password='111', host='localhost', encoding='UTF8')

try:

sql = delta.apply\_sql(params, mysql)

result = item.task.execute(sql, None, connection, mysql)

finally:

connection.close()

return result

Napomena: Ne postavljajte History atribute na True za ove tabele. Ako to uradite podići će se izuzetak. History tabela mora biti jedna za sve baze podataka koje koristite u projektu. Možete pokušati da kreirate history tabelu u drugoj bazi podatka i pišete *on\_open* i *on\_apply* event hendlere za nju.

## Kako mogu procesirati zahtev ili dobiti neke podatke iz druge aplikacije ili servisa?

Možete pristupiti podacima vaše aplikacije za čitanje i pisanje slanjem *post* zahteva koji ima *ext* dodan na url. Na primer:

http://example.com/ext/edit

Kada web aplikacija na serveru primi takav zahtev ona generiše *on\_ext\_request* event.

## Kako mogu izvršiti kalkulacije u pozadini?

Možete koristiti ovaj kod u task server modulu da pokrenete pozadinski thread u web aplikaciji jednom svaka 3 minuta. (možete promeniti interval) da izvršite neke kalkulacije:

import threading

import time

import traceback

def background(task):

interval = 3 \* 60

time.sleep(interval)

while True:

if not time:

return

with task.lock('background'):

try:

print('background')

# some code to execute in background for example:

# tracks = task.tracks.copy()

# tracks.open()

# for t in tracks:

# t.edit()

# t.sold.value = #some value

# t.post()

# tracks.apply()

except Exception as e:

traceback.print\_exc()

time.sleep(interval)

def on\_created(task):

bg = threading.Thread(target=background, args=(task,))

bg.daemon = True

bg.start()

Napomena: Kada više web aplikacija rade u paralelnim procesima, *background* funkcija će biti izvršena za svaki proces. Da sprečimo simultano izvršavanje ove funkcije, koristimo lock metod taska.

## Da li su podržani detalji unutar detalja?

Možete imati detalje unutar detalja. Pretpostavimo da imate tri objekta - Polls, Questions and Answers. Answers je detalj Questions. Napravićemo da Questions bude detalj Polls. Jedan način da to uradimo je da dodamo integer polje pool u Questions i sledeći kod u Poll klijent modul:

function on\_edit\_form\_created(item) {

var q = task.questions.copy();

item.edit\_form.find('.form-footer').hide();

q.view\_options.form\_header = false;

q.on\_view\_form\_created = function(quest) {

quest.paginate = false;

};

q.on\_before\_append = function(quest) {

if (!item.id.value) {

quest.alert\_error('Poll is not specified.');

quest.abort();

}

};

q.on\_before\_post = function(quest) {

quest.poll.value = item.id.value;

};

q.set\_where({poll: item.id.value});

q.view(item.edit\_form.find('.edit-detail'));

}

function on\_field\_changed(field, lookup\_item) {

var item = field.owner;

item.apply();

item.edit();

}

function on\_before\_delete(item) {

var q = task.questions.copy();

q.set\_where({poll: item.id.value});

q.open();

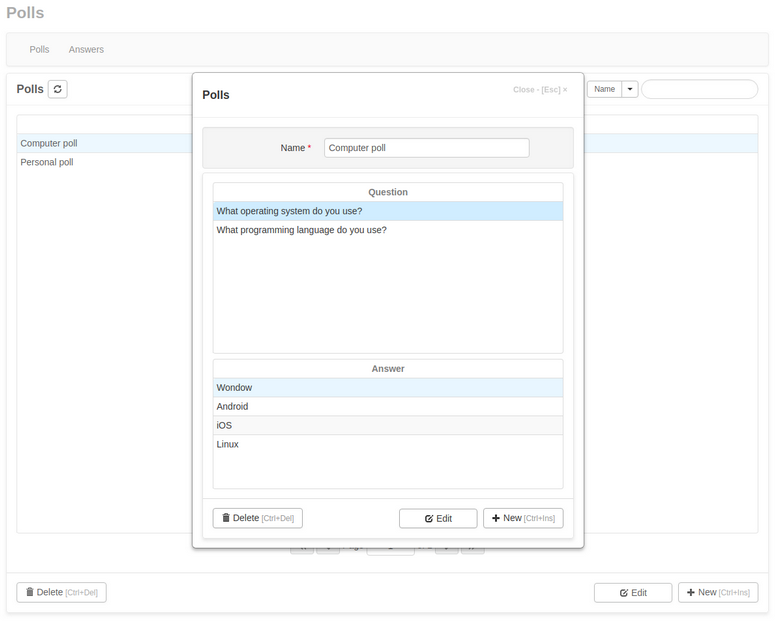
while (!q.eof()) {

q.delete();

}

q.apply();

}



## Izvoz u/Uvoz iz CSV fajlova

Prvo, u klijent modulu kreiramo dva dugmeta, koje izvršavaju odgovarajuće funkcije:

function on\_view\_form\_created(item) {

var csv\_import\_btn = item.add\_view\_button('Import csv file'),

csv\_export\_btn = item.add\_view\_button('Export csv file');

csv\_import\_btn.click(function() { csv\_import(item) });

csv\_export\_btn.click(function() { csv\_export(item) });

}

function csv\_export(item) {

item.server('export\_scv', function(file\_name, error) {

if (error) {

item.alert\_error(error);

}

else {

var url = [location.protocol, '//', location.host,

location.pathname].join('');

url += 'static/files/' + file\_name;

window.open(encodeURI(url));

}

});

}

function csv\_import(item) {

task.upload('static/files', {accept: '.csv', callback: function(file\_name) {

item.server('import\_scv', [file\_name], function(error) {

if (error) {

item.warning(error);

}

item.refresh\_page(true);

});

}});

}

Ove funkcije pozivaju sledeće funkcije u server modulu. U server modulu koristimo Python csv modul. Ne izvozimo sistemska polja – polje primarnog ključa i deleted fleg polje. Ispo je kod za Python3:

import os

import csv

def export\_scv(item):

copy = item.copy()

copy.open()

file\_name = item.item\_name + '.csv'

path = os.path.join(item.task.work\_dir, 'static', 'files', file\_name)

with open(path, 'w', encoding='utf-8') as csvfile:

fieldnames = []

for field in copy.fields:

if not field.system\_field():

fieldnames.append(field.field\_name)

writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

writer.writeheader()

for c in copy:

dic = {}

for field in copy.fields:

if not field.system\_field():

dic[field.field\_name] = field.text

writer.writerow(dic)

return file\_name

def import\_scv(item, file\_name):

copy = item.copy()

path = os.path.join(item.task.work\_dir, 'static', 'files', file\_name)

with open(path, 'r', encoding='utf-8') as csvfile:

copy.open(open\_empty=True)

reader = csv.DictReader(csvfile)

for row in reader:

print(row)

copy.append()

for field in copy.fields:

if not field.system\_field():

field.text = row[field.field\_name]

copy.post()

copy.apply()

## Autentifikacija

U Jam.py repozitorijumu ima Authentication projekt export file <http://jam-py.com/repository/auth.zip>. Ovaj projekat demonstrira tri poglavlja ove sekcije. Možete ga preuzeti, kreirati novi projekat i uvesti ovaj fajl.

### Kako autentifikovati iz prilagodjene korisničke tabele?

Podrazumevano, sve korisničke informacije se čuvaju u jednoj tabeli u bazi *admin*.*sqlite*. Ova tabela ima fiksnu strukturu koja se ne može menjati. U ovom odeljku opisujemo kako autentifikovati korisnika koristeći podatke iz prilagodjene tabele korisnika.

Prvo, kreiramo grupu objekata Autentifikacija, izaberemo je i dodamo objekat Users koji ima sledeća polja... Nećemo skladištiti u tabeli korisničku lozinku i koristiti ovo polje u interfejsu. Smestićemo *salted heš* u polju *password\_hash*. Takođe smo kreirali lookup listu *Roles*, koje ćemo koristili u definiciji polja roles. U njemu smo dodali iste uloge (ID-ovi i imena) kao u tabeli Roles, moraćemo ubuduće da sinhronizujemo ove uloge. U tabeli Roles je potrebno dozvoliti da Users objektu imaju pristup samo ljudi odgovorni za to.

Uklonili smo *password\_hash* polje sa liste polja u dijalogu prikaza i dijalogu za uređivanje obrasca. U User modulu servera definišemo sledeće *on*\_*apply* event hendler:

def on\_apply(item, delta, params, connection):

for d in delta:

if not (d.rec\_deleted() or d.rec\_modified() and d.login.value ==

d.login.old\_value):

users = d.task.users.copy(handlers=False)

users.set\_where(login=d.login.value)

users.open(fields=['login'])

if users.rec\_count:

raise Exception('There is a user with this login - %s' % d.login.value)

if d.password.value:

d.edit();

d.password\_hash.value = delta.task.generate\_password\_hash(d.password.value)

d.password.value = None

d.post();

Ovde proveramo da li je tu korisnik sa istim loginom i podižemo izuzetak ako takav korisnik postoji, inače generišemo hash koristeći *generate\_password\_hash* metod task objekta i setujemo passwor na vrednost *None*. U klijent modulu definišemo *on\_field\_get\_text* event hendler. On prikazuje **\*\*\*\*\*\*** string umesto lozinke.

function on\_field\_get\_text(field) {

var item = field.owner;

if (field.field\_name === 'password') {

if (item.id.value || field.value) {

return '\*\*\*\*\*\*\*\*\*\*';

}

}

}

Na kraju, definišemo *on\_login event* hendler u task server modulu:

def on\_login(task, form\_data, info):

users = task.users.copy(handlers=False)

users.set\_where(login=form\_data['login'])

users.open()

if users.rec\_count == 1:

if task.check\_password\_hash(users.password\_hash.value, form\_data['password']):

return {

'user\_id': users.id.value,

'user\_name': users.name.value,

'role\_id': users.role.value,

'role\_name': users.role.display\_text

}

Sada moramo dodati jednog admina u Users koji ima prava sa radi sa korisnicima. Posle toga možemo setovati *Safe mode* u *Parameters* dijalogu projekta.

### Kako kreirati registracionu formu?

U ovom delu pretpostavićemo da smo kreirali Users objekat iz prethodnog poglavlja. Sada kreiramo *register.html* file. On sadrži registracionu formu:

<form id=login-form target=dummy class=form-horizontal style=margin: 0;>

<div class=control-group>

<label class=control-label for=name>Name</label>

<div class=controls>

<input type=text id=name placeholder=Login>

</div>

</div>

<div class=control-group>

<label class=control-label for=login>Login</label>

<div class=controls>

<input type=text id=login placeholder=Login>

</div>

</div>

<div class=control-group>

<label class=control-label for=password1>Password</label>

<div class=controls>

<input type=password id=password1

placeholder=Password autocomplete=on>

</div>

</div>

<div class=control-group>

<label class=control-label for=password2>Repeat password</label>

<div class=controls>

<input type=password id=password2

placeholder=Repeat password autocomplete=on>

</div>

</div>

<div class=alert alert-success style=margin: 0; display: none>

You have been successfully registered.

</div>

<div class=alert alert-error style=margin: 0; display: none>

</div>

<div class=form-footer>

<input type=button class=btn expanded-btn pull-right

id=register-btn value=OK tabindex=3>

</div>

</form>

I javaskript kod:

$(document).ready(function(){

function register(name, login, password) {

$.ajax({

url: ext/register,

type: POST,

contentType: application/json;charset=utf-8,

data: JSON.stringify([name, login, password]),

success: function(response, textStatus, jQxhr) {

if (response.result.data) {

show\_alert(response.result.data);

}

else {

$(div.alert-success).show();

setTimeout(

function() {

window.location.href = index.html;

},

1000

);

}

},

error: function(jqXhr, textStatus, errorThrown) {

console.log(errorThrown);

}

});

}

function show\_alert(message) {

$(div.alert-error)

.text(message)

.show();

}

$('input').focus(function() {

$(div.alert).hide();

});

$(#register-btn).click(function() {

var name = $(#name).val(),

login = $(#login).val(),

password1 = $(#password1).val(),

password2 = $(#password2).val();

if (!name) {

show\_alert('Name is not specified');

}

else if (!login) {

show\_alert('Login is not specified');

}

else if (!password1) {

show\_alert('Password is not specified');

}

else if (password1 !== password2) {

show\_alert('Passwords do not match');

}

else {

register(name, login, password1)

}

})

})

Kada korisnik klikne na *OK* dugme, javaskript će poslati na server *ajax post zahtev* sa *url*-om *ext/register* i parametrima *name*, *login*, *password*. Kada server primi zahtev koji počinje sa *ext/,*  on okida *on\_ext\_request* event hendler. U task server modulu je kod *on\_ext\_request* event hendlera:

def on\_ext\_request(task, request, params):

reqs = request.split('/')

if reqs[2] == 'register':

name, login, password = params

users = task.users.copy(handlers=False)

users.set\_where(login=login)

users.open()

if users.rec\_count:

return 'Existing login, please use different login'

users.append()

users.name.value = name

users.login.value = login

users.password\_hash.value = task.generate\_password\_hash(password)

users.role.value = 2

users.post()

users.apply()

Ovde proveramo da li je ’register’ u url-u a zatim proveramo da li postoji korisnik sa datim loginom, ako ne registrujemo korisnika.

### Kako dati korisniku mogućnost da menja lozinku?

Prvo kreiramo *Change password* objekat. Pri tome setujemo *Virtual* *table,* a *Visible* atribut resetujemo u *Item Editor Dialog*. Dodamo objektu dva polja: *Old password*, *New password*. Koristićemo ovaj objekat za prikazivnje *Change password* dijaloga. Da otvorimo ovaj dijalog daodajmo *Change password* meni stavku sa *id* prosledjenim u *index.html*:

<div class=container>

<div id=taskmenu class=navbar>

<div class=navbar-inner>

<ul id=menu class=nav>

</ul>

<ul id=menu-right class=nav pull-right>

<li id=pass><a href=#>Change password</a></li>

</ul>

</div>

</div>

</div>

i u task clijent modulu *on\_page\_loaded* event hendleru dodajmo kod:

if (task.change\_password.can\_view()) {

$(#menu-right #pass a).click(function(e) {

e.preventDefault();

task.change\_password.open({open\_empty: true});

task.change\_password.append\_record();

});

}

else {

$(#menu-right #pass a).hide();

}

Ovaj kod proverava da li korisnik ima prava da vdi objekat i potom otvara prazan dataset i kreira edit formu, inače sakriva meni stavku. U *Change password* clijent modulu dodajmo kod:

function on\_edit\_form\_created(item) {

item.edit\_form.find(#ok-btn)

.off('click.task')

.on('click', function() {

change\_password(item);

});

item.edit\_form.find(#cancel-btn)

.off('click.task')

.on('click', function() {

item.close\_edit\_form();

});

}

function change\_password(item) {

item.post();

item.server('change\_password',

[item.old\_password.value, item.new\_password.value],

function(res) {

if (res) {

item.warning('Password has been changed. <br> The application will be

reloaded.',

function() {

task.logout();

location.reload();

});

}

else {

item.alert\_error(Can't change the password.);

item.edit();

}

}

);

}

function on\_field\_changed(field, lookup\_item) {

var item = field.owner;

if (field.field\_name === 'old\_password') {

item.server('check\_old\_password', [field.value], function(error) {

if (error) {

item.alert\_error(error);

}

});

}

}

function on\_edit\_form\_close\_query(item) {

return true;

}

U njemu ponovo dodeljujemo dugmadima OK i Cancel klik evente. Podrazumevano, oni su definisani u task klijent modulu da sačuvaju promene u bazu podataka i odustanu od promena. U *on\_edit\_form\_close\_query* even hendler mi vraćamo *true* tako da on\_edit\_form\_close\_query declarsan u task klijent modulu, koji prikazuje *Yes No Cancel* dijalog neće biti izvršen.

*on\_field\_changed* event hendler će proveriti da li je *old password* korektan. Ovaj hendler i *change\_password* funkcija šalju zahtev na server da izvrši funkciju definisanu u objektovaom server modulu:

def change\_password(item, old\_password, new\_password):

user\_id = item.session['user\_info']['user\_id']

users = item.task.users.copy(handlers=False)

users.set\_where(id=user\_id)

users.open()

same\_password = item.task.check\_password\_hash(users.password\_hash.value,

old\_password)

if users.rec\_count== 1 and same\_password:

users.edit()

users.password\_hash.value = item.task.generate\_password\_hash(new\_password)

users.post()

users.apply()

return True

else:

return False

def check\_old\_password(item, old\_password):

user\_id = item.session['user\_info']['user\_id']

users = item.task.users.copy(handlers=False)

users.set\_where(id=user\_id)

users.open()

same\_password = item.task.check\_password\_hash(users.password\_hash.value,

old\_password)

if users.rec\_count == 1 and same\_password:

return

else:

return 'Invalid password'

Oni koriste sesiju da dobiju *id* trenutnog korisnika. Posle promene lozinke klijent se reučitava.

## Integracija sa postojećom bazom podataka

Možete koristiti Jampy sa postojećom bazom podataka, podržanom od strane frejmvorka:

* Kreirajte novi projekat sa postojećom bazom podataka.
* Da bi uvezli tabele u grupe kataloga ili žurnala obrišite *Common* polja, ili kreirajte nove prazne grupe.
* Izaberite *Project* nod i kliknite na *Database* dugme. Setuj *DB manual mode.*
* Izaberite grupu u koju želite da uvezete tabelu i kliknite na *Import* dugme.
* U formi koja će se pojaviti duplo kliknite na tabelu da bi je uvezli.
* U Item Editor Dialog proverite da li su sva polja imaju validne tipove. Ako je tip polja prikazan crvenom bojom, pokušajte da izaberete odgovarajući tip.
* Možete importovati subset polja tabele.

Pre nego sačuvate, spec. primarni ključ objekta i ime generatora ako je neophodno.

* Pošto ste sačuvali uvezeni objekat, idite na projektnu stranu i pogledajte kako je prikazan.
* Posle uvoza nekoliko tabela, možete spec. lookup polja, (u *DB manual mode*).

Napomena: Budite vrlo pažljivi kada radite ove operacije. Kada je *DB manual mode* uklonjen bilo koja promena na objektu će promeniti ogovarajuću DB tabelu. Ako obrišete objekat, tabela će biti uklonjena iz baze podataka.

Napomena: Da bi bila uvezena, tabela baze podataka mora imati primarni ključ sa jednim poljem.

Napomena: Binarna polja ne smeju biti uvožena.

## Zaključavanje zapisa

U Jampy aplikacija koju gradite može implementirati zaključavanje zapisa dok korisnici konkurentno menjaju jedan zapis. Jampy koristi optimistički model zaključavanja, takodje poznat kao *optimistička konkurentna kontrola*.

Kada serverska aplikacija izvrši *edit\_record* metod, ona dobija tekuću verziju zapisa i čuva ga. Kada korisnik počne da upisuje zapis, serverska aplikacija proverava tekuću verziju zapisa. Ako je različita od sačuvane vrednosti (drugi korisnik je promenio zapis), aplikacija upozorava korisnika i zabranjuje upis.

Ovaj mehanizam zaključavanja je veoma lak za implementaciju. Sve što treba da uradite je da kreirate objekat koji čuva verzije zapisa. Otvori projektni Parameters dijalog i klikni na dugme desno od *Lock**item* ulaza. U dijalogu koji se pojavljuje klikni na *Create**lock**item* dugme. Posle toga morate setovati *Edit**lock* atribut u Item Editor Dialog-u.

## Čuvanje istorije promena koje su napravli korisnici

Da biste sačuvali istoriju promena koje su napravili korisnici, morate odrediti objekat koji će ih čuvati. Da biste to učinili, otvorite *Parameters* dijalog projekta i kliknite na dugme desno od *History item input*. U dijalogu koji će se pojaviti kliknite na dugme *Create History*. Posle toga morate da postavite atribut *Keep* *history* objekta za koga želite da čuvate istoriju promena.

Da biste videli istoriju promena zapisa kliknite na ikonu na levo od dugmeta za zatvaranje na desnom delu zaglavlja edit forme. Ili to možete učiniti pomoću *show*\_*history* metode objekta.

Napomena: Promene se upisuju kada se koristi *apply* metoda objekta. Promene na bazi podataka napravljene sa prilagođenim sql zahtevima nisu sačuvane u istoriji promena.

Napomena: Ove promene mogu značajno da povećaju veličinu baze podataka. Budite oprezni.

# Jam.py reference klasa

Serverska strana je implementirana u *Python*-u i koristi *Werkzeug* biblioteku, klijentska strana u *JavaScript* i koristi *JQuery* i *Bootstrap*.

## Klijentska strana - javascript reference

Svi objekti okvira predstavljaju task stablo. Ispod su klase za svaku vrstu objekata task stabla.

### *class* AbstractItem()

*AbstractItem* klasa je predak za sve objekte task stabla. Ispod su atributi i metodi klase.

#### Atributi

##### ID

*ID* atribut je jedinstven id objekta u okviru. *ID* atribut je najvažniji za referenciranje objekta. Koristi se interno.

##### item\_caption

*Item\_caption* atribut spec. ime objekta koje se pojavljuje korisniku.

##### item\_name

Spec. ime objekta koje se koristi u kodu. Koristi *item\_name* za referenciranje objekta u kodu.

##### item\_type

Spec. tip objekta. Koristi *item\_type* atribut za dobijanje tipa objekta. Može imati jedno od sledećih vrednosti:

* task,
* items,
* details,
* reports,
* item,
* detail\_item,
* report,
* detail

##### items

Lista svih objekata kojima je vlasnik objekat. Koristi *items* za pristup objektima kojima je vlasnik ovaj objekat.

##### owner

Indicira vlasnika objekta. Koristi *owner* da pronadješ vlasnika objekta.

##### task

Indicira koren task stabla koji je vlasnik objekta. Koristi *task* atribut da pronadješ koren objekat task stabla kojeg je objekat član.

#### Metode

##### abort(*message*)

Koristi *abort* metod da baciš izuzetak. Može biti korisno kada je potrebno sa prekinete izvršenje nekog *on\_before* eventa.

*Primer*

Sledeći kod će baciti izuzetak sa tekstom: ’*execution aborted: invoice\_table - a quantity value is required*’

function on\_before\_post(item) {

if (item.quantity.value === 0) {

item.abort('a quantity value is required');

}

}

##### alert(mess, options)

Koristi *alert* metod za kreiranje pop-up poruke u gornjem desnom uglu aplikacije koja isčezava posle prvog klika na stranu.

* mess parametar spec. text koji će biti prikazan.
* options parametar je objekat sa sledećim atributima:
  + type - indicira tip poruke – njen font, boju pozadine i header tekst, ako nije spec. u heder parametrima. Mora biti jedno od:
    - info,
    - error,
    - success

Podrazumevana vrednost je *info*.

* + header – spec. header poruke upozorenja.
  + pulsate – ako je *true*, *header* će pulsirati, podrazumevano je *true.*
  + show\_header – ako je *false*, header neće biti prikazan.

Metode alert\_error i alert\_success su iste kao alert sa odgovarajućim tipom.

*Primeri*

item.alert\_error('Failed to send the mail: ' + err);

item.alert('Successfully sent the mail');

##### can\_view()

Koristi *can\_view* metod da odredite da li korisnik ima prava da dobije pristup objektovom datasetu ili da vidi generisan izveštaj kada je projektov *Safe mode* parametar postavljen. Ako projektni *Safe mode* parametar nije postavljen metod uvek vraća *true*. Korisničke privilegije su postavljene u Role nodu projektnog stabla.

*Primer*

if (item.visible && item.can\_view()) {

$(#submenu)

.append($('<li></li>')

.append(

$('<a href=></a>')

.text(item.item\_caption)

.data('item', item);

)

);

}

##### each\_item(function(item))

Koristi *each\_item* metod za iteriranje objekata koje su vlasništvo objekta.

* *each\_item()* metod spec. funkciju koja će raditi na svakom dečijem objektu (dečiji objekat je prosledjen kao parametar).

Možete prekinuti *each\_item* petlju na partikularnoj iteracji ako *callback* funkcija vrati *false*.

*Primer*

Sledeći kod će ispisati sve kataloge projekta u konzoli brauzera:

function on\_page\_loaded(task) {

task.catalogs.each\_item(function(item) {

console.log(item.item\_name);

})

}

##### hide\_message(*form*)

koristi *hide*\_*message* metod da zatvoriš modalnu formu kreiranu *message* metodom.

* *form* parametar je JQuery objekat vraćen *message* metod.

##### item\_by\_ID(*ID*)

*item\_by\_ID* pretražuje sve objekte projektnog task stabla, startujučći od trenutnog objekta, za objekat čiji je *ID* atribut je jednak ID parametru.

##### load\_module(*callback*)

Koristi *load\_module* metod za dinamičko učitavanje javascript fajla objekt modula, pre izvršenja *callback-a*. Metod proverava da li je modul učitan, ako nije učitava modul sa servera, inicijalizuje objekat i izvršava *callback* funkciju, inače samo se *callback* funkcija izvršava.

* Objekat je prosledjen callback funkciji kao parametar.

Zahtev na serveru se izvršava asinhrono.

*Primer*

Ispod, *do\_some\_work* funkcija je izvršena samo ako objekat modul je učitan:

function some\_work(item) {

item.load\_module(do\_some\_work);

}

function do\_some\_work(item) {

// some code

}

##### load\_modules(module\_array, callback)

Koristi load*\_*modules metod za dinamičko učitavanje spec. modula pre izvršenja callback. Metod radi na isti način kao load\_module, samo učitava i inicijalizuje sve module objekta spec. u module*\_*array.

*Primer*

Ispod, *do\_some\_work* funkcija je izvršena samo kada moduli objekta i njegovog vlasnika su učitani.

function some\_work(item) {

item.load\_modules([item, item.owner], do\_some\_work);

}

function do\_some\_work(item) {

// some code

}

##### load\_script(js\_filename, callback, onload)

Koristi *load\_script* metod za učitavanje javascript fajla sa severa, pre izvršenja *callback*.

* Metod proverava da li je fajl već učitan, ako nije učitava ga sa servera, izvršava *onload* funkciju ako je spec. i potom izvršava *callback*, inače izvršava *callback* funkciju.
* Objekat je prosledjen *callback* funkciji kao parametar.
* *js****\_****filename* trebalo bi da spec. put do javascript fajla relativno na serverski folder.

Zahtev na serveru se izvršava asinhrono.

*Primer*

Ispod, *do\_some\_work* funkcija je izvršena samo kada je *lib.js* fajl iz serverskog js foldera učitan.

function some\_work(item) {

item.load\_script('js/lib.js', do\_some\_work);

}

function do\_some\_work(item) {

// some code

}

##### message(*mess*, *options*)

Koristi *message* metod za kreiranje modalne forme.

* *mess* parametar spec. *text* ili *html* sadržaj koji će se pojaviti u telu forme.
* *options* parametar je objekat sa sledećim atributima:
  + title – naslov forme,
  + width – širina forme, podrazumevano 400px
  + height – visina forme,
  + margin – koristi *margin* atribut za definiciju margina tela forme,
  + text*\_*center – ako je *true*, tag body će biti centriran, podrazumevano je *false*,
  + buttons – objekat koji definiše dugmad koja će biti kreirana u footer-u forme,
    - key - imena dugmadi,
    - value - funkcije koje će biti izvršene kada su dugmad kliknuta
  + button*\_*min*\_*width – min. Šrina dugmadi, podrazumevano je 100px,
  + center*\_*buttons – iako je *true*, dugmad su centrirana, podrazumevano je *false*,
  + close*\_*button – ako je *true*, aplikacija kreira *close* dugme u gornjem desnom uglu forme, podrazumevano je *true*,
  + close*\_*on*\_*escape – ako je *true*, forma će biti zatvorena kada korisnik pritisne Escape, podrazumevano je *true*,
  + print – ako je *true*, palikacija kreira *print* dugme u gornjem desnom uglu forme za štampu tela forme, podrazumevano je *false.*

Metod vraća jquery objekat forme. Za zatvaranje forme iz kodaprosledi ovaj objekat *hide\_message* metodu.

*Primer*

task.message(

'<a href=http://jam-py.com/ target=\_blank><h3>Jam.py</h3></a>' +

'<h3>Demo application</h3>' +

' with <a href=http://chinookdatabase.codeplex.com/ target=\_blank>

Chinook Database</a>' +

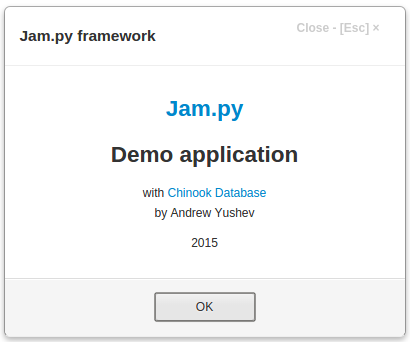
'<p>by Andrew Yushev</p>' +

'<p>2015</p>',

{title: 'Jam.py framework', margin: 0, text\_center: true, buttons: {OK: undefined},

center\_buttons: true}

);



##### question(mess, yes\_callback, no\_callback, options)

Kreira modalnu formu sa *OK*, *Cancel* dugmadima.

* mess parametar spec. *text* ili *html* sadržaj koji će se pojaviti u telu forme.
* Ako yes\_callback, no\_callback funkcije su spec. one će biti izvršene kada se klikne na OK ili Cancel dugme, respektivno, i forma će se zatvoriti.

*Primer*

Sledeći kod kreira modalnu formu i briše izabrani zapis ako korisnik klikne na *OK* dugme:

item.question('Delete record?',

function() {

item.delete();

}

);

##### server(func\_name, params, callback)

Koristi sever metod da izvrši funkciju definisanu u server modulu objekta.

* *Sever* metod izvršava funkciju sa imenom *func\_name* definisanu u server modulu objekta sa parametrima spec. u *params*.
* Ako je *callback* spec., funkcija na serveru je izvršena asinhrono, posle toga je izvršen *callback* sa parametrom koji je rezultat izvršenja funkcije, inače funkcija je izvršena sinhrono i vraća rezultat izvršenja serverske funkcije.

Ako je izuzetak podignut za vreme operacije na serveru i *callback* parametar nje prosledjen (sinhrono izvršenje), klijent baca izuzetak. Ako *callback* parametar postoji, izuzetak je prosleđen u *callback* kao parametar. Kada je izuzetak podignut za vreme izvršenja funkcije na serveru, aplikacija na klijenu baca izuzetak sa tekstom serverskog izuzetka.

Prvi parametar funkcije mora biti item, mora biti praćen parametrima spec. u funkciji na serveru, *params* je lista parametara. Ako funkcija nema parametara *params* može biti izostavljeno.

*Primer*

Funkcija definisana u Invoices journal server modulu:

def get\_total(item, id\_value):

result = 0;

copy = item.copy()

copy.set\_where(id=id\_value)

copy.open()

if copy.record\_count():

result = copy.total.value

else:

raise Exception, 'Journal invoices does not have a record with id %s' %

id\_value

return result;

sledeći kod u Invoices journal kljent modulu će izvršiti serversku *get\_total* funkciju:

task.invoices.server('get\_total', [17], function(total, err) {

if (err) {

throw err;

}

else {

console.log(total);

}

});

##### warning(*mess*, *callback*)

koristi *warning* za kreiranje modalne forme sa *OK* dugmetom.

* *mess* parametar spec. *text* ili *html* sadržaj koji će se pojaviti u telu forme.
* Akoje *callback* funkcija spec., biće izvršena kada korisnik klikne na dugme i forma će biti izvršena.

*Primer*

item.warning('No record selected.');

##### yes\_no\_cancel(mess, yes\_callback, no\_callback, cancel\_callback)

Koristi yes**\_**no**\_**cancel za kreiranje malne forme sa Yes, No, Cancel dugmadima.

* mess parametar spec. *text* ili *html* sadržaj koji će se pojaviti u telu forme.
* Ako yes**\_**callback, no**\_**callback, cancel**\_**callback funnkcije su spec. one će biti izvršene kada korisnik klikne na *Yes*, *No* ili *Cancel* dugme, respektivno, i forma će se zatvoriti.

*Primer*

Sledeći kod je izvršen kada korisnik klikne na *close* dugme u gornjem levom uglu objetove edit forme.

function on\_edit\_form\_close\_query(item) {

var result = true;

if (item.is\_changing()) {

if (item.is\_modified()) {

item.yes\_no\_cancel('Data has been modified. Save changes?',

function() {

item.apply\_record();

},

function() {

item.cancel\_edit();

}

);

result = false;

}

else {

item.cancel();

}

}

return result;

}

### *class* Task()

*Task* klasa se koristi za kreiranje korena task stabla projekta. Ispod su atributi, metodi i eventi. Isto tako klasa nasleđuje atribute i metode klase *AbstractItem*.

#### Atributi

##### forms\_container

*forms\_container* je JQuery objekat u kome aplikacija kreira forme. Za inicijalizaciju *forms\_container* koristi *set\_forms\_container* metod ili *create\_menu* metod. Podrazumevano koristi *create\_menu* metod.

##### forms\_in\_tabs

Ako je *forms\_in\_tabs* atribut postavljen i *forms\_container* je spec. aplikacija će kreirati forme u tabovima. Ovaj atribut može biti postavljen u Interface tabu Parameters dijalog projekta.

##### safe\_mode

Proverite safe\_mode atribut ta odredite da li je *safe mode* parametar projekta postavljen..

*Primer*

function on\_page\_loaded(task) {

$(#title).html(task.item\_caption);

if (task.safe\_mode) {

$(#user-info).text(task.user\_info.role\_name + ' ' +

task.user\_info.user\_name);

$('#log-out')

.show()

.click(function(e) {

e.preventDefault();

task.logout();

});

}

task.tasks.view($(#content));

}

##### templates

*templates* atribut smešta šablone formi projekta.

##### user\_info

Koristi user\_info atribut za dobijanje informacija o korisniku kada je *safe mode* parametar projekta postavljen. *user\_info* je objekat koji ima sledeće atribute:

* user\_id – korisnički *id*
* user\_name – korisničko ime
* role\_id – korisnički *role id*
* role\_name – *rola* kojoj je korisnik pridružen
* admin – ako je *true* korisnik može raditi u AppBuilder-u

Ako *safe mode* je *false* *user\_info* atribut je prazan objekat.

*Example*

function on\_page\_loaded(task) {

$(#title).html('Jam.py demo application');

if (task.safe\_mode) {

$(#user-info).text(task.user\_info.role\_name + ' ' + task.user\_info.user\_name);

$('#log-out')

.show()

.click(function(e) {

e.preventDefault();

task.logout();

});

}

// some initalization code

}

#### Metodi

##### add\_tab(container, tab\_name, options)

add\_tab metod kreira tab za kontejner.

* container je JQuery objekat za kontejnerski element.
* tab\_name je ime taba.
* Koristi options da spec. opcione parametre. To je objekat može imati sledeće atribute:
  + tab\_id – jedinstveni string koji ident. tab
  + show\_close\_btn – ako je *true* *close* tab dugme će se pojaviti, može biti korišćeno za zatvaranje taba
  + set\_active – ako je *true* novi tab postaje aktivan
  + on\_close - *callback* funkcija koja će biti pozvana kada je close dugme kliknuto

Funkcija vraća JQuery objekat sa *tab-pane* klasom koja će biti prikazana kada tab postane aktivan.

*Primer*

Sledeći kod kreira tab za editovanje Customers kataloga. Koristi create\_inputs metod:

function on\_edit\_form\_created(item) {

var container = item.edit\_form.find('.tabs');

task.init\_tabs(container);

item.create\_inputs(task.add\_tab(container, 'Customer'),

{fields: ['firstname', 'lastname', 'company', 'support\_rep\_id']}

);

item.create\_inputs(task.add\_tab(container, 'Address'),

{fields: ['country', 'state', 'address', 'postalcode']}

);

item.create\_inputs(task.add\_tab(container, 'Contact'),

{fields: ['phone', 'fax', 'email']}

);

}

Ispod je edit html šablon za Customers katalog:

<div class=customers-edit>

<div class=form-body>

<div class=tabs>

</div>

</div>

<div class=form-footer>

<button type=button id=ok-btn class=btn btn-ary expanded-btn>

<i class=icon-ok></i> OK<small class=muted>&nbsp;[Ctrl+Enter]</small>

</button>

<button type=button id=cancel-btn class=btn expanded-btn>

<i class=icon-remove></i> Cancel

</button>

</div>

</div>

##### close\_tab(container, tab\_id)

Koristi *close\_tab* metod za zatvaranje taba u kontejneru ident. sa *tab-id*.

##### create\_menu(menu, forms\_container, options)

*create\_menu* metod kreira menije zasnovane na objektima task stabla. Ako je *Display forms in tabs* atribut u Projects parameters - Interface postavljen, inicijalizuje tabove koji će biti kreirani za prikazivanje formi. Metod iterira kroz objekte task stabla i dodaje objekte u meni za koje *visible* atribut postavljen na *true*, korisnik će ga videti ako ima odgovarajuća prava. Metod koristi pridruživanje klik eventa meni stavke tako da će *print* metod izveštaja biti izvršen kada korisnik klikne na na njega ili *view* metod odgovarajućeg objekta kada korisnik klikne na neku drugu stavku menija.

Sledeći parametri mogu biti prosledjeni metodu:

* menu - JQuery objekat meni elementa iz *index.html* fajla,
* forms\_container - JQuery objekat elemnta koji sadrži formu kreiranu *view* metodom,
* options – objekat koji može imati sledeće atribute:
  + custom\_menu – koristi za kreiranje prilagodjenog menija, vidi dole za detalje,
  + view\_first – ako je *true*, *view* forma prvog objekta u meniju će biti prikazana pošto je meni kreiran, podrazumevano je *false,*
  + create\_single\_group – ako je *true*, i samo jedna grupa u task stablu ima objekte, meni stavka za grupu će biti kreirana, imaće padajući meni za objekte grupe, inače meni stavka za svaki objekat će biti kreirana, podrazumevano je *false*.
  + splash\_screen – html koji će biti prikazan u *forms\_container* kada su svi tabovi zatvoreni.

###### Opcije korisničkog menija

Da kreirate svoj prilagodjeni meni morate postaviti *custom\_menu* opciju.

Opcija ima listu objekata, od kojih svaki može biti:

* *Naziv* 
  + Jam.py objekta ili
  + grupe objekata,
* *Niz*:
  + prvi element niza je string - *ime meni stavke*,
  + drugi je lista *Jam.py* *objekata*,
* *Objekat:* 
  + ključelementjestring *– ime meni stavke,*
  + vrednost - lista *Jam.py objekata,*
* *Objekat:* 
  + ključ element - *ime meni stavke*,
  + vrednost - lista Jam.py *objekata,* ili/i
  + *vrednost - objekat* 
    - *čiji je ključ ime stavke menija,*
    - *vrednost - funkcija* koje će biti izvršena kada je na meni stavku kliknuto,
* Objekat:
  + kombinacija gore navedenih mogućnosti, radi izgradnje podmenija.

Da dodate separator, prazni string (’’) može biti dodat u listu meni stavki.

Primer

task.create\_menu($("#menu"), $("#content"), {

splash\_screen: '<h1 class="text-center">Jam.py Demo Application</h1>',

view\_first: true

});

Primer sa prilagodjenim menijem:

let menu = [

['First', [task.invoices, task.customers]],

{'Second': [task.catalogs, '', task.reports]},

{Third: [task.tracks, {Params: function() {alert('params clicked')}}]},

{Fourth: [task.task.analytics, {'Artists list': [task.artists]}]},

task.reports, {Params: function() {alert('params clicked')}},

];

task.create\_menu($("#menu"), $("#content"), {

custom\_menu: menu,

splash\_screen: '<h1 class="text-center">Jam.py Demo Application</h1>',

view\_first: true

});

##### init\_tabs(container, tabs\_position)

*init\_tabs* metod inicializje tabove za kontejner.

* container je JQuery objekat za kontejner element.
* tabs\_position parametar spec. gde će tabovi, kreirani sa *add\_tab* metodom biti pozicionirani. To je string, sa jednom od sledećih vrednosti:
  + tabs-below
  + tabs-left
  + tabs-right

Ako je parametar izostavljen, tab će biti pozicioniran na vrhu kontejnera.

Pošto je ovaj metod pozvan možete koristiti *add\_tab* metod za kreiranje tabova.

##### load(callback)

*load* metod učitava projektno task stablo sa servera i inicijalizuje ga. Kada web brauzer učitava *jam.js* bibliotečni fajl u index.html fajlu, jam.js kreira prazni task objekat. Posle toga aplikacija pokreće *on\_page\_loaded* event.

*Primer*

Sledeći kod je iz projektnog *index.html* fajla:

<script src=/jam/js/jam.js></script>

<script src=/js/events.js></script>

<script>

$(document).ready(function(){

task.load();

});

</script>

##### login(*callback*)

*login* metod kreira login formu iz div-a templates index.html fajla. Poziva ga *load* metod kada je projektni *safe mode* parametar postavljen.

##### logout()

Pozovi *logout* za odjavu korisnika.

*Primer*

function on\_page\_loaded(task) {

$(#title).html('Jam.py demo application');

if (task.safe\_mode) {

$(#user-info).text(task.user\_info.role\_name + ' ' + task.user\_info.user\_name);

$('#log-out')

.show()

.click(function(e) {

e.preventDefault();

task.logout();

});

}

// some initalization code

}

##### set\_forms\_container(*container*, *options*)

*set\_forms\_container* može biti korišćen za inicijalizaciju *forms\_container* atributa koj će sadržavati forme aplikacije. Ako je *forms\_in\_tabs* atribut postavljen aplikacija takodje inicijalizuje tabove koji će biti korišćen za prikaz formi.

* container je JQuery objekat koji će biti korišćen kao kontejner aplikacionih formi.
* options parametar može imati sledeće atribute:
  + splash\_screen - *html* koji će biti prikazan u forms\_container kada su svi tabovi zatvoreni.

*Primer*

task.set\_forms\_container($(#content), {

splash\_screen: '<h1 class=text-center>Jam.py Demo Application</h1>'

});

##### upload(*options*)

Koristi *upload* metod za izbor fajla u File open dijalog boksu i otpremanje u *static/files* folder u server folderu. Kada se upisuje fajl na server, ime bajla se menja sa *Werkzeug* *secure\_filename* funkcijom i tada je tekući datum dodan. Vidi <http://werkzeug.pocoo.org/docs/0.14/utils/>. *options* parametar je objekat koji može imati sledeće atribute:

* callback – je *callback* funkcija koja se izvršava kada je fajl otpremljen.
* show\_progress – ako je *true* u fajl koji se otprema je veliki, progres bar će biti prikazan. Podrazumevano je *true*.
* accept - atribut spec. tipove fajlova koji mogu biti otpremljeni kroz fajl otpremu, vidi Accept string.

Napomena: Obratite pažnju da *accept* atribut spec. samo tipove fajlova koji mogu biti izabrani od strane korisnika u brauzeru. Server proverava sve otpremljene fajlove u odnosu na *Upload file extensions* atribut Projecktnih parametara.

#### Eventi

##### on\_edit\_form\_close\_query(item)

*on\_edit\_form\_close\_query* event je pokrenut od *close\_edit\_form* metoda objekta.

* item parametar je objekat koji pokreće event.

##### on\_edit\_form\_created(item)

*on\_edit\_form\_created* event je pokrenut od strane *create\_edit\_form* metoda objekta kada je forma kreirana ali još nije prikazana.

* item parametar je objekat koji pokreće event.

Ovaj event, ako je definisan, je pokrenut za svaki objekat taska, čiji *create\_edit\_form* metod je pozvan.

##### on\_edit\_form\_keydown(item, event)

*on\_edit\_form\_keydown* event je pokrenut kada se *keydown* event pojavi u *edit\_form* objekta.

* item parametar je objekat koji pokreće event.
* event je JQuery event objekat.

##### on\_edit\_form\_keyup(item, event)

*on\_edit\_form\_keyup* event je pokrenut kada se *keyup* event pojavi u *edit\_form* objekta.

* item parametar je objekat koji je pokrenuo event.
* event je JQuery event objekat.

##### on\_edit\_form\_shown(item)

*on\_edit\_form\_shown* event je pokrenut od *create\_edit\_form* metoda objekta kada je forma prikazana.

* item parametar je objekat koji je pokrenuo event.

Ovaj event, ako je definisan, je pokrenut za svaki objekat taska, čiji *create\_edit\_form* metod je pozvan.

##### on\_filter\_form\_close\_query(item)

*on\_filter\_form\_close\_query* event je pokrenut od *close\_filter\_form* metoda objekta.

* item parametar je objekat koji je pokrenuo event.

##### on\_filter\_form\_created(item)

*on\_filter\_form\_created* event je pokrenut od *create\_filter\_form* metoda objekta klada je forma kreirana ali još nije prikazana.

* item parametar je objekat koji je pokrenuo event.

Ovaj event, ako je definisan , je pokrenut za svaki objekat taska, čuji *create\_filter\_form* metod je pozvan.

##### on\_filter\_form\_shown(item)

*on\_filter\_form\_shown* event je pokrenut od *create\_filter\_form* metoda objekta kada je forma prikazana.

* item parametar je objekat koji je pokrenuo event.

Ovaj event, ako je definisan, je pokrenut za svaki objekat taska, čiji je *create\_filter\_form* metod pozvan.

##### on\_page\_loaded(task)

*on\_page\_loaded* event je prvi event koji je pokrenut na klijentu. Korist se za inicijalizaciju klijenta.

* task parametar je koren klijentovog task stabla.

##### on\_param\_form\_close\_query(item)

*on\_param\_form\_close\_query* event je pokrenut od *close\_param\_form* metoda.

* item parametar je objekat koji je pokrenuo event..

##### on\_param\_form\_created(item)

*on\_param\_form\_created event* je pokrenut od *create\_param\_form* metoda, koji obično je pozvan od *print* metoda.

* item parametar je objekat koji je pokrenuo event.

##### on\_param\_form\_shown(item)

on\_param\_form\_shown event je pokrenut od *create\_param\_form* metoda, koji je obično, pokrenut od *print* metoda.

* item parametar je objekat koji je pokrenuo event..

##### on\_view\_form\_close\_query(item)

*on\_view\_form\_close\_query* event je pokrenut od *close\_view\_form* metoda objekat.

* item parametar je objekat koji je pokrenuo event.

##### on\_view\_form\_created(item)

*on\_view\_form\_created* event je pokrenut od *view* metoda objekta kada je forma kreirana ali još nije prikazana.

* item parametar je objekat koji je pokrenuo event.

Ovaj event, ako je definisan, pokrenut je za svaki objekat taska, čiji *view* metod je pozvan.

##### on\_view\_form\_keydown(item, event)

*on\_view\_form\_keydown* event je pokrenut kada se *keydown* event pojavi za *view\_form* objekta.

* item parametar je objekat koji je pokrenuo event.
* event je JQuery event objekat.

##### on\_view\_form\_keyup(item, event)

*on\_view\_form\_keyup* event is pokrenut kada se *keyup* event pojavi za *view\_form* objekat.

* item parametar je objekat koji je pokrenuo event.
* event je JQuery event objekat.

##### on\_view\_form\_shown(item)

*on\_view\_form\_shown* event je pokrenut od *view* metoda objekta kada je forma prikazana.

* item parametar je objekat koji je pokrenuo event.

Ovaj event, kada je definisan, biće pokrenut za svaki objekat čiji *view* metod je pozvan.

### *class* Group()

Group klasa se koristi za kreiranje grupe objekata task stabla. Ispod su dati eventi klase. Ona nasledjuje AbstractItem klasu, samim time i njene atribute, metode i evente.

#### Eventi

##### on\_edit\_form\_close\_query(item)

*on\_edit\_form\_close\_query* event pokrenut je od *close\_edit\_form* metoda objekta of the item.

* item parametar je objekat koji pokreće event.

##### on\_edit\_form\_created(item)

*on\_edit\_form\_created* event pokrenut je od *create\_edit\_form* metoda objekta kada je forma kreirana ali još nije prikazana.

* item parametar je objekat koji pokreće event.

Ovaj event, ako je definsan, pokrenut je za svaki objekat grupe, čiji *create\_edit\_form* metod je pozvan.

##### on\_edit\_form\_keydown(item, event)

*on\_edit\_form\_keydown* event pokrenut je kada *keydown* event se pojavi za *edit\_form* objekat.

* item parametar je objekat koji pokreće event.
* event je JQuery event objekat.

##### on\_edit\_form\_keyup(item, event)

*on\_edit\_form\_keyup* event pokrenut je kada *keyup* event se pojavi za *edit\_form* objekta.

* item parametar je objekat koji pokreće event.
* event je JQuery event objekat.

##### on\_edit\_form\_shown(item)

*on\_edit\_form\_shown* event pokrenut je od *create\_edit\_form* metoda objekta kada je forma prikazana.

* item parametar je objekat koji pokreće event.

Ovaj event ako je definisan, pokrenut je za svaki objekat grupe, čiji *create\_edit\_form* metod je pozvan.

##### on\_filter\_form\_close\_query(item)

*on\_filter\_form\_close\_query* event pokrenut je od *close\_filter\_form* metoda objekta.

* item parametar je objekat koji pokreće event.

##### on\_filter\_form\_created(item)

*on\_filter\_form\_created* event pokrenut je od *create\_filter\_form* metoda objekta kada je forma kreirana, ajoš nije prikazana.

* item parametar je objekat koji pokreće event.

Ovaj event, ko je definisan, pokrenut je za svaki objekat grupe, čiji *create\_filter\_form* metod je pozvan.

##### on\_filter\_form\_shown(item)

*on\_filter\_form\_shown* event pokrenut je od *create\_filter\_form* metoda objekat kada je forma prikazana.

* item parametar je objekat koji pokreće event.

Ovaj event, ko je definisan, pokrenut je za svaki objekat grupe, čiji *create\_filter\_form* metod je pozvan.

##### on\_view\_form\_close\_query(item)

*on\_view\_form\_close\_query* event pokrenut je od *close\_view\_form* metoda objekta.

* item parametar je objekat koji pokreće event.

##### on\_view\_form\_created(item)

*on\_view\_form\_created* event pokrenut je od *view* metoda objekat kada je forma kreirana, ali jon nije prikazana.

* item parametar je objekat koji pokreće event.

Ovaj metod, ako je definisan, pokrenut je za svaki objekat grupe, čiji *view* metod je pozvan.

##### on\_view\_form\_keydown(item, event)

on\_view\_form\_keydown event pokrenut je kada se *keydown* event pojavi za *view\_form* objekta.

* item parametar je objekat koji pokreće event.
* event je JQuery event objekat.

##### on\_view\_form\_keyup(item, event)

*on\_view\_form\_keyup event* pokrenut je kada se *keyup* event pojavi za *view\_form* objekta.

* item parametar je objekat koji pokreće event.
* event je JQuery event objekat.

##### on\_view\_form\_shown(item)

*on\_view\_form\_shown* event pokrenut je od *view* metoda objekata kada je forma prikazana.

* item parametar je objekat koji pokreće event.

Ovaj event kada je definisan, pokrenut je za svaki objekat grupe, čiji je *view* metod pozvan.

### *class* Item()

Item klasa se koristi za kreiranje objekata task stabla koji mogu imati pridružene tabele baze podataka. Ispod su atributi, svojstva, metodi i eventi. Klasa nasledjuje AbstractItem klasu i sve njene atribute i metode.

#### Atrubuti i svojstva

##### active

Spec. da li je dataset objekta otvoren. *active* je read only svojstvo klase. *open* metod menja vrednost *active* na *true,* *close* metod ga postavlja na *false*. Kada je dataset otvoren njegovi zapisi se mogu navigovati i podaci mogu biti menjani i promene sačuvane u objektovu tabelu u bazi podataka.

##### can\_modify

Postavi *can\_modify* svojstvo na *false* ako je potrebno zabrana promene na objektu u vizuelnim kontrolama. Kada je *can\_modify* *false, can\_create*, *can\_edit*, *can\_delete* metode vraćaju *false*. Podrazumevano, *can\_modify* svojstvo je *true*.

##### details

Lista svih objekata detalja objekta.

##### edit\_form

Koristi *edit\_form* atribut za dobijanje pristupa na Jquery objekat koji predstavlja edit formu objekta. Kreiran je od *create\_edit\_form* metoda. *close\_edit\_form* metod postavlja *edit\_form* vrednost na *undefined*.

*Primer*

U sledećem kodu, dugmetu definisanom u objektovom edit html šablonu je pridružen klik event:

item.edit\_form.find(#ok-btn).on('click.task',

function() {

item.apply\_record();

}

);

##### edit\_options

*edit\_options* atribut je skup opcija koje odredjuju kako će *edit* forma biti prikazana na strani. Ove opcije se postavljaju u Edit Form Dialog u AppBuilder. Možete promeniti *edit\_options* u *on\_edit\_form\_created* event heandleru objekta.

*edit\_options* je objekat sa sledećim atributima:

|  |  |
| --- | --- |
| **Opcija** | **Opis** |
| width | Širina modalne forme, podrazumevano *600 px*, |
| title | Naslov forme, podrazumevano *item\_caption* objekata |
| form\_border | Ako je *true*, ivica će biti prikazana oko forme |
| form\_header | Ako je *true*, heder forme će biti kreiran i prikazan. Sadržaće naslov i različitu dugmad |
| history\_button | Ako je *true* i *Saving change history* je *true*, history dugme će biti prikazano u form hederu |
| close\_button | Ako je *true*, *close* dugme biće kreirano u gornjem desnom uglu forme |
| close\_on\_escape | Ako je *true*, pritisak na Escape će izvršiti *close\_edit\_form* metod da zatvori formu |
| edit\_details | Lista imena *detalja*, koji će biti raspoloživi za editovanje u *edit* formi, ako edit šablon sadrži div sa klasom *edit-detail* (Podrazumevani šablon edit forme ima taj div) |
| detail\_height | Visina forme detalja prikazana u *view* formi, ako nije spec. visina tabele detalja je *200px* |
| fields | Spec. listu polja koje će *create\_inputs* metod koristiti, ako *fields* atribut njegovih opcija nije spec.. |
| template\_class | Ako je spec., div sa tom klasom će biti tražen u task templates atributu i koristiće ga za formin šablon kod kreiranja forme. Ovaj atribut mora biti postavljen pre kreiranja forme. |
| modeless | Ako je postavljen, edit forme će biti kreirane modeless, inače – modal. |

*Primer*

function on\_edit\_form\_created(item) {

item.edit\_options.width = 800;

item.edit\_options.close\_on\_escape = false;

}

##### fields

Lista svih polja objekta.

*Primer*

function customer\_fields(customers) {

customers.open({limit: 1});

for (var i = 0; i < customers.fields.length; i++) {

console.log(customers.fields[i].field\_caption,

customers.fields[i].display\_text);

}

}

##### filter\_form

Koristi *filter\_form* atribut za pristup Jquery objektu koji predstavlja filter formu objekta. On je kreiran od *create\_filter\_form* metoda. *close\_filter\_form* metod postavlja *filter\_form* vrednost na *undefined*.

*Primer*

U sledećem primeru dugmetu definisanom u objektovom filter *html* šablonu je pridružen klik event:

item.filter\_form.find(#cancel-btn).on('click',

function() {

item.close\_filter()

}

);

##### filter\_options

Koristi *filter\_options* atribute za spec. parametara modalne filter forme. *filter\_options* je objekat koji ima sledeće atribute:

* width – širina modalne forme, podrazumevano 560 px,
* title – naslov filter forme,
* close\_button – ako je *true, close* dugme će biti kreirano u gornjem desnom uglu forme, podrazumevano je *true*,
* close\_caption – ako je *true* i *close\_button* je *true*, prikazaće ‘Close - [Esc]’ blizu dugmeta,
* close\_on\_escape – ako je *true*, pritisak na Escape će pokrenuti *close\_filter\_form* metod.
* close\_focusout – ako je *true*, *close\_filter\_form* metod će biti pozvan kada forma izgubi fokus,
* template\_class – ako je spec., div sa tom klasom će biti tražen u task templates atributu i korišćen za formin *html* šablon pri kreiranju forme.

*Primer*

function on\_filter\_form\_created(item) {

item.filter\_options.width = 700;

}

##### filtered

Spec. da li ili ne filtriranje je aktivno za dataset. Ako je *filtered* je *true*, tada je filtriranje aktivno. Da bi primenili filterske uslove spec. u *on\_filter\_record* event hendlere, postavi *filtered* na *true*.

##### filters

Lista svih filter objekata na objektu.

*Primer*

function invoices\_filters(invoices) {

for (var i = 0; i < invoices.filters.length; i++) {

console.log(invoices.filters[i].filter\_caption, invoices.filters[i].value);

}

}

##### item\_state

Ispitaj *item\_state* za odredjivanje tekućeg operativnog moda objekta. *item\_state* odredjuje šta može biti radjeno sa podacima objektovog dataseta, kao što je editovanje postojećih zapisa ili insertovanje novih. *item\_state* se konstantno menja kako aplikacija procesira podatke.

Otvaranje objektovog dataseta menja *item\_state* iz *inactive* u *browse*. Aplikacija može zvati *edit* koji će postaviti *item\_state* u *edit* stanje, ili zvati *insert* ili *append*, koji će postaviti *item\_state* u u *insert* stanje. *Posting* ili *edit\_cancel*, *insert* ili *delete*, menja *item\_state* iz tekućeg u *browse* stanje. Zatvaranje dataseta menja *item\_state* u *inactive*.

Da proverite vrednost *item\_state* koristite sledeće metode:

* is\_new - indicira da li je objekat u insert stanju
* is\_edited – indicira dali je objekat u *edit* stanju
* is\_changing – indicira da li je objekat u *edit* ili *insert* stanju

*item\_state* vrednost može biti:

* 0 - inactive state,
* 1 - browse state,
* 2 - insert state,
* 3 - edit state,
* 4 - delete state

Objektov task atribut ima *consts* objekat koji definiše sledeće atribute:

* STATE\_INACTIVE: 0,
* STATE\_BROWSE: 1,
* STATE\_INSERT: 2,
* STATE\_EDIT: 3,
* STATE\_DELETE: 4

Tako dali je objekat je u insert stanju može biti provereno kao:

item.item\_state === 2

ili:

item.item\_state === item.task.consts.STATE\_INSERT

ili:

item.is\_new()

##### log\_changes

Indicira da li se promene na datasetu objekta upisuju. Kada *log\_changes* je *true* (podrazumevano), sve promene se upisuju. Kasnije one mogu biti primenjene na aplikativni server pozivanjem *apply* metoda. Kada *log\_changes* je *false*, promene na podacima se ne upisuju i ne mogu biti primenjene na aplikativni server.

##### lookup\_field

Koristite *lookup\_field* da proverite da li je objekat kreiran za izbor vrednosti za lookup field.

*Primer*

function on\_view\_form\_created(item) {

item.table\_options.multiselect = false;

if (!item.lookup\_field) {

var print\_btn = item.add\_view\_button('Print', {image: 'icon-print'}),

email\_btn = item.add\_view\_button('Send email', {image: 'icon-pencil'});

email\_btn.click(function() { send\_email() });

print\_btn.click(function() { print(item) });

item.table\_options.multiselect = true;

}

}

##### paginate

Atribut *paginate* određuje ponašanje tabele kreirane metodom *create\_table*. Kada je paginacija postavljena na *true*, kreira se paginator i tabela izračunava broj prikazanih redova na osnovu njene visine. Tabela će interno manipulisati parametrima *limit* i *offset* *open* metode, u zavisnosti od njene i visine trenutne stranice, ponovo otvarajući dataset podataka kada se stranica promeni. Ako je vrednost *paginate* *false*, tabela će prikazati sve dostupne zapise dataseta.

##### permissions

Postavite atribute *permissions* svojstva da zabranite promene objekta u vizuelnim kontrolama. *Permissions* svojstvo je objekat sa sledećim atributima:

* can\_create
* can\_edit
* can\_delete

Podrazumevano, atributi ovi atributi su postavljeni na *true*. Kada su ovi atributi postavljeni na *false*, odgovarajući

* can\_create,
* can\_edit,
* can\_delete

metode vraćaju *false*.

##### read\_only

Pročitajte read\_only svojstvo da biste utvrdili da li se podaci mogu modifikovati u kontrolama svesnim na podatke objekta. Podesite read\_only svojstvo na true da biste sprečili izmenu podataka u kontrolama koje su svesne podataka. Kada dodelite vrednost svojstvu *read\_only*, aplikacija postavlja svojstvo *read\_only* svih detalja i svojstvo *read\_only* svakog polja na tu vrednost. Ako korisnička uloga zabranjuje uređivanje zapisa, read\_only uvek vraća true.

*Primer*

U ovom primeru prvo postavljamo *read\_only* atribut invoices objekta na *true*. Sva polja invoices objekta i objekta invoice\_table su *read\_only*. Posle toga omogućujemo korisniku da edituje customer polje i invoice\_table detalje.

function on\_edit\_form\_created(item) {

item.read\_only = true;

item.customer.read\_only = false;

item.invoice\_table.read\_only = false;

}

##### rec\_count

Pročitajte svojstvo *rec\_count* da biste dobili broj zapisa u u vlasništvu objektovog dataseta. Ako modul definiše *event hendler on\_filter\_record* i postavljen je *filtered* atribut, ovo svojstvo izračunava broj zapisa koji zadovoljavaju ovaj filter, u protivnom se *record\_count* metoda koristi za izračunavanje broja zapisa.

*Example*

function edit\_invoice(invoice\_id) {

var invoices = task.invoices.copy();

invoices.open({ where: {id: invoice\_id} }, function() {

if (invoices.rec\_count) {

invoices.edit\_record();

}

else {

invoices.alert\_error('Invoices: record not found.');

}

});

}

##### rec\_no

Ispitajte *rec\_no* svojstvo da odredite redni broj tekućeg zapisa u datasetu objekta. *rec\_no* može biti postavljen na spec. broj zapisa zbog repozicioniranja kursora dataseta.

*Example*

function calculate(item) {

var subtotal,

tax,

total,

rec;

if (!item.calculating) {

item.calculating = true;

try {

subtotal = 0;

tax = 0;

total = 0;

item.invoice\_table.disable\_controls();

rec = item.invoice\_table.rec\_no;

try {

item.invoice\_table.each(function(d) {

subtotal += d.amount.value;

tax += d.tax.value;

total += d.total.value;

});

}

finally {

item.invoice\_table.rec\_no = rec;

item.invoice\_table.enable\_controls();

}

item.subtotal.value = subtotal;

item.tax.value = tax;

item.total.value = total;

}

finally {

item.calculating = false;

}

}

}

##### selections

Atribut *selections* čuva listu vrednosti polja primarnih ključeva. Kada je višestruki ček boks potvrdjen na Layout tabu u View Form Dialogu ili *multiselect* atribut *table\_options* postavljen iz koda, pojavljuje se ček boks u najlevljoj koloni forme, i svaki put kada korisnik klikne na ček boks *selections* atribut je promenjen. On takodje može biti promenjen iz koda, korišćenjem metoda *add* ili *remove* niza.

*Primer*

U ovom primeru, funkcija *send\_email*, na klijentu, koristi atribut *selections* Customers objekta, da bi dobili niz vrednosti polja primarnog ključa izabranih korisnika i da bi im poslali mejl sa funkcijom *send\_email* koja je definisana u modulu servera objekta koristeći *server* metod.

function send\_email(subject, message) {

var selected = task.customers.selections;

if (!selected.length) {

selected.add(task.customers.id.value);

}

item.server('send\_email', [selected, subject, message],

function(result, err) {

if (err) {

item.alert('Failed to send the mail: ' + err);

}

else {

item.alert('Successfully sent the mail');

}

}

);

}

Na serveru, ovaj niz se koristi za vraćanje informacije o izabranim kupcima korišćenjem *open* metoda.

import smtplib

def send\_email(item, selected, subject, mess):

cust = item.task.customers.copy()

cust.set\_where(id\_\_in=selected)

cust.open()

to = []

for c in cust:

to.append(c.email.value)

# code that sends email

##### table\_options

*table\_options* atribut je skup opcija koje odredjuju kako će tabela *view* forme biti prikazana. Ocije tu definisane se koriste od *create\_table* metoda ako njen *options* parametar ne definiše odgovarajuće opcije. Ove opcije su postavljene u Layout tabu View Form Dialog u AppBuilderu. Možete menjati *table\_options* u *on\_view\_form\_created* event hendleru objekta.

*table\_options* parameter je objekat koji može imati sledeće atribute:

|  |  |
| --- | --- |
| **Opcija** | **Opis** |
| row\_count | Spec. broj vrsta prikazanih u tabeli. |
| height | Ako *row\_count* nije spec. odredjuje visinu tabele, podrazumevano je *480px*. Tabela na kreiranju računa broj vrsta koji može biti prikazan (row\_count), zasnovan na vrednosti ovog parametra. |
| fields | Lista imena polja, koja ako je spec. definiše kolone koje će biti kreirane, ako nije (podrazumevano) tada *fields* atribut iz *view\_options* se uzima. |
| title\_line\_count | Spec. broj linija teksta prikazanog u vrsti naslova tabele, ako je 0, visina vrste je definisana sadržajem ćelije naslova. |
| row\_line\_count | Spec. broj linija teksta prikazanog u vrsti tabele, ako je 0, visina vrste je odredjena sadržajem ćelija vrste. |
| expand\_selected\_row | Ako *row\_line\_count* je postavljen i *expand\_selected\_row* je veći od 0, spec. minimalni broj linija teksta prikazanog u izabranoj vrsti tabele. |
| title\_word\_wrap | Spec. da li je kolona naslova može da se lomi. |
| column\_width | Širina kolona se računa od Web Browser. Možete koristiti ovu opciju da forsirate širine kolona. Opcija je objekt, ključ vrednosti su imena polja, a vrednosti su širine kolona kao CSS units. |
| editable\_fields | Lista polja koja mogu biti editovana u tabeli. |
| selected\_field | Ako je *editable\_fields* postavljeno, spec. ime polja čija kolona je izabrana, kada se izabrana vrsta promeni. |
| sortable | Ako je opcija postavljena, moguće je sortirati zapise klikom na heder kolone. Kada *sort\_fields* opcija nje postavljena (podrazumevano), korisnik može sortirati po bilo kojoj koloni odnosno polju, inače, samo na poljima koji su spec u *sort\_fields* opciji. |
| sort\_fields | Lista imena polja po kojima tabela može biti sortirana, klikom na heder kolone. Ako je objekat detalj operacija se izvodi na klijentu, inače sortiranje se radi na serveru. (open metod se koristi interno). |
| summary\_fields | Lista imena polja. Kada je lista spec., tabela izračunava sume za numerička polja i prikazuje ih u footeru tabele, za nenumerička polja prikazuje broj zapisa. |
| freeze\_count | Integer vrednost. Ako je veća od nule. Spec. broj kolona koje će biti zamrznute – one neće skrolovati kada tabela skroluje horizontalno. |
| show\_hints | Ako je *true*, tooltip će biti prikazan kada korisnik nadnese miša preko ćelije kolone i tekst ćelije se ne vidi kompletno. Podrazumevano je *true*. |
| hint\_fields | Lista imena polja, ako je spec. tooltip će biti prikazan samo na poljima it ove liste, bez obzira na vrednost na vrednost *show\_hints* opcije. |
| on\_click | Spec. funkciju, koja će biti izvršena kada korisnik klikne na vrstu tabele. Objekat će biti prosledjen kao parametar funkciji. |
| on\_dblclick | Spec. funkciju, koja će biti izvršena kada korisnik dva puta klikne na vrstu tabele. Objekat će biti prosledjen kao parametar funkciji. |
| dblclick\_edit | Ako je vrednost opcije postavljena na *true* i *on\_dblclick* opcija nije postavljena, edit forma će biti prikazana kada korisnik dva puta klikne na vrstu tabele. |
| multiselect | Ako je opcija postavljena, najlevlja kolona sa ček boksovima će biti kreirana za izbor zapisa. Potom kada korisnik klikne na čekboks, vrednost polja primarnog ključa je dodata ili izbrisana iz *selections* atributa. |
| select\_all | Ako je *true*, meni će se pojaviti u najlevljoj koloni hedera tabele, koji će omogućiti korisniku da izabere sve zapise koji odgovaraju tekućem filteru i vrednosti za pretragu. |
| row\_callback | *Callback* funkcija pozvana svaki put kada su polja zapisa promenjena. Dva parametra su prosledjena funkciji, objekat klon, čiji zapisi su promenjeni i JQuery objekat odgovarajuće vrste tabele. Budite oprezni, objekat prosledjen funkciji ne može biti objekat sam, nego objekat klon, koji deli isti dataset sa objektom. |

*Primer*

function on\_view\_form\_created(item) {

item.table\_options.row\_line\_count = 2;

item.table\_options.expand\_selected\_row = 3;

}

Sledeća dva primera rade istu stvar:

item.invoice\_table.create\_table(item.view\_form.find('.view-detail'), {

height: 200,

summary\_fields: ['date', 'total'],

});

item.invoice\_table.table\_options.height = 200;

item.invoice\_table.table\_options.summary\_fields = ['date', 'total'];

item.invoice\_table.create\_table(item.view\_form.find('.view-detail'));

##### view\_form

Koristite *view\_form* atribut da pristupite Jquery objektu koji predstavlja view formu objekta. Kreirao ga je *view* metod. *close\_view\_form* metod postavlja *view\_form* vrednost na *undefined*.

*Primer*

U sledećem primeru dugme definisano u objektovom *html* šablonu je pridruženo na klik event event:

item.view\_form.find(#new-btn).on('click',

function() {

item.insert\_record();

}

);

##### view\_options

*view\_options* atribut je skup opcija koje odredjuju kako će *view* forma biti prikazana. Ove opcije se postavljaju u View Form Dialog u AppBuilder. Možete promeniti *view\_options* u *on\_view\_form\_created* event hendleru objekta.

*view\_options* je objekat koji ima sledeće atribute:

|  |  |
| --- | --- |
| **Opcija** | **Opis** |
| width | Širina modalne forme, podrazumevano je 600 px. |
| title | Naslov forme, podrazumevano je vrednost *item\_caption* atributa. |
| form\_border | Ako je *true*, granična ivica će biti prikazana oko forme. |
| form\_header | Ako je *true*, heder forme će biti kreiran i prikazan. Sadržaće naslov forme i različiu dugmad. |
| history\_button | Ako je *true* i *Saving change history* je omogućeno, *history* dugme će biti prikazano u hederu forme. |
| refresh\_button | Ako je *true*, *refresh* dugme će biti kreirano u hederu forme, koje omogućava korisnicima da refresh stranu slanjem zahteva na server. |
| enable\_search | Ako je *true*, *search* ulaz će biti kreiran u hederu forme. |
| search\_field | Imepolja koje će biti podrazumevano za pretragu. |
| enable\_filters | Ako je *true* i postoje vidljivi filteri, filtersko dugme će biti kreirano u hederu forme |
| close\_button | Ako je *true*, *close* dugme će biti kreirano u gornjem levom uglu forme. |
| close\_on\_escape | Ako je *true*, pritisak na Escape će izvršiti *close\_view\_form* metod da zatvori formu. |
| view\_details | Lista imena detalja, koja će biti prikazana u *view* formi, ako view form šablon sadrži div sa klasom *view-detail* (podrazumevani view form šablon ima taj div) |
| detail\_height | Visina detalja prikazana u *view* formi, ako nije spec. visina tabele detalja je is 200px |
| modeless | Ako je *true*, forma će biti prikazana kao modeless, inače modal. |
| template\_class | Ako je spec., div sa tom klasom će biti tražen u task templates atributu i korišćen za formin *html* šablon kada kreirate formu. Ovaj atribut mora biti postavljen pre nego što je forma kreirana. |

*Primer*

function on\_view\_form\_created(item) {

item.view\_options.width = 800;

item.view\_options.close\_button = false;

item.view\_options.close\_on\_escape = false;

}

##### virtual\_table

Koristite read-only *virtual\_table* svojstvo da pronadjete da li objekat ima odgovarajuću tabelu u projektnoj bazi podataka. Ako *virtual\_table* je *True* nema odgovarajuće tabele u bazi podataka. Možete koristiti ovakav objekat da radite sa datasetom u memoriji ili koristiti module za pisanje koda. Pozivanje *open* metoda, kreira prazan dataset, a pozivanje *apply* metode ne radi ništa.

#### Metodi

##### add\_edit\_button(*text*, *options*)

Koristite *add\_edit\_button* da dinamički dodate dugme na *edit* formu.

* Ovaj metod ima iste parametre kao add\_view\_button metod.

##### add\_view\_button(*text*, *options*)

Koristite *add\_view\_button* da dinamički dodate dugme na *view* formu. Ovaj metod se obično koristi u *on\_view\_form\_created* eventima. Sledeći parametri su prosledjeni metodi:

* text - text koji će biti prikazan na dugmetu
* options – *options* dodatno spec. svojstva dugmeta. *Options* parametar je objekat koji može imati sledeće atribute:
  + parent\_class\_name - ime klase roditeljskog elementa, podrazumevano je ‘form-footer’.
  + btn\_id - *id* atribute dugmeta
  + btn\_class – klasa dugmeta
  + type – spec. top (boju) dugmeta, može biti jedna od sledećih vrednosti:
    - primary
    - success
    - info
    - warning
    - danger
* image – klasa ikona, jedna od ikona iz *Glyphicons* <http://getbootstrap.com/2.3.2/base-css.html>
* secondary – Ako je *true*,dugme će biti desno poravnato ako *Buttons on top* atribut View Form Dialog je postavljen, inače levo poravnat.
* expanded – Ako je *true* dugme će imati klasu ‘expanded-btn’ koja definiše *min-width* na *120px*, podrazumevano je *true.*

Metoda vraća JQuery objekat dugmeta.

*Primeri*

function on\_view\_form\_created(item) {

var btn = item.add\_view\_button('Select', {type: 'primary'});

btn.click(function() {

item.select\_records('track');

});

}

function on\_view\_form\_created(item) {

if (!item.view\_form.hasClass('modal')) {

var print\_btn = item.add\_view\_button('Print', {image: 'icon-print'}),

email\_btn = item.add\_view\_button('Send email', {image: 'icon-pencil'});

email\_btn.click(function() { send\_email() });

print\_btn.click(function() { print(item) });

}

}

##### append()

Otvara novi, prazni zapis na kraju dataseta. Posle poziva *append*, aplikacija može omogućiti korisnicima da unose podatke u polja zapisa, i mogu zatim da *post* promene objektovog dataseta koristeći *post* metod, i tada *apply* promene na objektovu tabelu u bazi podataka, koristeći *apply* metod.

*append* metod

* provera da li je dataset *active*, inače podiže *izuzetak*
* ako je objekat detalj, provera da li je master objekat u *edit* ili *insert* stanju, inače podiže izuzetak
* ako objekat nije detalj, proverava da li je u *browse* stanju, inače podiže izuzetak
* pokreće *on\_before\_append* event hendler ako je dfinisan za objekat
* otvara novi, prazni zapis na kraju dataseta
* postavlja objekat u *insert* stanje
* pokreće *on\_after\_append* event hendler ako je definisan za objekat
* ažurira kontrole svesne podataka

##### append\_record(*container*)

Otvara novi, prazni zapis na kraju dataseta, i kreira *edit\_form* za vizuelno editovanje zapisa.

* Ako je *container* parametar (Jquery objekat DOM elementa) spec. html šablon edit forme je insertovan u kontejner.
* Ako *container* parametar nije spec. ali
  + *modeless* atribut forme je postavljen, u Edit Form Dialog ili
  + *modeless* atribut *edit\_options* je postavljen programski i
  + task ima *forms\_in\_tabs* atribut postavljen i
  + aplikacija nema modalne forme

modeless edit forma će biti kreirana u novom tabu *forms\_container* objekta taska. U svim drugim slučajevima biće kreirana modalna forma.

Ako je dodavanje zapisa omogućeno u modeless modu, aplikacija poziva copy metod da kreira kopiju objekta. Ta kopija će biti korišćena za dodavanje zapisa.

*append*\_*record* metod:

* poziva *can\_create* metod da proveri da li korisnik ima prava da doda zapis, ako ne vraća,
* provera da li je objekat u *edit* ili *insert* stanju, ako nije poziva *append* metod da doda zapis,
* poziva *create\_edit\_form* metod da kreira formu za vizuelno editovanje zapisa.

##### apply(callback, params, async)

Šalje sve promenjene, insertovane i obrisane zapise iz dataseta objekta na aplikacioni server za upis u bazu podataka. *apply* metod može imati sledeće parametre:

* callback - ako *callback* parametar nije prisutan and *async* parametar je *false* ili *undefined*, zahtev na server je poslan sinhrono, inače, zahtev je izvršen asinhrono i posle odgovora, *callback* je izvršen.
* params – objekat spe. korisnički definisane parametre, koji kogu biti korišćeni u on\_apply event hendleru za neko dodatno procesiranje
* async: ako je *true*, i *callback* parameter nedostaje, zahtev je izvršen asinhrono.

Poredak parametara nije bitan.

apply metod:

* proverava da li je objekat detalj, ako jeste vraća (master objekat upisuje promene na detaljima)
* proverava da li je objekat u *edit* ili *insert* stanju, ako jeste post zapis
* proverava da li *change log* ima promena, ako nema, izvršava *callback* ako je definisan i vraća
* pokreće *on\_before\_apply* event hendler ako je definisan za objekat
* šalje promene na server
* server na prijemu zahteva proverava da li je *on\_apply* event hendler definisan za objekat, ako jeste, izvršava ga, inače generiše i izvršava sql upit za upis promena u bazu podataka, vidi *on\_apply*
* kada generiše sql upit, proverava da li korisnik, koji je poslao zahtev ima prava da pravi promene, inače podiže izuzetak
* piše promene u bazu podataka
* posle pisanja promena u bazu podataka, server šalje klijentu rezultat izvršenja
* ako je izuzetak podignut za vreme operacie na serveru, klijent baca izuzetak, pre bacanja izuzetka, ako je *callback* definisan, on je pozvan i greška je prosledjena *callback* funkciji kao parametar.
* klijent, u zavisnosti od rezultata, ažurira *change log*
* pokreće *on\_after\_apply* event hendler ako je definisan za objekat
* ako je *callback* parametar prosledjen, poziva ga.

Napomena: Server pre nego što novi zapis upiše u tabelu baze podataka, generiše vrednosti za polja primarnih ključeva. Klijent ažurira te ključeva, na osnovu informacija primljenim od servera. Ako vi promenite vrednosti polja u *on\_apply event handler*, te promene se neće reflektovati na klijentu. Možete ažurirati sami zapise, pozivom *refresh\_record* metoda.

*Primer*

var self = this;

this.apply(function(err) {

if (err) {

self.alert\_error(err);

}

else {

//some code to execute after appling changes

}

});

##### apply\_record()

Upisuje promene u aplikacioni dataset *apply\_record* metod

* poziva *apply* dapiše promene u dataset.
* poziva *close\_edit\_form* metod da uništi *edit\_form.*

##### assign\_filters(*item*)

Koristite *assign\_filters* da postavite vrednosti filtera objekta na vrednosti filtera item parametra.

*Primer*

function calc\_footer(item) {

var copy = item.copy({handlers: false, details: false});

copy.assign\_filters(item);

copy.open(

{fields: ['subtotal', 'tax', 'total'],

funcs: {subtotal: 'sum', tax: 'sum', total: 'sum'}},

function() {

var footer = item.view\_form.find('.dbtable.' + item.item\_name + ' tfoot');

copy.each\_field(function(f) {

footer.find('div.' + f.field\_name)

.css('text-align', 'right')

.css('color', 'black')

.text(f.display\_text);

});

}

);

}

##### bof()

Testirajte *bof* (*beginning of file*) metod da odredite poziciju kurzora u datasetu objekta. Ako *bof* vrati *true*, kurzor je nedvosmisleno na poziciji pre prvog zapisa u datasetu. *bof* vraća *false* u svim drugim slučajevima.

Napomena: Ako i *eof* i *bof* vraćaju *true*, dataset objekta je prazan.

##### calc\_summary(*detail*, *fields*)

Koristite *calc\_summary* metod da izračunate sumu za polja objekta detalja i sačuvate vrednost u poljima master objekta u *on\_detail\_changed* event hendleru.

* detail parametar je detalj objekat za polja na kojima se sume računaju.
* fields parametar je objekat koji definiše odnos izmedju polja master i objekta detalja.

Ključevi *fields* objekta su polja *master* objekta, vrednosti su odgovarajuća polja objekta *detalja*. Ako je polje detalja numeričko, suma će biti izračunata, inače rezultat će biti broj broj zapisa. Vrednost može biti funkcija koja vraća rezultat kalkulacije za jedan zapis detalja.

*Primer*

function on\_detail\_changed(item, detail) {

var fields = [

{total: total},

{tax: tax},

{subtotal: function(d) {return d.quantity.value \* d.unitprice.value}}

];

item.calc\_summary(detail, fields);

}

##### can\_create()

Koristite *can\_create* metod da odredite da li korisnik ima prava da kreira novi zapis. Ovaj metod uzima u obzir korisničke dozvole postavljene u rolama u AppBuilderu kada je projektni *safe mode* parametar postavljen kao i vrednost *permissions* atributa i vrednosti *can\_modify* atributa.

*Example*

if (item.can\_create()) {

item.view\_form.find(#new-btn).on('click',

function() {

item.append\_record();

}

);

}

else {

item.view\_form.find(#new-btn).prop(disabled, true);

}

##### can\_delete()

Koristite *can\_delete* metod da odredite da li korisnik ima prava da obriše zapis u datasetu objekta. Ovaj metod uzima u obzir korisničke dozvole postavljene u rolama u AppBuilderu kada je projektni *safe mode* parametar postavljen kao i vrednost *permissions* atributa i vrednost *can\_modify* atributa.

Example

if (item.can\_delete()) {

item.view\_form.find(#delete-btn).on('click',

function() {

item.delete\_record();

}

);

}

else {

item.view\_form.find(#delete-btn).prop(disabled, true);

}

##### can\_edit()

Koristite *can\_edit* metod da odredite da li korisnik ima prava da edituje zapis dataseta objekta. Ovaj metod uzima u obzir korisničke dozvole postavljene u rolama u AppBuilderu kada je projektni *safe mode* parametar postavljen kao i vrednost *permissions* atributa i vrdnost *can\_modify* atributa.

*Primer*

if (item.can\_edit()) {

item.view\_form.find(#edit-btn).on('click',

function() {

item.edit\_record();

}

);

}

else {

item.view\_form.find(#edit-btn).prop(disabled, true);

}

##### cancel()

Pozovite *cancel* da poništite promene napravljene na jednom ili više polja koja pripadaju trenutnom zapisu, sve dok te promene nisu već objavljene u dataset objekta.

*Cancel*:

* pokreće *on\_before\_cancel* event hendler ako je definisan za objekat.
* poništava promne napravljene na tekućem zapisu i njegovim detaljima ako je zapis bio editovan ili uklanja novi zapis ako je dodan ili insertovan.
* postavlja objekat u *browse* stanje
* pokreće *after\_cancel* event hendler ako je definisan za objekat.
* ažurira kontrole svesne podataka

##### cancel\_edit()

Poništava vizuelno editovanje zapisa. *cancel\_edit* metod

* poziva *close\_edit\_form* metod da uništi *edit\_form.*
* poziva cancel metod da poništi promene napravljene na zapisu.

##### clear\_filters()

Koristite *clear\_filters* da postavite filter vrednosti objekta na *null*.

##### clone(keep\_filtered)

Koristite *clone* metod da kreirate kopiju objekta sa kojim će da deli njegov dataset. Klonirani objekat ima sopstven kursor, tako da možete navigovati a kurzor pozicija objekta neće se promeniti.

* keep\_filtered – postavite na *true* ako želite da klon ima iste lokalne filtere kao objekat.

*Primer*

function calc\_sum(item) {

var clone = item.clone(),

result = 0;

clone.each(function(c) {

result += c.sum.value;

})

return result;

}

##### close()

Pozovite da zatvorite dataset objekta. Pošto je dataset zatvoren, *active* svojstvo objekta je *false*.

##### close\_edit\_form()

Koristite *close\_edit\_form* metod da zatvorite *edit* formu objekta. *close\_edit\_form* metod pokreće *on\_edit\_form\_close\_query* event hendler objekta, ako je definisan. Ako je event hendler definisan i

* vraća *true* – forma je uništena, objektov *edit\_form* arribut je postavljen na *undefined* i metod vraća.
* vraća *false* – operacija je prekinuta i metod vraća.

Ako ne vraća vrednost (*undefined*) metod pokreće *on\_edit\_form\_close\_query* grupe koja je vlasnik objekta, ako je definisan za grupu. Ako je event hendler definisan i

* vraća *true* – forma je uništena, objektov *edit\_form* arribut je postavljen na *undefined* i metod vraća.
* vraća *false* – operacija je prekinuta i metod vraća.

Ako ne vraća vrednost (*undefined*) metod pokreće *on\_edit\_form\_close\_query* taska, ako je definisan. Ako je event hendler definisan i

* vraća *true* – forma je uništena, objektov *edit\_form* arribut je postavljen na *undefined* i metod vraća.
* vraća *false* – operacija je prekinuta i metod vraća.

Ako nema definisanog event hendlera ili nijedan od tih event hendlera ne vraća *false*, forma je uništena i *edit\_form* atribute objekta je postavljen na *undefined*.

##### close\_filter\_form()

Koristite *close\_filter\_form* metod da zatvorite *filter* formu objekta. *close\_filter\_form* metod pokreće *on\_filter\_form\_close\_query* event hendler objekta, ako je definisan. Ako je event hendler definisan i

* vraća *true* – forma je uništena, objektov *filter\_form* atribut je postavljen na *undefined* i metod vraća.
* vraća *false* – operacija je prekinuta i metod vraća,

Ako ne vraća vrednost (*undefined*) metod pokreće *on\_filter\_form\_close\_query* grupe koja je vlasnik objekta, ako je definisan za grupu. Ako je event hendler definisan i

* vraća *true* – forma je uništena, objektov *filter\_form* arribut je postavljen na *undefined* i metod vraća.
* vraća *false* – operacija je prekinuta i metod vraća.

Ako ne vraća vrednost (*undefined*) metod pokreće *on\_filter\_form\_close\_query* taska, ako je definisan. Ako je event hendler definisan i

* vraća *true* – forma je uništena, objektov *filter\_form* arribut je postavljen na *undefined* i metod vraća.
* vraća *false* – operacija je prekinuta i metod vraća,

Ako nema definisanog event hendlera ili nijedan od tih event hendlera ne vraća *false*, forma je uništena i *filter\_form* atribute objekta je postavljen na *undefined*.

##### close\_view\_form()

Koristite *close\_view\_form* metod da zatvorite *view* formu objekta. *close\_view\_form* metod pokreće on\_view\_form\_close\_query event hendler objekta, ako je definisan. Ako je event hendler definisan i

* vraća *true* – forma je uništena, objektov *view\_form* atribut je postavljen na *undefined* i metod vraća.
* vraća *false* – operacija je prekinuta i metod vraća.

Ako ne vraća vrednost (*undefined*) metod pokreće *on\_view\_form\_close\_query* grupe koja je vlasnik objekta, ako je definisan za grupu. Ako je event hendler definisan i

* vraća *true* – forma je uništena, objektov *view\_form* arribut je postavljen na *undefined* i metod vraća.
* vraća *false* – operacija je prekinuta i metod vraća.

Ako ne vraća vrednost (*undefined*) metod pokreće *on\_view\_form\_close\_query* taska, ako je definisan. Ako je event hendler definisan i

* vraća *true* – forma je uništena, objektov *viwe\_form* arribut je postavljen na *undefined* i metod vraća.
* vraća *false* – operacija je prekinuta i metod vraća.

Ako nema definisanog event hendlera ili nijedan od tih event hendlera ne vraća *false*, forma je uništena i view*\_form* atribute objekta je postavljen na *undefined*.

##### copy(*options*)

Koristite *copy* za kreiranje kopije objekta. Kreirana kopija nije dodata u task stablo i biće uništena kada više ne bude trebala. Svi atributi kopije objekta su definisani kao u vreme učitavanja task stabla kada je aplikacija startovala.

* options parametar nadalje spec. kreiranu kopiju, i može imati sledeće atribute:
  + handlers – ako je *true*, sva podešavanja napravljena u Form Dialogs AppBuildera i sve funkcije i eventi definisani u klijent modulu objekta će takođe biti raspoloživi u kopiji. Podrazumevano je *true*.
  + filters – ako je *true*, the filteri će biti kreirani za kopiju, inače neće biti filtera. Porazumevano je *true*.
  + details – ako je *true*, detalji će biti kreirani za kopiju,inače neće biti detalja. Podrazumevano je true.

*Primer*

Sledeći kod je korišćen u Demo projekat za asinhrono izračunavanje vrednosti polja total, prikazan u footer-u Invoices žurnala:

function on\_filter\_applied(item) {

var copy = item.copy({handlers: false, details: false});

copy.assign\_filters(item);

copy.open(

{fields: ['subtotal', 'tax', 'total'],

funcs: {subtotal: 'sum', tax: 'sum', total: 'sum'}},

function() {

var footer = item.view\_form.find('.dbtable.' + item.item\_name + ' tfoot');

copy.each\_field(function(f) {

footer.find('div.' + f.field\_name)

.css('text-align', 'right')

.css('color', 'black')

.text(f.display\_text);

});

}

);

}

##### create\_detail\_views(*container*)

Koristite *create\_detail\_views* za kreiranje *view* formi detalja objekta. Ti detalji mogu biti spec. u *Edit* *details* atributu Edit Form Dialog ili postavlheni u *edit\_details* atributu *edit*\_*options*. Ovaj metod je obično korišćen u *on\_edit\_form\_created* event hendleru. Sledeći parametri su prosledjeni metodu:

* container - JQuery objekat će sadržavi vie formu detalja, ako nema kontejnera metoda vraća.

Ako postoji više od jednog detalja, metod kreira *view* forme detalja u tabovima. Ako detalji nisu aktivni, metod poziva njihov *open* metod.

*Primer*

function on\_edit\_form\_created(item) {

item.edit\_form.find(#cancel-btn)

.on('click.task', function(e) { item.cancel\_edit(e) });

item.edit\_form.find(#ok-btn)

.on('click.task', function() { item.apply\_record() });

if (!item.master && item.owner.on\_edit\_form\_created) {

item.owner.on\_edit\_form\_created(item);

}

if (item.on\_edit\_form\_created) {

item.on\_edit\_form\_created(item);

}

item.create\_inputs(item.edit\_form.find(.edit-body));

item.create\_detail\_views(item.edit\_form.find(.edit-detail));

return true;

}

##### create\_edit\_form(*container*)

Koristite *create\_edit\_form* metod da kreirate *edit* formu objekta za vizuelno editovanje zapisa.

* Metod pretražuje za html šablon objekta u task *templates* atributu, kreira klon šablona i pridružuje ga u *edit\_form* atribut objekta.
* Ako je container parametar spec. metod ga prazni i dodaje html šablon. Inače kreira modalnu formu i dodaje html šablon.
* Pokreće *on\_edit\_form\_created* taska. Pokreće *on\_edit\_form\_created* grupe koja je vlasnik objekta, ako je definisan za grupu. Pokreće *on\_edit\_form\_created* objekta, ako je deinisan.
* Pridružuje JQuery *keyup* i *keydown* evente na *edit\_form* tako da kada JQuery event prozora se pojavi, *on\_edit\_form\_keyup* i *on\_edit\_form\_keydown* eventi su pokrenuti. Oni su pokrenuti (ako su definisani) na isti način: prvo task event hendler, potom event hendler grupe i na kraju event hendler samog objekta. Posle toga JQuery *stopPropagation* metod eventa je pozvan.
* Ako je forma modalna, prikazuje je. Pre prikazivanja forme metod primenjuje *options* spec. u *edit\_options* atributu.
* Pokreće *on\_edit\_form\_shown* taska. Pokreće *on\_edit\_form\_shown* grupe koja je vlasnik objekta, ako je definisan. Pokreće *on\_edit\_form\_shown* objekta, ako je definisan.

##### create\_filter\_form(*container*)

Koristite *create\_filter\_form* metod da kreirate filter formu objekta za vizuelno editovanje filtera objekta.

* Metod pretražuje *html* šablon objekta u task *templates* atributu, kreira klon šablona i pridružuje ga *filter\_form* atributu objekta.
* Ako je container parametar spec. metod ga prazni i dodaje html šablon. Inače, kreira modal formu i dodaje html šablon na nju.
* Pokreće *on\_filter\_form\_created* taska. Pokreće *on\_filter\_form\_created* grupe koja je vlasnik objekta, ako je definisan za grupu. Pokreće *on\_filter\_form\_created* objekta, ako je definisan.
* Pridružuje JQuery *keyup* i *keydown* evente na *filter\_form* tako da kada je *JQuery* event prozora pojavi, *on\_filter\_form\_keyup* i *on\_filter\_form\_keydown* eventi su pokrećeni. Oni su pokrenuti (ako su definisani) na isti način: prvo task event hendler, potom event hendler grupe i na kraju event hendler samog objekta. Posle toga JQuery *stopPropagation* metod eventa je pozvan.
* Ako je forma modalna, prikazuje je. Pre prikazivanje forme metod primenjuje *options* spec. u *filter\_options* atributu.
* Pokreće *filter\_form\_shown* taska. Pokreće *on\_filter\_form\_shown* grupe koja je vlasnik objekta, ako je definisan. Pokreće *on\_filter\_form\_shown* objekta, ako je definisan.

##### create\_filter\_inputs(*container*, *options*)

Koristite *create\_filter\_inputs* da kreirate vizulne kontrole, (ulaze, ček boksove) za editovanje filtera objekta. Ovj metod se obično koristi u *on\_filter\_form\_created* eventima pokrenutim od *create\_filter\_form* metoda. Sledeći parametri su prosledjeni metodu:

* container - JQuery objekat koji će sadržati vizuelne kontrole, ako je dužina *container* 0 (nema kontejnera), metod vraća.
* options – spec. kako su kontrole prikazane. *options* parametar je objekat koji može imati sledeće atribute:
  + filters - lista imena filtera. Ako je spec., vizuelna kontrola će biti kreirana za svaki filter čije je ime u listi, ako nije spec. (podrazumevano), tada *fields* atribut filter\_options biće korišćen (podrazumevano izlistava sve filtere spec. u AppBuilderu),
  + col\_count – broj kolona koje će biti kreirane za vizuelne kontrole, podrazumevano je 1.
  + label\_on\_top: podrazumevano je *false*, labele su smeštene lefo od kontrola, inače su kreirane iznad kontrola
  + tabindex – ako je spec., to će biti *tabindex* prve vizuelne kontrole, tabindex svih iza kontrola će biti povećan za 1.
  + autocomplete – podrazumevano je *false*.

Pre kreiranja kontrola aplikacija prazni kontejner.

*Primer*

function on\_filter\_form\_created(item) {

item.filter\_options.title = item.item\_caption + ' - filter';

item.create\_filter\_inputs(item.filter\_form.find(.edit-body));

item.filter\_form.find(#cancel-btn).on('click.task', function() {

item.close\_filter()

});

item.filter\_form.find(#ok-btn).on('click.task', function() {

item.apply\_filter()

});

}

##### create\_inputs(*container*, *options*)

Koristite *create\_inputs* za kreiranje kontrola svesnih podataka (ulazi, ček boksovi) za editovanje polja objekta. Ovaj metod se obično poziva u *on\_edit\_form\_created* eventima. Sledeći parametri su prosledjeni metodu:

* container - JQuery objekat koji će sadržati vizualne kontrole, ako je container dužine 0 (nema kontejnera), metoda vraća.
* options – spec. kako su kontrole prikazane. options parametar je objekat koji može imati sledeće atribute:
  + fields – lista imena polja, ako je spec. vizualne kontrole će biti kreirane za svako polje čije ime je u ovoj listi, ako nije spec. tada će *fields* atribut *edit\_options* biti korišćeno(ako je definisano), inače će izgled, kreiran u Edit Form Dialog AppBuildera, biti kreiran
  + col\_count – broj kolona koji će biti kreiran za vizuelne kontrole, podrazumevano je 1.

Pre kreiranja kontrola aplikacija prazni kontejner.

*Primer*

function on\_edit\_form\_created(item) {

item.create\_inputs(item.edit\_form.find(.left-div),

{fields: ['firstname', 'lastname', 'company', 'support\_rep\_id']}

);

}

##### create\_table(*container*, *options*)

Koristite *create\_table* metod da kreirate tabelu koja prikazuje zapise dataseta objekta. Ponašanje tabele je odredjeno *paginate* atributom objekta. Kada je *paginate* tačno, paginator će biti kreiran, koji će interno ažurirati dataset kada se strana menja. Ako je vrednost *paginate* *false*, svi zapisi dataseta objekta će biti prikazani u tabeli. Tabela, kreirana ovim metodom je svesna podataka, kada promenite dataset, te promene će istog trenutka biti vidljive u tabeli. Tako da možete kreirati tabeli i potom pozvati *open* metod.

Sledeći parametri mogu biti prosledjeni metodi:

* container - JQuery objekat koji će sadržati tabelu, ako je dužina kontejnera 0, metoda će vratiti. Pre kreiranja tabele aplikacija prazni kontejner.
* options – spec. način na koji će tabela biti prikazana. Podrazumevano, metoda koristi table\_options koji su postavljeni u View Form Dialog u AppBuilderu. *options* atributi imaju prednost u odnosu na *table\_options* atribute.

*options* parameter je objekat koji može imati ise atribute kao *table\_options*.

*Primer*

function on\_edit\_form\_created(item) {

item.edit\_options.width = 1050;

item.invoice\_table.create\_table(item.edit\_form.find(.edit-detail),

{

height: 400,

editable\_fields: ['quantity'],

column\_width: {track: 60%}

});

}

##### create\_view\_form(*container*)

Koristite *create\_view\_form* metod da kreirate *view* formu objekta.

* Metod pretražuje za html šablon u task *templates* atributu i kreira klon šablona koga dodeljuje *view\_form* atributu objekta. Ako kontejner parametar je spec. metod ga prazni i dodaje html šablon. Inače kreira modalnu formi i dodaje html šablon unutra.
* Pokreće *on\_view\_form\_created* taska. Pokreće *on\_view\_form\_created* grupe koja je vlasnik objekta, ako je definisan. Pokreće *on\_view\_form\_created* objekta, ako je definisan.
* Pridružuje JQuery *keyup* i *keydown* evente na *view\_form* tako da kada se JQuery eventprozora pojavo, *on\_view\_form\_keyup* i *on\_view\_form\_keydown* eventi su pokrenuti. Oni su pokrenuti (ako su definisani): prvo taske event hendler, potom event hendler grupe i na kraju event hendler samog objekta. Posle toga JQuery *stopPropagation* metod eventa je pozvan.
* Ako je forma modalna, prikazuje je. Pre prikazivanja forme metod primenjuje *options* spec. u *view\_options* atributu.
* Pokreće *on\_view\_form\_shown* taska. Pokreće *on\_view\_form\_shown* grupe koja je vlasnik objekta, ako je definisan. Pokreće *on\_view\_form\_shown* objekta, ako je definisan.

##### delete()

Briše aktivni zapis i pozicionira kurzoe na sledeći zapis. *delete* metod:

* proverava da li je dataset objekta *aktivan*, inače podiže izuzetak
* proverava da li dataset objekat nije *prazan*, inače podiže izuzetak
* ako je objekat detalj, proverava da li je master objekat u *edit* ili *insert* stanju, inače podiže izuzetak
* ako objekat nije detalj, proverava da li je u *browse* stanju, inače podiže izuzetak
* pokreće *on\_before\_delete* event hendler ako je definisan za objekat
* postavlja objekat u *delete* stanje
* briše aktivni zapis i pozicionira kurzor na *sledeći* zapis
* postavlja objekat u *browse* stanje
* pokreće *on\_after\_delete* event hendler ako je definisan za objekat
* *ažurira* kontrole svesne podataka

##### delete\_record()

* poziva *can\_delete* metod da proveri da li korisnik ima prava da obriše zapis, ako nema, vraća
* pita korisnika za *potvrdu* operacije
* poziva *delete* metod da obriše zapis
* poziva *apply* metod da upiše promene u aplikacionu bazu podataka.

##### disable\_controls()

Pozovite *disable\_controls* da isključite osvežavanje kontrola svesnih podataka, tako da se one neće ažurirati pri promeni podataka dataseta. Pozovite *enable\_controls* da ponovo uključite osvežavanje kontrola svesnih podataka pridruženih datasetu.

*Example*

function calculate(item) {

var subtotal,

tax,

total,

rec;

if (!item.calculating) {

item.calculating = true;

try {

subtotal = 0;

tax = 0;

total = 0;

item.invoice\_table.disable\_controls();

rec = item.invoice\_table.rec\_no;

try {

item.invoice\_table.each(function(d) {

subtotal += d.amount.value;

tax += d.tax.value;

total += d.total.value;

});

}

finally {

item.invoice\_table.rec\_no = rec;

item.invoice\_table.enable\_controls();

}

item.subtotal.value = subtotal;

item.tax.value = tax;

item.total.value = total;

}

finally {

item.calculating = false;

}

}

}

##### disable\_edit\_form()

Pozovite *disable\_edit\_form* da sprečite bilo koju akciju kada *edit\_form* je vidljiva. Pozovite *enable\_edit\_form* da povo omogućite *edit* formu.

*Primer*

function on\_edit\_form\_created(item) {

var save\_btn = item.add\_edit\_button('Save and continue');

save\_btn.click(function() {

if (item.is\_changing()) {

item.disable\_edit\_form();

item.post();

item.apply(function(error){

if (error) {

item.alert\_error(error);

}

item.edit();

item.enable\_edit\_form();

});

}

});

}

##### each(function(item))

Koristite *each* metod da iterirate preko *zapisa* dataseta objekta. *each()* metod spec. funkciju koja će biti pokrenuta za svaki zapis. Možete prekinuti *each* petlju na partikularnoj iteraciji pravljenjem da *callback* funkcija vrati *false*.

*Primer*

U sledećem primeru *t* i *item.invoice\_table* su reference na isti objekta:

var subtotal = 0,

tax = 0,

total = 0;

item.invoice\_table.each(function(t) {

subtotal += t.amount.value;

tax += t.tax.value;

total += t.total.value;

});

##### each\_detail(function(detail))

Koristite *each\_detail* metod da iterirate preko *zapisa* objekta *detalja* master objekta. *each\_detail()* metod spec. funkciju koja će biti pokrenuta za svaki zapis objekta detalja objekta. (tekući detalj zapis je prosledjen kao parametar). Možete prekinuti *each\_detail* petlju na partikularnoj iteraciji paravljenjem da *callback* vrati *false*.

##### each\_field(function(field))

Koristite *each\_field* metod ad iterirate preko *polja* zapisa objekta. *each\_field()* metod spec. funkciju koja će biti pokrenuta za svako polje (tekuće polje je prosledjeno kao parametar. Možete prekinuti *each\_field* petlju na partkularnoj iteraciji pravljenjem da *callback* vrati *false*.

*Primer*

function customer\_fields(customers) {

customers.open({limit: 1});

customers.each\_field(function(f) {

console.log(f.field\_caption, f.display\_text);

});

}

##### each\_filter(function(filter))

Koristite *each\_filter* metod ad iterirate preko *filtera* objekta. *each\_filter()* metod spec. funkciju koja će biti pokrenuta za svaki filter (tekući filter je prosledjen kao parametar). Možete prekinuti *each\_filter* petlju na partkularnoj iteraciji pravljenjem da *callback* vrati *false*.

*Primer*

function customer\_filters(customers) {

customers.each\_filter(function(f) {

console.log(f.filter\_caption, f.value);

});

}

##### edit()

Omogućava editovanje podataka dataseta. Posle poziva *edit*, aplikacija može omogućiti korisnicima da menjaju podatke u poljima zapisa, i posle toga mogu objaviti promene na dataset objekta korišćenjem *post* metoda, i potom primeniti promene na bazu podataka, korišćenjem *apply* metoda.

*edit* metod:

* proverava da li je dataset *aktivan*, inače podiže izuzetak
* proverava da li dataset nije prazan, inače podiže izuzetak
* provera da li je dataset već u edit stanju, ako jeste vraća
* ako je objekat *detail*, proverava da li je *master* objekat u *edit* ili *insert* stanju, inače podiže izuzetak
* ako objekat nije detalj, proverava da li je u *browse* stanju, inače pože izuzetak
* pokreće *on\_before\_edit* event hendler ako je definisan za objekat
* postavlja objekat u stanje, omogućujujći da apliklacija ili korisnici menjaju polja u zapisu
* pokreće *on\_after\_edit* event hendler ako je definisan za objekat

##### edit\_record(*container*)

Postavlja *trenutni* zapis u *edit* stanje i kreira *edit\_form* za vizuelno editovanje zapisa.

* Ako je container parametar (Jquery objekat DOM elementa) je spec., html šablon edit forme je umetnut u kontejner.
* Ako container parametar nije spec. ali *modeless* atribut forme je postavljen u Edit Form Dialog ili *modeless* atribut *edit\_options* je postavljen programski i aplikacija nema modalnih formi, modeless edit forma će biti kreiranau novom tabu *forms\_container* objekta taska. U svim drugim slučajevima modalna forma će biti kreirana.
* Ako je editovanje omogućeno u *modeless* modu, korisnik mže editovati nekoliko zapisa istovremeno. U tom slučaju aplikacija poziva *copy* metod da kreira kopiju objekta. Ova kopija će biti korišćena za editvanje zapisa. Aplikacija će pozvati *open* metod da dobije zapis sa servera koristeći vrednost polja primarnog ključa kao filter. U slučaju modalnog editovanja aplikacija izvršava *refresh\_record* metod da dobije sa servera poslednje podatke.
* Ako je zaključavanje zapisa omogućeno za objekat, zajedno sa primanjem podataka zapisa sa servera aplikacija prima verziju zapisa.

Potom edit metod:

* Poziva *can\_edit* metod da proveri da li korisnik ima prava da edituje zapis,
* Ako korisnik ima prava da edituje zapis, proverava da li objekat u *edit* ili *insert* stanju, ako nije poziva *edit* metod
* Poziva *create\_edit\_form* metod da kreira formu za vizuelnio editovanje zapisa.

##### enable\_controls()

Pozovite *enable\_controls* da dozvolite prikazivanje u kontrolama svesnih podataka i osvežite posle poziva *disable\_controls*.

##### enable\_edit\_form()

Pozovite *enable\_edit\_form* da omogućite edit formu posle prethodnog poziva *disable\_edit\_form*.

*Example*

function on\_edit\_form\_created(item) {

var save\_btn = item.add\_edit\_button('Save and continue');

save\_btn.click(function() {

if (item.is\_changing()) {

item.disable\_edit\_form();

item.post();

item.apply(function(error){

if (error) {

item.alert\_error(error);

}

item.edit();

item.enable\_edit\_form();

});

}

});

}

##### eof()

Testirajte *eof* (*end-of-file*) da odredite poziciju kurzora u datasetu objekta. Ako *eof* vrati *true*, kurzor je nedvosmisleno na poziciji posle poslednjeg zapisa dataseta. *eof* vraća *false* u svim drugim slučajevima.

Napomena: Ako i *eof* i *bof* vrate *true*, dataset objekta je prazan.

##### field\_by\_name(*field\_name*)

Pozovite *field\_by\_name* da vratite informacije o polju kada je poznato samo ime polja.

* field\_name parametar je ime pstojećeg polja. *field\_by\_name* vraća objekat polja za spec. polje.

Ako spec. polje ne postoji, *field\_by\_name* vraća null.

##### filter\_by\_name(*filter\_name*)

Pozovite *filter\_by\_name* da vratite informacije o filteru kada je poznato samo ime filtera.

* filter\_name parametar je ime postojećeg filter.

*filter\_by\_name* vraća filter objekat za spec. filter. Ako spec. filter na postoji, *filter\_by\_name* vraća *null*.

##### first()

Pozovite *first* da pozicionirate kurzor na prvi zapis u datasetu objekta i napravite ga aktivnim zapisom. *First* metodom postuje sve promene aktivnog zapisa.

##### insert()

Insertuje novi, prazan zapis u datasetu objekta. Posle poziva *insert*-a, aplikacija može omogućiti korisnicima da unose podatke u polja zapisa, i posle toga mogu objaviti promene korišćenjem *post* metoda, a potom primeniti promene na bazu bodataka korišćenjem *apply* metoda.

*insert* metod

* proverava da li je aktivan, inače podiže izuzetak
* ako je objekat detalj, proverava da li master objekat u *edit* ili *insert* stanju, inače podiže izuzetak
* ako objekat nije detalj proverava da li je u *browse* stanju, inače podiže izuzetak
* pokreće *on\_before\_append* event hendler ako je definisan za objekat
* insertuje novi, prazan objekat u dataset objekta
* postavlja objekat u insert stanje
* pokreće *on\_after\_append* event hendler ako je definisan za objekat
* ažurira kontrole svesne podataka

##### insert\_record(*container*)

Otvara novi, prazan zapis na početku dataseta i kreira *edit\_form* za vizuelno editovanje zapisa.

* Ako *container* parametar (JQuery objekat DOM elementa) je spec. html šablon *edit* forme je insertovan u kontejner.
* If *container* parametar nije spec. ali *modeless* atribut forme je postavljen u Edit Form Dialog ili *modeless* atribute *edit\_options* je postavljen programski i task ima *forms\_in\_tabs* atribut postavljen i aplikacija nema modalne forme, *modeless* atribut *edit* forma će biti kreirana u novom tabu *forms\_container* objektu taska. U svim drugim slučajevima modalna forma će biti kreirana.
* Ako je insertovanje zapisa dozvoljeno u *modeless* modu, aplikacija poziva *copy* metod da kreira kopiju objekta. Ova kopija će biti korišćena za insert zapisa.

*insert\_record* metod:

* Poziva *can\_create* metod da proveri da li korisnik ima prava da insertuje zapis, ako nema vraća.
* Proverava da li je objekat u *edit* ili *insert* stanju, ako nije, poziva *insert* metod da insertije zapis
* Poziva *create\_edit\_form* metod da kreira formu za vizuelno editovanje zapisa.

##### is\_changing()

Proverava da li je objekat u edit ili insert stanju i vraća *true* ako jeste. Aplikacija poziva *edit* metod da postavi objekat u *edit* stanje ili append ili insert da postavi objekat u *insert* stanje.

##### is\_edited()

Proverava da li je objekat u *edit* stanju i ako jeste vraća *true*. Aplikacija poziva *edit* metod da postavi objekat u *edit* stanje.

##### is\_modified()

Proverava da li je tekući zapis dataseta objekta promenjen za vreme edit ili insert operacija. Metod vraća false posle izvršenja *post* metoda.

##### is\_new()

Proverava da li je objeakat u *insert* stanju i vraća *true* ako jeste. Aplikacija poziva *append* ili *insert* metod da postave objekat u *insert* stanje.

##### last()

Pozovite last da pozicionirate kursor na poslednji zapis u datasetu objekta i napravite zapis aktivnim.

##### locate(fields, values)

Implementira metod za pretragu dataseta objekta za spec. zapis i pravi pronadjeni zapis aktivnim.

Parametri:

* fields: ime polja, ili lista imena polja
* values: vrednost polja ili liste polja

Metod locira zapis po spec. vrednostima. Vraća true ako zapis ima vrednosti spec u value parametru i repozicionira kursor na locirani zapis. Ako odgovarajući zapis nije pronadjen, pozicija kurzora nije promenjena, metod vraća false.

##### next()

Pozovite *next* da pozicionirate kurzor na sledeći zapis u datasetu objekta i pravi taj zapis aktivnim. *Next* metod postuje sve promene na tekućem aktivnom zapisu.

##### open(options, callback, async)

Pozovite *open* da pošaljete zahtev serveru za dobijanje dataseta objekta. Može imati sledeće parametre:

* options – objekat koji spec. parametre zahteva poslatog na server
* callback – ako nije prisutan, zahtev je poslat na server sinhrono, inače zahtev je izvršen asinhrono i pošto je dataset primeljen, *callback* izvršen
* async – ako je *true*, i *callback* nedostaje, zahtev je izvršen asinhrono

Redosled parametra nije bitan.

Metod incijalizuje polja objekta, formuliše parametre zahteva, zasnovan na opcijama i pokretanju *on\_before\_open* event hendleru ako je definisan za objekat. Posle slanja zahteva na server. Ako *callback* je spec. zahtev je izvršen asinhrono, inače sinhrono.

Server, posle primanja zahteva, proverava da li odgovarajući objekat na serveru (objekat taska stabla sa *id* atributom) ima *on\_open* event hendler, izvršava taj event hendler i vraća rezultat klijentu, inače generiše *select* sql upit, zasnovan na parametrima zahteva, izvršava ovaj upit i vraća rezultat klijentu.

Klijent, posle primanja rezultata zahteva, promene dataseta i postavlja *active* na *true*, *item\_state* na *browse* mod, ide na prvi zapis dataseta, pokreće *on\_after\_open* i *on\_filters\_applied* event hendlere (ako su definisani za objekat), i ažurira kontrole.

Posle poziva *callback* funkciju ako je spec.

*options* objekat parametar mogu imati sledeće atribute:

* expanded – ako je *true*, *select* upit generisan na serveru, imaće *join* klauzulu da bi lookup vrednosti lookup polja bile vraćene, inače lookup vrednosti će biti vraćene. Podrazumevano je *true*.
* fields – koristite da spec. *where* klauzulu *select* upita. Ovaj parametar je lista imena polja. Ako je izostavljen, polja definisana *set\_fields* metod će biti korišćena. Ako je *set\_fields* metod nije bio pozvan pre *open* metoda, sva polja definisana od strane korisnika biće korišćena.
* where – spec. kako će zapisi biti filtrirani u sql upitu. Ovaj objekat je skup key-value parova, gde su ključevi imena polja, koji su praćeni posle duble podcrte filtrirajućim simbolima (vidi Filterirajuće simbole). Ako je ovaj parametar izostavljen, vrednosti definisane *set\_where* metod će biti korišćene. Ako *set\_where* metod nije bio pozvan pre izvršenja *open* metoda, i ako su parametri izostavljeni, tada vrednosti filtera definisane za objekat će biti korišćene za filtriranje zapisa.
* order\_by – da spec. poredak sortiranja. Ovaj parametar je lista imena polja. Ako je znak ‘-’ pre imena polja, tada po tom polju zapisi će biti sortirani u padajućem poretku. Ako je parametr izostavljen, lista definsana po *set\_order\_by* metod će bit korišćenja.
* offset – koristite *offset* da spec. offset prvog zapisa za vraćanje.
* limit – koristite *limit* da ograničite izlaz sql upita na prvih zapisa.
* funcs – ovaj parametar može biti objekat key-value prarova, gde su ključevi imena polja i vrednosti su imena funkcija koja će biti primenjena na polje u select upitu.
* group\_by – koristite *group\_by* da spec. polja za grupisanje rezultata upita. Ovaj parametar mora biti listu imena polja.
* open\_empty – ako je *true*, aplikacija ne šalje zahtev serveru ali inicijalizuje prazan dataset. Podrazumevana vrednost je *false*.
* params – koristite da prosledite neke korisničke definisane opcije za korišćenje u on\_open event hendleru na serveru. Ovaj parametar mora biti objekat key-value parove.

Napomena: Kada je *paginate* atribut objekta postavljen na *true* i tabela je kreirana od *create\_table* metoda, *limit* i *offset* parametri su postavljeni interno tabelom zavisno od njene broja vrste i tekućoj strani.

*Primeri*

function get\_customer\_sales(task, customer\_id) {

var date1 = new Date(new Date().setYear(new Date().getFullYear() - 5)),

date2 = new Date(),

invoices = task.invoices.copy();

invoices.open({

fields: ['customer', 'invoicedate', 'total'],

where: {customer: customer\_id, invoicedate\_\_ge: date1, invoicedate\_\_le: date2},

order\_by: ['invoicedate']

});

}

function get\_customer\_sales(task, customer\_id) {

var date1 = new Date(new Date().setYear(new Date().getFullYear() - 5)),

date2 = new Date(),

invoices = task.invoices.copy();

invoices.set\_fields(['customer', 'invoicedate', 'total']);

invoices.set\_where({customer: customer\_id, invoicedate\_\_ge: date1, invoicedate\_\_le: date2});

invoices.set\_order\_by(['invoicedate']);

invoices.open();

}

function get\_sales(task) {

var sales = task.invoices.copy();

sales.open({

fields: ['customer', 'id', 'total'],

funcs: {'id': 'count', 'total': 'sum'},

group\_by: ['customer'],

order\_by: ['customer']

});

}

##### post()

Piše promenjeni zapis u dataset objekta. Pozovite *post* da sačuvate promene napravljene na zapisu pošto su *insert*, *append* ili *edit* metode bile pozvane.

*Post* metod

* proverava da li je objekat u edit ili insert stanju, inače podiže izuzetak
* pokreće *on\_before\_post* event hendler ako je definisan za objekat
* proverava da li je zapis validan, ako nije podiže izuzetak
* ako objekat ima detalje, objavljuje tekući zapis u detaljima
* dodaje promene u objektov *change log*
* postavlja objekat u *browse* stanje
* pokreće *on\_after\_post* event hendler ako je definisan za objekat.

##### prior()

Pozovite *prior* da pozicionirate kurzor na prethodni zapis u datasetu objektu i napravite ga aktivnim zapisom. *Prior* postuje sve promene na aktivnom zapisu.

##### record\_count()

Pozovite *record\_count* da dobijete broj zapisa u vlasništvu dataseta objekta.

*Primer*

item.open()

if (item.record\_count()) {

// some code

}

##### refresh\_page(*callback*, *async*)

Pozovite *refresh\_page* da pošaljete na server zahtev da dobijete tekuće podatke tekuće strane i osvežite vizualne kontrole. *refresh\_page* metod može imati sledeće parametre:

* callback – ako je parametar nije definisan, zahtev je poslat na server sinhrono, inače, zahtev je izvršen asinhrono i posle toga je callback izvršen
* async – ako je *true,* i *callback* nedostaje, poziv će biti izvršen asinhrono

##### refresh\_record(*options*, *callback*, *async*)

Pozovite *refresh\_record* da pošaljete zahtev za dobijanje tekućih podataka tekućeg zapisa i osveži vizuelne kontrole.

*refresh\_record* metod može imati sledeće parametre:

* callback – ako parametar nije definisan, zahtev je poslan na server snhrono, inače, zahtev je izvršen asinhrono i posle toga *callback* je izvršen
* async – ako je *true*, i *callback* parametar nedostaje, zahtev je izvršen asinhrono
* options – objekat koji može imati atribut *details* – lista *item\_names* sadrži detalje objekta. Ti detalji su takodje osveženi.

Poredak parametara nije bitan.

##### search(field\_name, value, search\_type, callback)

Pozovite *search* da pošaljete na server zahtev za generisanje i izvršenje sql upita za dobijanje svih zapisa koji zadovoljavaju kriterijum pretrage za polje. Upit će takodje zadovoljiti tekući skup *filtera* ili *where* uslove za objekat. Vizuelne kontrole će biti ažurirane sa vraćenim datasetom.

Parametri:

* field\_name – ime polja
* value – vrednost uslova
* search\_type – tip pretrage kao string, (vidi Filter symbol u Filtering records)
* callback - *callback* funkcija koja će biti izvršena posle pretrage

##### select\_records(field\_name, all\_records)

Koristite *select\_records* metod da dodate zapise objektu izborom iz lookup polja objekta. Na primer, ovaj metod je korišćen u Demo aplikaciji za dodavanje trekova fakturi izborom iz Tracks kataloga. Parametri:

* field\_name parametar je ime lookup objekta
* Ako je all\_records parametar *true*, svi zapisi su dodati, inače metod izostavlja postojeće zapise (one koje su bili selektovani ranije).

*Primer*

function on\_view\_form\_created(item) {

var btn = item.add\_view\_button('Select', {type: 'primary'});

btn.click(function() {

item.select\_records('track');

});

}

##### set\_fields(*field\_list*)

Koristite *set\_fields* metod da definišete i smestite interno *fields* opciju koju će koristiti *open* metod, kada njegova sopstvena opcija nije spec. *open* metod čisti interno smeštenu vrednost parametra.

* field\_list parametar je lista imena polja.

*Primer*

Rezultat izvršenja sledećih kodova je isti:

item.open({fields: ['id', 'invoicedate']});

ili:

item.set\_fields(['id', 'invoicedate']);

item.open();

##### set\_order\_by(*field\_list*)

Koristite *set\_order\_by* metod da definišete ismestite interno *order\_by* opciju koju će koristiti *open* metod, kada njegova sopstvena *order\_by* opcija nije spec. *open* metod čisti interno smeštenu vrednost parametra.

* field\_list parametar je lista imena polja.

Ako postoji znak *’-’* pre imena polja, tada po tom polju zapisi će biti sortirani u opadajućem poretku.

*Primer*

Rezultat izvršenja sledećih kodova je isti:

item.open({order\_by: ['-invoicedate']});

ili:

item.set\_order\_by(['-invoicedate']);

item.open();

##### set\_where(*where*)

Koristite *set\_where* metod da definišete i smestite interno *where* opciju koja će biti korišćena u *open* metodu, kada njegova sopstvena *where* opcije nije spec. *open* metod čisti interno smeštenu vrednost parametra.

* where parametar je objekat ključ-vrednost, gde su ključevi imena polja, koji su praćeni, posle duple podcrte, filtrirajućim simbolima.

*Primer*

Rezultat izvršavanja sledećih kodova biće isti:

item.open({where: {id: 100}});

ili:

item.set\_where({id: 100});

item.open();

##### show\_history()

*show\_history* metod objekta otvara dijalog koji prikazuje istoriju promena izabranog zapisa.

##### update\_controls()

Pozovite *update\_controls* da kontrole svesne podataka osveže sadržaj i prikažu tekuće podatke.

##### view(container)

Koristite *view* metod da kreirate *view* formu objekta. Metod proverava da li javascript moduli objekta i njegovog vlasnika su učitani, ako nisu (*Dynamic JS modules loading* parametar projekta je postavljen) tada ih učitava.

* Ako container parametar (Jquery objekat Dom elementa) je spec. *html* šablon *view* forme je insertovan u kontejner.

Ako *init\_tabs* metod taska je pozvan za ovaj kontejner, tab je kreiran za formu. Posle toga, poziva *create\_view\_form* metod.

*Primer*

U sledećm kodu, view forma za žurnal taska je kreiran u *on\_page\_loaded* event hendlera:

function on\_page\_loaded(task) {

$(#title).html(task.item\_caption);

if (task.safe\_mode) {

$(#user-info).text(task.user\_info.role\_name + ' ' +

task.user\_info.user\_name);

$('#log-out')

.show()

.click(function(e) {

e.preventDefault();

task.logout();

});

}

task.init\_tabs($(#content));

task.tasks.view($(#content));

$(window).on('resize', function() {

resize(task);

});

}

#### Eventi

##### on\_after\_append(item)

Pojavljuje se pošto aplikacija insertuje ili doda zapis.

* item parametar je objekat koji pokreće event.

Napišite *on\_after\_append* event hendler da napravite akciju neposredno posle inserta ili dodavanja zapisa datasetu objekta. *on\_after\_append* je pokrenut od *insert* ili *append* metoda.

##### on\_after\_apply(item)

Pojavljuje se posle upisa promena *change log-a* u bazu podataka.

* item parametar je objekat koji pokreće event.

Napišite *on\_after\_apply* event hendler da napravite akcju neposredno posle upisa promena podataka u bazu podataka. *on\_after\_apply* je pokrenut od *apply* metaoda.

##### on\_after\_cancel(item)

Pojavljuje se posle odustajanja od promena podataka dataseta objekta.

* item parametar je objekat koji pokreće event.

Napišite *on\_after\_cancel* event hendler da napravite akciju neposredno posle odustajanja od promena podataka dataseta objekta. *on\_after\_cancel* je pozvan od *cancel* metoda.

##### on\_after\_delete(item)

Pojavljuje se posle brisanja zapisa.

* item parametar je objekat koji pokreće event.

Napišite *on\_after\_delete* event hendler da napravite akciju posle brisanja zapisa dataseta objekta. *on\_after\_deleteje* pozvan od *delete* metoda. *Kurzor* je repozicionaran na zapis pre obrisanog zapisa.

##### on\_after\_edit(item)

Pojavljuje se posle početka editovanja zapisa.

* item parametar je objekat koji pokreće event.

Napišite *on\_after\_edit* event hendler da napravite akciju neposredno posle početka editovanja zapisa. *on\_after\_edit* je pozvan od *edit* metoda.

##### on\_after\_open(item)

Pojavljuje se posle prijema odgovara sa servera za dobijanje dataseta.

* item parametar je objekat koji pokreće event.

Napišite *on\_after\_open* event hendler da napravite akciju neposredno posle dobijanja dataseta sa servera. *on\_after\_open* je pozvan od *open* metoda.

##### on\_after\_post(item)

Pojavljuje se posle objavljivanje zapisa u dataset objekta.

* item parametar je objekat koji pokreće event.

Napišite *on\_after\_post* event hendler da napravite akciju neposredno posle objavljivanja zapisa u dataset. *on\_after\_post* je pozvan od *post* metoda.

##### on\_after\_scroll(item)

Pojavljuje se posle skrolovanja sa jednog na drugi zapis.

* item parametar je objekat koji pokreće event.

Napišite *on\_after\_scroll* event hendler da napravite akciju neposredno posle skrolovanja sa jednog na drugi zapis, a kao rezultat *first*, *last*, *next*, *prior* i *locate* metoda. *on\_after\_scroll* je pozvan posle svih drugih eventa pokrenutih ovim metodama i bilo kojim drugim metodama koje menjaju poziciju kurzora u datasetu.

*Example*

Sledeći kod korišćen je u Demo projektu da asinhrono otvori invoice\_table detalj posle promene aktivnog zapisa u invoices objektu:

var ScrollTimeOut;

function on\_after\_scroll(item) {

clearTimeout(ScrollTimeOut);

ScrollTimeOut = setTimeout(

function() {

item.invoice\_table.open(function() {});

},

100

);

}

##### on\_before\_append(item)

Ppojavljuje se pre inserta ili dodavanja zapisa.

* item parametar je objekat koji pokreće event.

Napišite *on\_before\_append* event hendler da napravite akciju neposredno pre inserta ili dodavanja zapis u dataset objekta. *on\_before\_append* je pozvan od *insert* ili *append* metoda.

##### on\_before\_apply(item, params)

Pojavljuje se pre nego aplikacija upiše promene na datasetu u bazu podataka.

* item parametar je objekat koji pokreće event.
* params je objekat prosledjen *apply* medodu ili prazan objekat ako je objekat nedefinisan. Ovaj objekat je prosledjen u *on\_apply* event hendleru da izvrši neke akcije kada se upisuju promene u bazu podataka.

Napišite *on\_before\_apply* event hendler da napravite akciju neposredno pre nego što aplikacija napravi upis promena na *change log*-u u bazu podataka. *on\_before\_apply* je pokrenut od *apply* metoda.

##### on\_before\_cancel(item)

Pojavljuje se pre nego se odustane od promena na datasetu.

* item parametar je objekat koji pokreće event.

Napišite *on\_before\_cancel* event hendler da napravite akciju neposredno pre odustajanja od promena na datasetu objekta.

##### on\_before\_delete(item)

Pojavljuje se brisanja zapisa.

* item parametar je objekat koji pokreće event.

Napišite *on\_before\_delete* event hendler da napravite akciju neposredno pre brisanja aktivnog zapisa dataseta objekta. *on\_before\_delete* je pozvan od *delete* metoda.

##### on\_before\_edit(item)

Pojavljuje se pre početka starta editovanja aktivnog zapisa.

* item parametar je objekat koji pokreće event.

Napišite on\_before\_edit event hendler da napravite akciju neposredno pre početka editovanja zapisa aktivnog zapisa dataseta objekta. *on\_before\_edit* je pozvan od *edit* metoda.

##### on\_before\_field\_changed(field)

Napišite *on\_before\_field\_changed* event hendler da implementirate bilo koje posebno procesiranje pre nego su podaci polja budu promenjena.

* field parametar je polje čiji podaci će biti promenjeni.

Za dobijanje vlasnika polja, upotrebite *owner* atribut polja. Pre pokretanja ovog event hendlera aplikacija pridružuje novu vrednost koja će biti postavljena kao vrednost polja na *new\_value* atribut polja. Možete promeniti vrednost tog atributa. Ta vrednost će biti korišćena za promenu podataka polja.

*Example*

function on\_before\_field\_changed(field) {

if (field.field\_name === 'quantity' && field.new\_value < 0) {

field.new\_value = 0;

}

}

##### on\_before\_open(item, params)

Pojavljujje se pre slanja zahteva na server za dobjanje dataseta.

* item parametar je objekat koji pokreće event.
* params parametar je objekat koji je prosledjen open metodu ili empty objekat ako je objekat nedefinisan. Objekat je prosledjen na server i može biti korišćen u on\_open event hendleru da izvrši neke akcije kada je dataset dobijen.

Napišite *on\_before\_open event* hendler da napravite akciju neposredno pre dobijanja dataseta sa servera. *on\_before\_open* je pozvan od *open* metoda.

##### on\_before\_post(item)

Pojavljuje se pre objavljivanja zapisa.

* item parametar je objekat koji pokreće event.

Napišite *on\_before\_post* event hendler da napravite neposredno pre objavljivanja zapisa u dataset objekta. *on\_before\_post* je pozvan od *post* metoda.

##### on\_before\_scroll(item)

Pojavljuje se pri skrolovanju sa jednog zapisa na drugi.

* item parametar je objekat koji pokreće event.

Napišite *on\_before\_scroll* event hendler da napravite akciju neposredno pre skrolovanja kurzora. A kao rezultat pozivanja *first*, *last*, *next*, *prior* i *locate* metoda. *on\_before\_scroll* je pozvan pre svih drugih eventa pokrenutih pomenutim metodama i drugim metodama koje menjaju poziciju kurzora zapisa u datasetu objekta.

##### on\_detail\_changed(item, detail)

Pojavljuje se posle objavljivanja zapisa detalja. Koristi *clearTimeout* i *setTimeout* Javascript funkcije tako da ako su zapisi promenjeni u petlji biće pokrenut je kada se pojavi promena poslednjeg zapisa.

* item parametar je objekat koji pokreće event.
* detail parametar je objekta detalaj koji je promenjen.

Napišite *on\_detail\_changed* event hendler da izračunate koristeći *calc\_summary* metod, sume polja detalja i sačuvate te vrednosti u polja mastera.

*Primeri*

function on\_detail\_changed(item, detail) {

var fields;

if (detail.item\_name === 'invoice\_table') {

fields = [

{total: total},

{tax: tax},

{subtotal: amount}

];

item.calc\_summary(detail, fields);

}

}

##### on\_edit\_form\_close\_query(item)

*on\_edit\_form\_close\_query* event je pokrenut od *close\_edit\_form* metode objekta.

* item parametar je objekat koji pokreće event.

##### on\_edit\_form\_created(item)

*on\_edit\_form\_created* event je pokrenut od *create\_edit\_form* metoda kada je forma kreirana ali još nije pokrenuta.

* item parametar je objekat koji pokreće event.

##### on\_edit\_form\_keydown(item, event)

*on\_edit\_form\_keydown* event je pokrenut kada se *keydown* event pojavio za *edit* form objekta.

* item parametar je objekat koji pokreće event.
* event je JQuery event objekat.

##### on\_edit\_form\_keyup(item, event)

*on\_edit\_form\_keyup* event je pokrenut kada se *keyup* event pojavio za *edit* form objekta.

* item parametar je objekat koji pokreće event.
* event je JQuery event objekat.

##### on\_edit\_form\_shown(item)

*on\_edit\_form\_shown* event je pokrenut od *create\_edit\_form* metoda kada je forma prikazana.

* item parametar je objekat koji pokreće event.

##### on\_field\_changed(field, lookup\_item)

Napišite *on\_field\_changed* event hendler da odgovorite na promenu podataka polja.

* field parametar je polje čiji su podaci promenjeni
* lookup\_item parametar je definisan kada je polje lookup polje

Za dobijanje vlasnika polja, koristite *owner* atribut polja.

*Example*

function on\_field\_changed(field, lookup\_item) {

var item = field.owner;

if (field.field\_name === 'quantity' || field.field\_name === 'unitprice') {

item.owner.calc\_total(item);

}

else if (field.field\_name === 'track' && lookup\_item) {

item.quantity.value = 1;

item.unitprice.value = lookup\_item.unitprice.value;

}

}

##### on\_field\_get\_html(field)

Napišite *on\_field\_get\_html* event hendler da spec. html koji će biti insertovan u ćeliju tabele za polje. Ako event hendler ne vrati vrednost, aplikacija proverava da li *on\_field\_get\_text* event hendler je definisan i da li vraća vrednost, inače *display\_text* svojstvo će biti korišćeno za prokazivanje vrednosti polja u ćeliji tabele.

* field parametar je polje čiji display\_text je procesiran

*Primeri*

function on\_field\_get\_html(field) {

if (field.field\_name === 'total') {

if (field.value > 10) {

return '<strong>' + field.display\_text + '</strong>';

}

}

}

##### on\_field\_select\_value(field, lookup\_item)

Kada korisnik klikne na dugme desno od ulaza polja il koristi typeahead, aplikacija kreira kopiju lookup objekta poja i pokreće *on\_field\_select\_value* event. Koristite *on\_field\_select\_value* da spec. polja koja će biti prikazana, postavite filtere za lookup objekat, pre nego što će biti otvoren.

* field parametar je polje čiji podaci će biti izabrani
* lookup\_item parametar je kopija lookup objekta polja.

*Primeri*

function on\_field\_select\_value(field, lookup\_item) {

if (field.field\_name === 'customer') {

lookup\_item.set\_where({lastname\_\_startwith: 'B'});

lookup\_item.view\_options.fields =

['firstname', 'lastname', 'address', 'phone'];

}

}

##### on\_field\_validate(field)

Napišite *on\_field\_validate* event handler da validirate promene na pravljene na podacima polja.

* field parametar je polje čiji se podaci validiraju.

Event handler mora da vrati string ako vrednost polja je pogrešnoa. Kada event hendler vrati string aplikacija baca izuzetak. Even je pokrenut kada je *post* metod pozvan, ili kada korisnik napusta ulaz korišćen za editovanje vrednosti polja.

*Example*

function on\_field\_validate(field) {

if (field.field\_name === 'sum' && field.value > 10000000) {

return 'The sum is too big.';

}

}

##### on\_filter\_changed(filter, lookup\_item)

Napiši *on\_filter\_changed* event hendler da odgovorite na promene filterskih podataka.

* filter parametar je filte čiji su podaci promenjeni.

Da dobijete vlasnika filtera, koristite owner atribut filtera.

##### on\_filter\_form\_close\_query(item)

*on\_filter\_form\_close\_query* event je pokrenut od *close\_filter\_form* metoda objekta.

* item parametar je objekat koji pokreće event.

##### on\_filter\_form\_created(item)

*on\_filter\_form\_created* event je pokrenut od *create\_filter\_form* metoda kada je forma kreirana ali još nije prikazana.

* item parametar je objekat koji pokreće event.

##### on\_filter\_form\_shown(item)

*on\_filter\_form\_shown* event je pokrenut od *create\_filter\_form* metoda kada je forma prikazana.

* item parametar je objekat koji pokreće event.

##### on\_filter\_record(item)

Koristite *on\_filter\_record* event da filtrirate dataset lokalno. Event je pokrenut kada kursor skroluje na drugi zapis i *Filtered* svojstvo je *true*. Napišite *on\_filter\_record* event handler da spec. za svaki zapis u datasetu da li bi trebao biti vidljiv u aplikaciji. Da indicirate da zapis prolazi filterski uslov, *on\_filter\_record* event handler mora vratiti *true*.

* item parametar je objekat koji pokreće event.

*Example*

function on\_filter\_record(item) {

if (item.type.value === 2) {

return true;

}

}

function enable\_filtering(item) {

item.filtered = true;

}

function disable\_filtering(item) {

item.filtered = false;

}

##### on\_filters\_applied(item)

Napišie *on\_filters\_applied* event hendler da napravtite specijalno procesiranje kada su filteri primenjeni na dataset.

##### on\_field\_get\_text(field)

Napišite *on\_field\_get\_text* event hendler da izvršite prilagodjeno procesiranje za *display\_text* svojstvo. Ako event hendler ne vrati vrednost, aplikacija će koristiti *display\_text* svojstvo za prikaz vrednosti polja, inače će vraćena vrednost biti korišćena.

* field parametar je polje čiji *display\_text* je procesiran

*Primer*

function on\_field\_get\_text(field) {

if (field.field\_name === 'customer') {

return field.owner.firstname.lookup\_text + ' ' + field.lookup\_text;

}

}

##### on\_view\_form\_close\_query(item)

*on\_view\_form\_close\_query* event je pokrenut od *close\_view\_form* metoda objekta.

* item parametar je objekat koji pokreće event.

##### on\_view\_form\_created(item)

*on\_view\_form\_created* event je pokrenut od *view* metoda kada je forma kreirana ali još nije prikazana.

* item parametar je objekat koji pokreće event.

##### on\_view\_form\_keydown(item, event)

*on\_view\_form\_keydown* event je pokrenut kada se *keydown* event pojavi za *view* formu objekta.

* item parametar je objekat koji pokreće event.
* event je JQuery event objekat.

##### on\_view\_form\_keyup(item, event)

*on\_view\_form\_keyup* event je pokrenut kada se *keyup* event pojavi za *view* formu objekta.

* item parametar je objekat koji pokreće event.
* event je JQuery event objekat.

##### on\_view\_form\_shown(item)

*on\_view\_form\_shown* event je pokrenut od *view* metod objekta kada je forma prikazana.

* item parametar je objekat koji pokreće event.

### *class* Detail()

Klasa *Detail* nasledjuje atribute, metode i evente Item klasa.

#### Atrubuti

##### master

Koristite master atribut da dobijaete referencu na *master* objekat detalja.

#### *class* Reports()

Klasa *Reports* koristi se za kreiranje grupe objekata taska stabla koji su vlasnici izveštaja u projektu. Ispod su atributi i metode klase. Klasa nasledjuje atribute i metode klase *AbstractItem.*

### *class* Report()

Ispod su atributi, metode i eventi klase. Klasa nasledjuje atribute i metode klase *AbstractItem*.

#### Atributi

##### extension

Koristite *extension* atribut da spec. tip izveštaja. Server, zasnovano na šablonu izveštaja, generiše ods fajl. Ako je ext izveštaja drugačija server radi konverzju koristeći LibreOffice. Vrednost atributa može biti bilo koja ext za koju LibreOffice ima podršku za konverziju.

*Example*

function on\_before\_print\_report(report) {

report.extension = 'html';

}

##### param\_form

Koristite *param\_form* atribut da dobijete Jquery objekat koji predstavlja *param* formu izvešaja. Ona je kreirana od *create\_param\_form* metoda, koja je obično pozvana od *print* metoda. *close\_param\_form* metod postavlja *param\_form* vrednost na *undefined*.

*Primeri*

function on\_param\_form\_created(report) {

report.create\_param\_inputs(report.param\_form.find(.edit-body));

report.param\_form.find(#cancel-btn).on('click.task', function() {

report.close\_param\_form();

});

report.param\_form.find(#ok-btn).on('click.task', function() {

report.process\_report()

});

}

##### param\_options

Koristite *param\_options* atribut da spec. parametre modalne modal *param* forme. *param\_options* je objekat koji ima sledeće atribute:

* width – širina modalne forme, podarazumevano je *560 px*,
* title – naslov modalne forme, podrazumevano je vrednost atribute *report\_caption*,
* close\_button – ako je *tačno*, *close* dugme će biti kreiran u gornjem desnom uglu forme, podrazumevana vrednost je *true*,
* close\_caption – ako je *true* i *close\_button* je *true*, biće prikazano *Close - [Esc]* blizu dugmeta,
* close\_on\_escape – ako je true, pritiskom na *Escape* će pokrenuti *close\_param\_form* metod,
* close\_focusout – ako je *true*, *close\_param\_form* metod biće pozvan kada forma izgubi fokus
* template\_class – ako je spec., div sa ovom klasom biće traženo u task *templates* atributu i korišćeno kao html šablon forme pri njenom kreiranju.

*Primer*

function on\_param\_form\_created(report) {

report.param\_options.width = 800;

report.param\_options.close\_button = false;

report.param\_options.close\_on\_escape = false;

}

#### Metodi

##### close\_param\_form()

Koristite *close\_param\_form* metod izveštaja da zatvorite *param* formu objekta. *close\_param\_form* metod pokreće *on\_param\_form\_close\_query* event hendler izveštaja, ako je definisan. Ako je event hendler definisan i

* vraća *true* – forma je uništena, *param\_form* atribut izveštaja je postavljen na *undefined* i metod vraća.
* vraća *false* – operacija je prekinuta i metod vraća,

Ako ne vraća vrednost (*undefined*) metod pokreće *on\_param\_form\_close\_query* grupe koja je vlasnik izveštaja, ako je definisan za grupu. Ako je event hendler definisan i

* vraća *true* – forma je uništena, *param\_form* arribut izveštaja je postavljen na *undefined* i metod vraća.
* vraća *false* – operacija je prekinuta i metod vraća,

Ako ne vraća vrednost (*undefined*) metod pokreće *on\_view\_form\_close\_query* taska, ako je definisan. Ako je event hendler definisan i

* vraća *true* – forma je uništena, *param\_form* atribut izveštaja je postavljen na *undefined* i metod vraća.
* vraća *false* – operacija je prekinuta i metod vraća,

Ako nema definisanog event hendlera ili nijedan od tih event hendlera ne vraća *false*, forma je uništena i *param\_form* atribut izveštaja je postavljen na *undefined.*

##### create\_param\_form()

*create\_param\_form* metod je pozvana od *print* metoda da kreira formu da postavi parametre izveštaja pre slanja zahteva na server *process\_report* metodom. Metod proverava da li javascript moduli izveštaja i njegovog vlasnika su učitani, ako nisu (Dynamic JS modules loading parametar je postavljen) učitava ih. Potom pretražuje za html šablon u task *templates* atributu i kreira klon šablona i pridružuje ga na *param\_form* atribut izveštaja.

Kreira formu i dodaje joj html. Pokreće *on\_param\_form\_created* taska. Pokreće *on\_param\_form\_created* grupe izveštaja, ako je definisan. Pokreće *on\_param\_form\_created* izveštaja, ako je definisan.

Prikazuje formu. Pre prikazivanja forme metod primenjuje *options* spec u *param\_options* atributu. Pokreće *on\_param\_form\_shown* taska. Pokreće *on\_param\_form\_created* grupe izveštaja, ako je definisan. Pokreće *on\_param\_form\_shown* izveštaja, ako je definisan.

##### create\_param\_inputs(*container*, *options*)

Koristite create\_param\_inputs za kreiranje vizuelnih kontrola koje su svesne podataka (unosi, polja za potvrdu) za uređivanje parametara izveštaja. Ovaj metod se obično koristi u *on\_param\_form\_created* eventima pokrenutim metodom *create\_param\_form*, koja se obično poziva metodom *print*. Sledeći parametri se prosleđuju metodu:

* container - JQuery objekat koji će sadržati vizuelne kontrole, ako je dužina kontejnera 0 (bez kontejnera), metod vraća.
* Options - opcije koje određuju kako se kontrole prikazuju. Ovaj objekat ima sledeće atribute:
  + params - lista imena parametara. Ako je spec., vizuelna kontrola će biti kreirana za svaki param čije se ime nalazi na ovoj listi, ako nije spec. (podrazumevano), kontrola će biti kreirana za sve vidljive parametre spec. u AppBulderu,
  + col\_count - broj kolona koje će biti kreirane za vizuelne kontrole, podrazumevano je 1
  + label\_on\_top - podrazumevano je *false*. Podrazumevano oznake se postavljaju levo od kontrola, u suprotnom se kreiraju iznad kontrola.
  + tabindex - ako je *tabindex* spec., to će biti tabindex prve vizuelne kontrole, tabindex svih narednih kontrola će biti povećan za 1.
  + autocomplete – podrazumevano je *false*.

Pre kreiranja kontrola aplikacija prazni kontejner.

*Example*

function on\_param\_form\_created(item) {

item.create\_param\_inputs(item.param\_form.find(.edit-body));

item.param\_form.find(#cancel-btn).on('click.task', function() {

item.close\_param\_form()

});

item.param\_form.find(#ok-btn).on('click.task', function() {

item.process\_report()

});

}

##### print(create\_form)

Koristite print za štampanje izveštaja.

* Ako je *create*\_*form* parametar izostavljen ili je jednak *true*, metoda poziva metodu *create\_param\_form* da kreira formu zasnovanu na html šablonu definisanom u fajlu *index*.*html*.
* Ako *create*\_*formje* parametar postavljen na *false*, a izveštaj nema vidljive parametre, on poziva *process\_report* da pošalje zahtev serveru za generisanje izveštaja

##### process\_report()

*process\_report* metod šalje izveštaj serveru da generiše njegov sadržaj i prihvata fajl izveštaja koju server vraća klijentu i otvara ili čuva. Poziva se metodom *print* direktno, ako je *create*\_*form* parametar *false* i nema vidljivih parametara izveštaja. Ako postoje vidljivi parametri izveštaja, metoda *print* kreira formu za spec. vrednosti parametara i formu bi trebalo pozvati (na primer, nekim *onclick* eventom dugmeta). Proverava da li su vrednosti parametara važeće i pokreće sledeće događaje:

* on\_before\_print\_report event handler grupe izveštaja
* on\_before\_print\_report event handler izveštaja

U ovim event hendler programer može definisati neke uobičajene (event hendler grupe izveštaja) ili specifične (event hendler izveštaja) atribute izveštaja. Posle toga metod process\_report šalje asinhroni zahtev serveru da generiše sadržaj izveštaja. Server vraća metodu *url* do fajla sa generisanim sadržajem izveštaja. Metod zatim proverava da li je event hendler *on\_open\_report* grupe izveštaja definisan. Ako je definisan poziva ga, inače proverava on\_open\_report izveštaja. Ako je definisan poziva ga. Ako nijedan od ovih event hendlera nije definisan, on (u zavisnosti od atributa *extension* izveštaja ) otvara izveštaj u baruzeru ili ga čuva na disku.

*Primer*

Sledeći event hendler, definisan u klijent modulu invoice izveštaka Demo aplikacije postavlja vrednost *report id* parametra:

function on\_before\_print\_report(report) {

report.id.value = report.task.invoices.id.value;

}

#### Eventi

##### on\_before\_print\_report(report)

*on*\_*before*\_*print*\_*report* event pokrenut je od *process\_report* metoda. Koristite *on*\_*before*\_*print*\_*report* za preduzimanje određenih radnji pre slanja zahteva serveru za generisanje izveštaja. Parametar *report* je izveštaj koji je pokrenuo događaj.

##### on\_open\_report(report)

*on*\_*open*\_*report* event pokrenut je od *process\_report* metoda. Parametar *report* je izveštaj koji je pokrenuo događaj.

##### on\_param\_form\_close\_query(report)

*on*\_*param*\_*form*\_*close*\_*query* event je pokrenut od *close\_param\_form* metoda. Parametar *report* je izveštaj koji je pokrenuo događaj.

##### on\_param\_form\_created

*on*\_*param*\_*form*\_*created* event je pokrenut od *create*\_*param*\_form metoda, koji je obično pozvan od *print* metoda. *report* parametar je izveštaj koji je pokrenuo event.

##### on\_param\_form\_shown

*on*\_*param*\_*form*\_*shown* event je pokrenut od *create*\_*param*\_form metoda, koji je obično pozvan od *print* metoda. *report* parametar je izveštaj koji je pokrenuo event.

### *class* Field()

#### Atributi i svojstva

##### display\_text

Predstavlja vrednost polja kao *read-only* string. Svojstvo*display****\_****text* je prikaz vrednosti polja koji se prikazuje u kontroli koja je svesna podataka. Ako je definisan event hendler *on\_get\_field\_text*, *display\_text* je vrednost koju vraća ovaj rukovalac događaja. Inače, *display\_text* je vrednost svojstva *lookup\_text* za lookup polja i svojstvo text, konvertovana u skladu sa postavkama locale za jezik, za druga polja. *display\_text* je string predstava svojstva vrednosti polja kada se ne edituje. Kada se polje edituje, koristi se svojstvo text.

*Primer*

function on\_get\_field\_text(field) {

if (field.field\_name === 'customer') {

return field.owner.firstname.lookup\_text + ' ' + field.lookup\_text;

}

}

##### field\_caption

*field\_caption* atribut spec. ime polja koje se pojavljuje korisnicima.

##### field\_mask

Maska omogućava korisniku da lakše unese unos fiksne širine gde želite da unese podatke u određenom formatu (datumi, brojevi telefona itd.). Maska je definisana formatom sastavljenim od *literala* maske i *definicija* maske. Svaki znak koji nije na listi definicija ispod smatra se literalom maske. Literali maske će biti automatski uneti za korisnika dok kucaju i korisnik ih neće moći ukloniti. Sledeće definicije maske su unapred definisane:

* a – Predstavlja alfa znak (AZ,az)
* 9 – Predstavlja numerički znak (0-9)
* *\** - Predstavlja alfanumerički znak (AZ,az,0-9)

*Primer*

function on\_edit\_form\_created(item) {

item.phone.field\_mask = '999-99-99';

}

##### field\_name

Spec. ime polja koje se koristi u kodu. Koristite *field\_name* da referencirate polje u kodu.

##### field\_size

Identifikuje veličinu objekta tekst polja.

##### field\_type

Identifikuje *tip* objekta polja. Koristite *field\_type* atribut da saznate tip podatka koje polje sadrži. To je jedna od sledećih vrednosti:

* text,
* integer,
* float,
* currency,
* date,
* datetime,
* boolean,
* blob

##### lookup\_text

Koristite *lookup\_text* svojstvo da dobijete lookup vrednost lookup polja konvertovanu u string. Ako je polje lookup polje daje *lookup\_text*, inače daje vrednost *text* svojstva.

##### lookup\_type

Za lookup polja idetifikuje tip lookup vrednosti, inače vraća *field\_type* atributa.

##### lookup\_value

Koristite *lookup\_value* svojstvo da dobijete lookup vrednost lookup polja. Ako je polje lookup polje daje *lookup\_value*, inače daje vrednost *value* svojstva.

##### owner

Identifikuje objekat vlasnik polja.

*Example*

function calculate(item) {

}

function on\_field\_changed(field, lookup\_item) {

if (field.field\_name === 'taxrate') {

calculate(field.owner);

}

}

##### raw\_value

Koristite *raw\_value* read-only svojstvo da čitate podatke direktno iz dataseta objekta. Druga svojstva kao što su *value* i *text* koriste konverziju. *value* svojstvo konvertuje *null* u *0* za numerička polja.

##### read\_only

Odredjuje da li polje može biti promenjeno u kontolama svesnim podataka. Postavlja *read\_only* na *true* da spreči da polje bude promenjeno u kontrolama svesnim podataka.

##### required

Određuje da li je potrebna vrednost koja nije prazna za polje. Koristite *required* da biste saznali da li polje zahteva vrednost ili polje može biti prazno. Kada je traženo svojstvo postavljeno na *true*, pokušaj da se postavi null vrednost će dovesti do podizanja izuzetka.

##### text

Koristite *text* svojstvo da dobijete ili postavite *text* vrednost polja. Ovo svojstvo uzima vrednost *value* svojstva i konvertuje ga u *text*. Obrnuto, konvertuje *text* u tip polja i pridružuje vrednost *value* svojstvu polja na ovu vrednost.

##### value

Koristite svojstvo *value* da biste dobili ili postavili vrednost polja.

###### Dobijanje vrednosti

Kada su podaci polja ***null***, polje ih konvertuje u ***0***, ako je *field\_type* *integer*, *float* ili *currency*, ili u prazan string ako je *field\_type* *text*. Za lookup polja vrednost ovog svojstva je ceo broj koji je vrednost polja *id* odgovarajućeg zapisa u lookup objektu. Da biste dobili traženu vrednost polja, koristite svojstvo *lookup\_value*.

###### Postavljanje vrednosti

Kada se dodeli nova vrednost, polje proverava da li trenutna vrednost nije jednaka novoj. Ako je tako

* postavlja svoj atribut ***new\_value*** na ovu vrednost,
* pokreće događaj *on\_before\_field\_change* ako je definisan za polje,
* menja podatke polja u vrednost atributa ***new\_value*** i njega postavlja na ***null***,
* označi objekat kao promenjen, tako da će metoda is\_modified vratiti ***tačno,***
* pokreće događaj *on\_fied\_changed* ako je on definisan za polje,
* ažurira kontrole svesne podataka.

*Example*

function calc\_total(item) {

item.amount.value = item.round(item.quantity.value \* item.unitprice.value, 2);

item.tax.value = item.round(item.amount.value \* item.owner.taxrate.value / 100, 2);

item.total.value = item.amount.value + item.tax.value;

}

#### Metodi

##### download()

Pozovite *download* za polja tipa *file* za dopemanje fajla.

*Example*

function on\_view\_form\_created(item) {

item.add\_view\_button('Download').click(function() {

item.attachment.download();

});

}

##### open()

Pozovite *open* za polja tipa *file* za otvaranje *url* fajla koristeći *window.open*.

*Example*

function on\_view\_form\_created(item) {

item.add\_view\_button('Open').click(function() {

item.attachment.open();

});

}

### *class* Filter()

#### Attributi i svojstva

##### filter\_caption

*filter\_caption* atribut spec. ime filtera koji se pojavljuje korisnicima.

##### filter\_name

Spec. ime filtera koji se koristi u kodu.

##### owner

Identifikuje objekat koji je vlasnik filtera.

##### value

Koristite *value* svojstvo da dobijete ili postavite vrednost filtera.

*Example*

function on\_view\_form\_created(item) {

item.filters.invoicedate1.value = new Date(new Date().setYear(new Date().getFullYear() - 1));

}

##### visible

Ako je vrednost ovog svojstva *true* ulazana kontrola za taj filter će biti kreirana od *create\_filter\_inputs* metoda, ako filter\_optiona nije spec.

## Serverska strana - python reference

Svi objekti okvira predstavljaju task stablo. Ispod su klase objekata task stabla:

### class App

*App* klasa se koristi za kreiranje *WSGI* aplikacije. Ispod su atributi klase.

#### admin

Vraća referencu na AppBuilder task stablo.

#### task

Vraća refeencu na Project task stablo.

### *class* AbstractItem

Klasa *AbstractItem* je predak za sve objekte koji sadrže podatke u task stablu. Ispod su atributi i metode klase.

#### Attributi

##### environ

Spec. *WSGI* *environment* rečnik tekućeg zahteva sa klijenta.

##### ID

ID atribut je jedinstven u okviru identifikator objekta. ID atribut je najkorisniji kada referencira na objekat. Koristi se interno.

##### item\_caption

Spec. ime objekta koje se pojavljuje korisnicima.

##### item\_name

Spec. ime objekta koje se koristi u kodu.

##### item\_type

Spec. tip objekta. Može imati jednu od sledečih vrednosti:

* task,
* items,
* details,
* reports,
* item,
* detail\_item,
* report,
* detail

##### items

Lista svih objekata u vlasništvu nekog objekta. Koristite *items* da pristupite bilo kom od objekta u vlasništvu tog objekta.

##### owner

Indicira vlasnika objekta.

##### session

Koristite *session* svojstvo da pristupite *session* objektu tekućeg zahteva sa klijenta. *session* je rečnik koji ima sledeće objekte:

* ip - ip adresa korisnika
* user\_info – rečnik koji sadrži informacije o korisniku
  + user\_id - id korisnika
  + user\_name – ime korisnika
  + role\_id - id role korisnika
  + role\_name – ime role korisnika

*Primer*

def on\_open(item, params):

user\_id = item.session['user\_info']['user\_id']

if user\_id:

params['\_\_filters'].append(['user\_id', item.task.consts.FILTER\_EQ, user\_id])

def on\_apply(item, delta, params):

user\_id = item.session['user\_info']['user\_id']

if user\_id:

for d in delta:

d.edit()

d.user\_id.value = user\_id

d.post()

##### task

Indicira koren task stabla koji je vlasnik objekta.

#### Metodi

##### can\_view(*self*)

Koristite *can\_view* metod da odredite da li korisnik u tekućoj sesiji može videti zapise objekta ili štampati izveštaj.

##### item\_by\_ID(*self*, *ID*)

*item\_by\_ID* pretražuje preko svih objekata projektno task stabla, počev od tekućeg objekta, za objekat čiji *id* je jednak *ID* parametru.

### class Task

Task klasa se koristi za kreiranje korena task stabla projekta. Ispod su atributi, metodi i eventi. Ova klasa nasledjuje atribute i metode klase *AbstractItem*.

#### Atributi

##### app

Vraća referencu na *WSGI* *app* objekat. Okvir koristi *Werkzeug* WSGI pomoćnu biblioteku..

##### work\_dir

Vraća realni absolutni put za projektni folder.

#### Metodi

##### check\_password\_hash(*self*, *pwhash*, *password*)

Koristite *check\_password\_hash* da proverite lozinku u odnosu na datu posoljenu hešovanu vrednost lozinke. Metod je omotač oko *Werkzeug* *check\_password\_hash* funkcije.

*Example*

def on\_login(task, login, password, ip, session\_uuid):

users = task.users.copy(handlers=False)

users.set\_where(login=login)

users.open()

for u in users:

if task.check\_password\_hash(u.password\_hash.value, password):

return {

'user\_id': users.id.value,

'user\_name': users.name.value,

'role\_id': users.role.value,

'role\_name': users.role.display\_text

}

##### connect(*self*)

Koristite *connect* da nabavite konekciju iz *SQLAlchemy* konekcionog pool-a. Vraćena vrednost je *DBAPI* konekcija. Progamer mora vratiti konekciju u konekcioni pool kada mu više nije potrebna, pozivanjem metode *close* objekta konekcije.

*Example*

def delete\_rec(item, item\_id):

conection = item.task.connect()

try:

cursor = conection.cursor()

cursor.execute('delete from %s where id=%s' % (item.table\_name, item\_id))

conection.commit()

finally:

conection.close()

##### copy\_database(self, dbtype, database=None, user=None, password=None, host=None, port=None, encoding=None, server=None)

Koristite *copy*\_*database* da kopirate podatke baze podataka kada migrirate na drugu bazu podataka.

*Primer*

U sledećm kodu kada je projektno task stablo kreirano aplikacija kopira podatke iz *demo.sqlite* baze podataka u projektnu bazu podataka:

from jam.db.db\_modules import SQLITE

def on\_created(task):

task.copy\_database(SQLITE, '/home/work/demo/demo.sqlite')

##### create\_connection(*self*)

Koristite *create\_connection* da kreirate konekciju na projektnu bazu podataka. Metod vraća novu konekciju. Programer mora zatvoriti konekciju kada mu više nije potrebna.

##### create\_connection\_ex(self, db\_module, database, user=None, password=None, host=None, port=None, encoding=None, server=None)

Koristite *create*\_*connection*\_*ex* da kreirate konekciju na drugu bazu podataka. Metod vraća novu konekciju. Programer mora zatvoriti konekciju kada mu više nije potrebna.

##### execute(*self*, *sql*)

Koristite *execute* da izvršite sql upit (izuzev *select* upita) korišćenjem multiprocesnog konekcionog pool-a. Za select upite koristite *select* metod. *sql* parametar može biti string upita, lista stringova upita, lista listi itd. Svi upiti su izvršeni u jednoj transakciji i ako je izvršenje uspešno *commit* sql komanda se izvršava, inače *rollback* sql komanda se izvršava.

*Primeri*

sql = []

for i in ids:

sql.append('UPDATE DEMO\_CUSTOMERS SET QUANTITY=2 WHERE ID=%s' % i)

item.task.execute(sql)

##### generate\_password\_hash(self, password, metod='pbkdf2:sha256', salt\_length=8)

Ovaj metod hešuje lozinku sa datim metodom i soli sa stringom date dužine. Format vraćenog stringa uključuje korišćeni metod tako da *check\_password\_hash* može proveriti hešovanje. Metod je omotač oko Werkzeug *generate\_password\_hash* funkcije.

*Primer*

def on\_apply(item, delta, params, connection):

for d in delta:

if d.password.value:

d.edit();

d.password\_hash.value = delta.task.generate\_password\_hash(d.password.value)

d.password.value = None

d.post();

##### lock(self, lock\_name, timeout = - 1)

Koristite *lock* da implementirate platformu nezavisne fajl brave u Pythonu, koja obezbedjuje jednostavan način inter-procesne komunikacije. Ovaj metod je omotač oko Python filelock biblioteke *py-filelock*. Jednom kada je brava dobijena, sledeći pokušaji za dobijanje brave biće blokirani, sve dok brava nije nije oslobodjena.

* lock\_name - ime brave. Mora biti jedinstveno u aplikaciji. *filelock* biblioteka kreira fajl u *locks* folderu sa imenom *lock\_name* i *.lock* extenzijom koji se koristi za implementaciju brave.
* timeout – ako brava ne može biti dobijena unutar *timeout* sekundi, *Timeout* izuzetak je podignut.

*Primer*

def calculate(item):

lock = item.task.lock('calculation'):

lock.acquire()

try:

#some code

finally:

lock.release()

je ekvivalentno:

def calculate(item):

with item.task.lock('calculation'):

#some code

Primer sa *timeout*:

from jam.third\_party.filelock import Timeout

def calculate(item):

try

with item.task.lock('calculation', timeout=10):

#some code

except Timeout:

print(Another instance of this application currently holds the lock.)

U sledećem primeru kada upisujemo invoice app izračunava sold tracks. Pre toga zahtevamo bravu:

def on\_apply(item, delta, params):

with item.task.lock('invoice\_saved'):

tracks\_sql = []

delta.update\_deleted()

for d in delta:

for t in d.invoice\_table:

if t.rec\_inserted():

sql = UPDATE DEMO\_TRACKS SET TRACKS\_SOLD = COALESCE(TRACKS\_SOLD, 0) + \

%s WHERE ID = %s (t.quantity.value, t.track.value)

elif t.rec\_deleted():

sql = UPDATE DEMO\_TRACKS SET TRACKS\_SOLD = COALESCE(TRACKS\_SOLD, 0) - \

(SELECT QUANTITY FROM DEMO\_INVOICE\_TABLE WHERE ID=%s) WHERE ID = %s \

(t.id.value, t.track.value)

elif t.rec\_modified():

sql = UPDATE DEMO\_TRACKS SET TRACKS\_SOLD = COALESCE(TRACKS\_SOLD, 0) - \

(SELECT QUANTITY FROM DEMO\_INVOICE\_TABLE WHERE ID=%s) + %s WHERE ID = %s \

(t.id.value, t.quantity.value, t.track.value)

tracks\_sql.append(sql)

sql = delta.apply\_sql()

return item.task.execute(tracks\_sql + [sql])

##### select(*self*, *sql*)

Koristite *select* da izvršite *select* sql upit. Da izvršite upit koristite connection pool. *sql* parametar je upit za izvršenje. Metod vraća listu zapisa.

*Example*

recs = item.task.execute\_select(SELECT \* FROM DEMO\_CUSTOMERS WHERE ID=41)

for r in recs:

print(r)

#### Eventi

##### on\_created(task)

Koristite *on\_created* za inicijalizaciju aplikacije na serveru. Event je pokrenut kada je projektno task stablo kreirano.

* task parameter je referenca na task stablo.

Napomena:

Vreme izvršenja koda u ovom hendleru mora biti veoma kratko jer odredjuje efekte na korisnički doživljaj.

*Example*

def on\_created(task):

# some code

##### on\_ext\_request(task, request, params)

Koristite *on\_ext\_request* da pošaljete zahtev na server na procesiranje:

* task - referenca na task stablo
* request - string koji mora startovati sa */ext*
* params – lista parametara.

*Primer*

Sledeći kod će slati svakih 60 sekundi zahtev na server Demo aplikacije

#!/usr/bin/env python

try:

# For Python 3.0 and later

from urllib.request import urlopen

except ImportError:

# Fall back to Python 2's urllib2

from urllib2 import urlopen

import json

import time

def send(url, request, params):

a = urlopen(url + '/' + request, data=str.encode(json.dumps(params)))

r = json.loads(a.read().decode())

return r['result']['data']

if \_\_name\_\_ == '\_\_main\_\_':

url = 'http://127.0.0.1:8080/ext'

while True:

result = send(url, 'get\_sum', [1, 2, 3])

print(result)

time.sleep(60)

Server će procesirati zahtev i vratiti sumu parametara. *on\_ext\_request* mora biti deklarisan u task server modulu:

def on\_ext\_request(task, request, params):

#print request, params

reqs = request.split('/')

if reqs[2] == 'get\_sum':

return params[0] + params[1] + params[2]

##### on\_login(task, form\_data, info)

Koristite *on\_login* da prepišete podrazumevanu login proceduru koristeći AppBuilder Users tabelu.

* task - reference na task stablu.
* form\_data - rečnik koji sadrži vrednosti koje je korisnik uneo u login formu. Ključevi rečnika su imena atributa ulaza.
* info parametar je rečnik sa sledećim atributima:
  + ip - ip adresa zahteva
  + session\_uuid je uuid sesije koja će biti kreirana.

Event hendler mora vratiti rečnik sa sledećim atributima:

* user\_id – jedinstven id korisnika
* user\_name – korisničko ime
* role\_id – id role
* role\_name – ime role

Login forma je locirana u *index*.htiml fajlu. Možete dodati svoju prilagodjenu formu i dobiti vrednosti ulaza korišćenjem using form\_data parametera.

<form id=login-form target=dummy class=form-horizontal data-caption=Log in>

<div class=control-group>

<label class=control-label for=input-login>Login</label>

<div class=controls>

<input type=text id=input-login name=login tabindex=1 placeholder=login>

</div>

</div>

<div class=control-group>

<label class=control-label for=input-password>Password</label>

<div class=controls>

<input type=password id=input-password name=password tabindex=2

placeholder=password autocomplete=on>

</div>

</div>

<div class=form-footer>

<input type=submit class=btn expanded-btn pull-right id=login-btn value=OK tabindex=3>

</div>

</form>

U ovom primeru korisničke informacije su smeštene u tabelu *Users* objekta projektne baze podataka:

def on\_login(task, form\_data, info):

users = task.users.copy(handlers=False)

users.set\_where(login=form\_data['login'])

users.open()

if users.rec\_count == 1:

if task.check\_password\_hash(users.password\_hash.value, form\_data['password']):

return {

'user\_id': users.id.value,

'user\_name': users.name.value,

'role\_id': users.role.value,

'role\_name': users.role.display\_text

}

### *class* Group

Klasa *Group* se koristi za kreiranje grupe objekata ask stabla. Nasledjuje atribute i metode klase *AbstractItem*.

### class Item

Klasa *Item* se koristi za kreiranje objekata task stabla koji mogu imati pridružene tabele baze podataka. Ispod su atributi, metodi i eventi. Klasa nasledjuje *AbstractItem* klasu.

#### Atributi i svojstva

##### active

Spec. da li je dataset objekta otvoren. *open* metod menja vrednost ovog atributa na *true*. *close* metod postavlja atribut na *false*. Kada je dataset otvoren njegovi zapisi mogu biti navigovani i podaci mogu biti promenjeni i promene sačuvane u tabeli baze podataka.

##### details

Lista svih objekata tetalja objekta.

##### fields

Lista svih objekata polja dataseta objekta.

*Primer*

def customer\_fields(customers):

customers.open(limit=1)

for f in customers.fields:

print f.field\_caption, f.display\_text

##### filters

Lista svih objekata filtera dataseta objekta.

*Primer*

def invoices\_filters(invoices):

for f in invoices.filters:

print f.filter\_name, f.value

##### item\_state

Pregledajte *item*\_*state* da biste utvrdili trenutni režim rada objekta. *Item*\_*state* određuje šta se može uraditi sa podacima u datasetu objekta, kao što je editovanje ili insertovanje novih. *item*\_*state* se stalno menja dok aplikacija obrađuje podatke. Otvaranje dataset objekta menja stanje iz *inactive* u *browse*. Aplikacija može pozvati *edit* da postavi objekat u *edit* stanje ili pozvati *insert* ili *append* da postavi objekat u *insert* stanje.

*Post* ili *cancel* promena, *inserta* ili *brisanja*, menja item\_state iz trenutnog stanja u *browse*. *Close* datasetas menja njegovo stanje u *inactive*.

Da biste proverili vrednost *item\_state*, koristite sledeće metode:

* *is\_new* - označava da li je objekta u insert stanju
* *is\_edited* - označava da li je objekat u edit stanju
* *is\_changing* - označava da li je objekat u stanju editovanja ili insertovanja.

item\_state vrednost može biti:

* *0 - inactive stanje*,
* *1 - browse stanje*,
* *2 - insert stanje*,
* *3 - edit stanje*,
* *4 - delete stanje*.

Task atribut objekat ima konstante koje definišu sledeće atribute:

* STATE\_INACTIVE: 0,
* STATE\_BROWSE: 1,
* STATE\_INSERT: 2,
* STATE\_EDIT: 3,
* STATE\_DELETE: 4

Tako da proverite da li je objekat u edit stanju:

item.item.state == 2

ili:

item.item\_state == item.task.consts.STATE\_INSERT

ili:

item.is\_new()

##### log\_changes

Koristite *log\_changes* da kontrolišete da li promene napravljene na datasetu objekta će biti upisani. Kada *log\_changes* je *true* (podrazumevano), sve promene se upisuju. One kasnije mogu biti primenjene na aplikativni serverr pozivom *apply* metod. Kada je *log\_changes* je *false*, promene podataka neće biti upisane i ne mogu biti priemnjene na aplikativni server.

##### rec\_no

Proverite *rec\_no* svojstvo za odredjivanje broj tekućeg zapisa u datasetu objekta. *rec\_no* može biti korišćen za repozicioniranje kurzora na spec. zapis.

##### table\_name

Pročitajte ovog svojstvo da bi dobili ime odgovarajuće tabele iz projektne baze podataka.

##### virtual\_table

Koristite read-only *virtual\_table* svojstvo da pronadjete da li objekat ima tabelu u bazi podataka. Ako je *virtual\_table* *True* ne postoji tabela u bazi podataka. Možete koristiti ovakav objekat za rad sa datasetom u memoriji ili koristiti njegove module za pisanje koda. Pozivanje *open* metoda kreira prazan dataset, a pozvanje *apply* metoda ne radi ništa.

#### Metodi

##### append(*self*)

Otvara novi, prazan zapis na kraju dataseta. Posle poziva *append*, aplikacija može omogućiti korisnicima da unesu podatke u polja zapisa, a zatim mogu da postuju te promene u dataset objekata koristeći *post* metod, a zatim da ih primene na tabelu baze podataka objekta, koristeći *apply* metod.

*append* metoda:

* proverava da li je skup podataka stavke *active*, u suprotnom pokreće izuzetak,
* ako je objekat *detalj*, proverava da li je *master* objekat u *edit* ili *insert* stanju, u suprotnom pokreće izuzetak,
* ako objekat nije *detalj* proverava da li je u *browse* stanju, u suprotnom pokreće izuzetak,
* otvora novi, prazan zapis na kraju dataseta,
* postavlja objekat u *insert* stanje.

##### apply(self, connection=None, params=None, safe=False):

Upisuje sve ažurirane, umetnute i izbrisane zapise iz dataseta objekta u bazu podataka.

*apply* metoda:

* proverava da li je objekat *detalj*, ako jeste vraća (*master* upisuje promene detalja),
* proverava da li je objekat u *edit* ili *insert* stanju, ako jeste objavljuje zapis,
* proverava da li *change log* ima promene, ako nema vraća,
* pokreće event hendler *on*\_*before*\_*apply* ako je definisan za objekat,
* ako je *connection* parametar *None*, poziva metod *connection* taska da bi dobio konekciju iz poola,
* ako je event hendler *on*\_*apply* taska definisan, izvršava ga,
* ako je event hendler *on\_apply* definisan za objekat izvršava ga,
* generiše i izvršava sql upit za upis promena u bazu podataka koristeći konekciju,
* ako connection parametar nije naveden, upisuje promene u bazu podataka i vraća vezu u konekcijoni pool,
* posle upisivanja promena u bazu podataka, ažurira *change log* i dataset objekta - ažurira vrednosti primarnog ključa novih zapisa,
* pokreće event hendler *on*\_*after*\_*apply* ako je definisan za objekat.

*Parametri*

* connection - ako je spec., aplikacija ga koristi za izvršavanje sql upita koji generiše, u suprotnom nabavlja vezu iz konekcionog poola taska, koja će biti vraćena u pool posle izvršenja promena.
* params - koristite ovaj parametar da prosledite neke korisnički definisane opcije koje će se koristiti u event hendleru *on\_apply*. Ovaj parametar mora biti objekat parova ključ/vrednost.
* safe - ako je *True*, metoda proverava da li korisnik koji je pozvao metod ima pravo da kreira, uređuje ili briše zapise u tabeli baze podataka objekta, ako ne, pokreće izuzetak. Podrazumevana vrednost je *False*.

*Primeri*

U drugom primeru ispod, promene su sačuvane u transakciji.

def change\_invoice\_date(item, item\_id):

inv = item.copy()

cust = item.task.customers.copy()

inv.set\_where(id=item\_id)

inv.open()

if inv.record\_count():

now = datetime.datetime.now()

cust.set\_where(id=inv.customer.value)

cust.open()

inv.edit()

inv.invoice\_datetime.value = now

inv.post()

inv.apply()

cust.edit()

cust.last\_action\_date.value = now

cust.post()

cust.apply()

def change\_invoice\_date(item, item\_id):

con = item.task.connect()

try:

inv = item.copy()

cust = item.task.customers.copy()

inv.set\_where(id=item\_id)

inv.open()

if inv.record\_count():

now = datetime.datetime.now()

cust.set\_where(id=inv.customer.value)

cust.open()

inv.edit()

inv.invoice\_datetime.value = now

inv.post()

inv.apply(con)

cust.edit()

cust.last\_action\_date.value = now

cust.post()

cust.apply(con)

finally:

con.commit()

con.close()

##### bof(*self*)

Testirajte *bof* (*beginning of file*) metod da odredite poziciju kurzora u datasetu objekta. Ako *bof* vrati *true*, kurzor je nedvosmisleno na poziciji pre prvog zapisa u datasetu. *bof* vraća *false* u svim drugim slučajevima.

Napomena:

Ako i eof i bof vrate *True*, dataset objekta je prazan.

##### can\_create(*self*)

Koristite *can\_create* metod da odredite da li korisnik tekuće sesije ima prava da kreira zapis.

*Primer*

def send\_email(item, selected, subject, mess):

if not item.can\_create():

raise Exception('You are not allowed to send emails.')

#code sending email

##### can\_delete(*self*)

Koristite *can\_delete* metod da odredite da li korisnik tekuće sesije ima pravo da obriše zapis.

##### can\_edit(*self*)

Koristite *can\_edit* metod da odredite da li korisnik tekuće sesije ima prava da edituje zapisa.

##### cancel(*self*)

Pozovite *cancel* da poništite promene napravljene na jednom ili više polja tekućeg zapisa, ako promene nisu već objavljene post metodom dataseta objekat.

*Cancel* metod:

* pokreće *on\_before\_cancel* event handler ako je definisan za objekat,
* poništava promene napravljene na tekućem zapisu i njegovim detaljima ako je zapis bio editovan ili uklanja novi zapis bio dodan ili insertovan.
* postavlja objekat u *browse* stanje
* pokreće *on\_after\_cancel* event handler ako je definisan za objekat.

##### clear\_filters(*self*)

Koristite *clear\_filters* da postavie *filter* vrednosti objekta na *None*.

##### close(*self*)

Pozovite *сlose* da zatvorite dataset objekta. Pošto je dataset zatvoren *active* svojstvo je *false*.

##### copy(self, filters=True, details=True, handlers=True)

Koristite *copy* da kreirate kopiju objekta. Kreirana kopija nije dodata u task stablo i biće uništena od Python garbage collector kada više nije potrebna. Svi atributi kopije objekta su definisani kao što su bili u vreme kreiranja task stabla.

Metod može imati sledeće parametri:

* handlers – ako je *True*, sve funkcije i eventi definisani u server modulu objekta biće raspoloživi u kopiji. Podrazumevano je *True*.
* filters - ako je *True*, svi filteri definisani u server modulu objekta biće raspoloživi u kopiji. Podrazumevano je *True*.
* details - ako je *True*, svi detalji definisani u server modulu objekta biće raspoloživi u kopiji. Podrazumevano je *True*.

*Example*

def on\_generate(report):

cust = report.task.customers.copy()

cust.open()

report.print\_band('title')

for c in cust:

firstname = c.firstname.display\_text

lastname = c.lastname.display\_text

company = c.company.display\_text

country = c.country.display\_text

address = c.address.display\_text

phone = c.phone.display\_text

email = c.email.display\_text

report.print\_band('detail', locals())

##### delete(*self*)

Briše aktivni zapis i pozicionira kursor n sledeći zapis. Delete metod:

* checks if item dataset is active, otherwise raises exception
* checks if item dataset is not empty, otherwise raises exception
* if item is a detail , checks if the master item is in edit or insert state, otherwise raises exception.
* if item is not a detail , checks if it is in browse state, otherwise raises exception.
* puts the item into delete state
* deletes the active record and positions the cursor on the next record
* puts the item into browse state

##### edit(*self*)

Omogućuje editovanje podataka u datasetu. Posle pozivanja *edit*, aplikacija može omogućiti korisnicima da promene podatke u poljima zapisa, i mogu tada objaviti promene na datasetu objekta koristeći post metod, i posle toga primeniti promene na bazu podataka koristeći *apply* metod.

*edit* metod

* proverava da li je objekat *active*, inače podiže izuzetak,
* proverava da li dataset objekat nije prazan, inače podiže izuzetak,
* proverava da li dataset objekta je već u *edit* stanju, ako jeste vraća,
* ako je objekat *detalj*, proverava da li je *master* objekat u *edit* ili *insert* stanju, inače podiže izuzetak
* ako objekat nije detalj, proverava da li je u browse stanju, inače podiže izuzetak,
* postavlja objekat u *edit* stanje, omogućujući aplikaciji ili korisniku da promeni polja zapisa.

##### eof(*self*)

Testirajte *eof* (*end of file*) metod da odredite poziciju kurzora u datasetu objekta. Ako *eof* vrati *true*, kurzor je nedvosmisleno na poziciji posle poslednjeg zapisa u datasetu. *eof* vraća *false* u svim drugim slučajevima.

Napomena: Ako i *eof* i *bof* vrate *True*, dataset objekta je prazan.

##### field\_by\_name(*self*, *field\_name*)

Pozovite *field\_by\_name* da vratite informacije o polju, ako je samo ime polja poznato. *field\_name* parametar je ime polja. *field\_by\_name* vraća field objekat za spec. ime polja. Ako spec. polje ne postoji *field\_by\_name* vraća *None*.

##### filter\_by\_name(*self*, *filter\_name*)

Pozovite *filter\_by\_name* da vratite informacije o filteru, ako je samo ime polja poznato. *filter\_name* parametar je ime filtera. *filter\_by\_name* vraća filter objekat za spec. ime filtera. Ako spec. folter ne postoji *filter\_by\_name* vraća *None*.

##### first(*self*)

Pozovite *first* da pozicionirate kurzor na prvi zapis u datasetu objekta i napravite ga aktivnim zapisom. *First* objavljuje bilo koju promenu na tekućem zapisu.

##### insert(*self*)

Insertuje novi, prazan zapis u dataset objekta. Posle poziva *insert*, aplikacija može omogućiti korisnicima da unose podatke u polja zapisa, i potom da objave promene na dataset objekta, koristeći *post* metod, i potom da primene promene na bazu podataka, koristeći *apply* metod.

*insert* metod:

* proverava da li je dataset *active*, inače podiže izuzetak,
* ako je objekat *detalj*, proverava da li je *master* objekat u *edit* ili *insert* stanju, inače podiže izuzetak,
* ako objekat nije detalj, proverava da li je objekat u *browse* stanju, inače podiže izuzetak
* insertuje novi, prazan zapis u dataset objekta,
* postavlja objekat u *insert* stanje.

##### is\_changing(*self*)

Proverava da li je objekat u *edit* ili *insert* stanju i vraća *True* ako jeste.

##### is\_edited(*self*)

Proverava da li je objekat u *edit* stanju i vraća True ako jeste.

##### is\_modified(*self*)

Proverava da li je tekući zapis dataset objekta bio promenjen za vreme edit ili insert operacija. Metod vraća false posle *post* metod poziva.

##### is\_new(*self*)

Proverava da li je objekat u *insert* stanju i vraća True ako jeste.

##### last(*self*)

Pozovite *last* da repozicionirate kursor na poslednji zapis u datasetu objekta i napravite ga aktivnim zapisom. *Last* objavljuje sve promene na aktivnom zapisu.

##### locate(self, fields, values)

Implementira metod za pretragu dataseta objekta za spec. zapis i pravi ga aktivnim.

*Parametri*:

* fields: ime polja ili lista imena polja
* values: vednost polja ili lista vrednosti polja

Ovaj metod locira zapis po spec. kriterijumu. *locate* vraća *True* ako je zapis pronadjen i kursor je repozicioniran na zapis. Ako odgovarajući zapis nije pronadjen, kursor nije repozicioniran, metoda vraća false.

##### next(*self*)

Pozovite *next* da pozicionirate kursor na sledeći zapis u datasetu objekta i napravite taj zapis aktivnim. *Next* objavljuje sve promene na aktivnom zapisu.

##### open(self, options=None, expanded=None, fields=None, where=None, order\_by=None, open\_empty=False, params=None, offset=None, limit=None, funcs=None, group\_by=None, safe=False)

Pozovite *open* da biste generisali i izvršili *select* *sql* upit za tabelu baze podataka za dobijanje dataseta objekta. Metoda inicijalizuje polja objekta, formuliše parametre zahteva i pokreće event hendler *on*\_*before*\_*open* ako je on definisan za stavku. Ako je za objekat definisan event hendler *on\_open*, *open* izvršava ovaj event hendler i dodeljuje datasetu rezultat koji je vratio, u suprotnom generiše *select* *sql* upit, na osnovu parametara zahteva, izvršava ovaj upit i dodeljuje rezultat izvršenja datasetu. Posle toga postavlja *active* na True, *item\_state* na *browse*, ide na prvi zapis dataseta, pokreće *on*\_*after*\_*open*, ako je definisan za objekat.

*Parametri:*

Možete proslediti *options* rečnik da biste naveli parametre zahteva u istom obliku kao i za open metod na klijentu:

invoices.open({

'fields': ['customer', 'invoicedate', 'total'],

'where': {customer: customer\_id, invoicedate\_\_ge: date1, invoicedate\_\_le: date2},

'order\_by': ['invoicedate']

})

Ili prosledite argumente kao ključne reči:

invoices.open(

fields=['customer', 'invoicedate', 'total'],

where={customer: customer\_id, invoicedate\_\_ge: date1, invoicedate\_\_le: date2},

order\_by=['invoicedate']

)

* expanded - ako je *True*, upit će imati *Join* klauzulue za dobijanje vrednosti lookup polja, inače neće biti vrednosti lookup polja. Podrazumevano je *True*.
* fields - koristite ovaj parametar da odredite klauzulu *Select* upita. Ovaj parametar je lista imena polja. Ako je izostavljen, koristiće se polja definisana metodom *set\_field*. Ako metoda *set \_fields* nije pozvana pre izvršenja open metode, koristiće se sva dostupna polja.
* where - koristite ovaj parametar da odredite kako će se zapisi filtrirati u sql upitu. Ovaj parametar je rečnik, čiji su ključevi nazivi polja, koji su praćeni, posle dvostruke donje crte, simbolima za filtriranje. Ako je ovaj parametar izostavljen, koristiće se vrednosti definisane metodom *set\_where*. Ako metoda *set\_where* nije pozvana pre izvršenja *open* metode, a *where* parametar je izostavljen, tada će se vrednosti filtera definisane za objekat koristiti za filtriranje zapisa.
* order\_by - koristite *order*\_*by* za određivanje sortiranja zapisa. Ovaj parametar je lista imena polja. Ako se ispred naziva polja nalazi znak *-*, tada će zapisi biti sortirani po ovom polju i opadajućem redosledu. Ako je ovaj parametar izostavljen, koristiće se lista definisana metodom *set\_order\_by*.
* offset - koristite *offset* da odredite *offset* prvog zapisa.
* limit - koristite *limit* za ograničavanje izlaza sql upita na prvih *limit* zapisa.
* funcs - ovaj parametar može biti rečnik, čiji su ključevi nazivi polja, a vrednosti su imena funkcija koje će biti primenjene na polja u *select* upitu.
* group\_by - koristite *group*\_*by* za određivanje polja za grupisanje rezultata upita. Ovaj parametar mora biti lista imena polja.
* open\_empty - ako je *True*, aplikacija ne šalje zahtev serveru već samo inicijalizuje prazan skup podataka. Podrazumevana vrednost ako je *False*.
* params - koristite ovaj parametar da prosledite neke korisnički definisane opcije koje će se koristiti u *on*\_*open* event hendleru. Ovaj parametar mora biti objekat parova ključ/vrednost.
* safe - ako je podešen na *True* metod proverava da li korisnik koji je pozvao metod ima pravo da vidi podatke objekta, ako nema pkreće izuzetak. Podrazumevana vrednost je *False*.

*Primeri*

U ovom primeru parametri zahteva su rečnik:

import datetime

def get\_sales(item):

date1 = datetime.datetime.now() - datetime.timedelta(days=3\*365)

date2 = datetime.datetime.now()

invoices = item.task.invoices.copy()

invoices.open({

'fields': ['customer', 'date', 'total'],

'where': {'date\_\_ge': date1, 'date\_\_le': date2},

'order\_by': ['customer', 'date']

})

Ispod su parametri u obliku liste ključnih reči:

import datetime

def get\_sales(item):

date1 = datetime.datetime.now() - datetime.timedelta(days=3\*365)

date2 = datetime.datetime.now()

invoices = item.task.invoices.copy()

invoices.open(

fields=['customer', 'date', 'total'],

where={'date\_\_ge': date1, 'date\_\_le': date2},

order\_by=['customer', 'date']

)

Isti rezultat može biti postignut koristeči *set\_fields*, *set\_where*, *set\_order\_by* metode:

import datetime

def get\_sales(item):

date1 = datetime.datetime.now() - datetime.timedelta(days=3\*365)

date2 = datetime.datetime.now()

invoices = item.task.invoices.copy()

invoices.set\_fields('customer', 'date', 'total')

invoices.set\_where(date\_\_ge=date1, date\_\_le=date2);

invoices.set\_order\_by('customer', 'date');

invoices.open();

def get\_sales(item):

date1 = datetime.datetime.now() - datetime.timedelta(days=3\*365)

date2 = datetime.datetime.now()

invoices = item.task.invoices.copy()

invoices.set\_fields([’customer’, ’date’, ’total’])

invoices.set\_where({’date\_\_ge’: date1, ’date\_\_le’: date2});

invoices.set\_order\_by([‘customer’, ‘date’]); invoices.open();

def get\_sales(task) {

sales = task.invoices.copy()

sales.open(fields=['customer', 'id', 'total'],

funcs={'id': 'count', 'total': 'sum'},

group\_by=['customer'],

order\_by=['customer'])

##### post(*self*)

Upisuje promenjeni zapis u dataset objekta. Pozovite *post* da sačuvate promene napravljene na zapisu posle poziva *append*, *insert* ili *edit* metoda.

*post* metod:

* proverava da li je objekat u *edit* ili *insert* stanju, inače podiže izuzetak,
* pokreće *before\_post* event handler ako je definisan za objekat,
* proverava da li zapis validan, ako nije podiže izuzetak,
* ako objekat ima detalje, detalje tekućeg zapisa,
* dodaje promene u *change log* objekta,
* postavlja objekat u *browse* stanje,
* pokreće *on\_after\_post* event handler ako je definisan za objekat.

##### prior(self)

Poziva *prior* da pozicionira zapis na prethodni zapis u datasetu objekta i napravi ga aktivnim zapisom. *last* objavljuje sve promene na aktivnom zapisu.

##### record\_count()

Pozovite *record\_count* da dobijete ukupan broj zapisa u datasetu objekta.

*Primer*

item.open()

if item.record\_count():

# some code

##### set\_fields(self, lst=None, \*fields)

Koristite *set\_fields* metod da definišete i smestite interno fields parametar koji će biti korišćen od *open* metoda, kada njegov sopstveni *fields* parametar nije spec. *open* metod čisti interno smeštenu vrednost. *Fields* je lista imena polja. Možete spec. polja kao listu, ili kao pozicione argumente:

*Primeri*

Rezultat izvršenja sledećih kodaova je isti:

item.open(fields=['id', 'invoicedate'])

ili:

item.set\_fields('id', 'invoicedate')

item.open()

##### set\_order\_by(*self*, *lst=None*, *\*fields*)

Koristite *set\_order\_by* metod to definišete i smestite interno *order\_by* parameter koji će biti korišćen od *open* metoda, kada njegov sopstveni *order\_by* parametar nije spec. *open* metod čisti interno smeštenu vrednost. Možete spec. polja kao listu, ili kao pozcione argumente: Ako postoji znak – pre imena polja, tada će dataset biti sortiran u opadajućem poretku po tom polju.

*Primer*

Rezultat izvršavanja sledećeg koda je isti:

item.open(order\_by=['customer', '-invoicedate'])

item.set\_order\_by('customer', '-invoicedate')

item.open();

##### set\_where(self, dic=None, \*\*fields)

Koristite *set\_where* metod to definišete i smestite interno *where* filtere koji će biti korišćeni od *open* metoda, kada njegov sopstveni *where* parametar nije spec. *open* metod čisti intreno smeštenu vrednost. Možete spec. filtere kao rečnike, ili pozicione parametre.

*Primer*

Resultat izvršenja koda će biti isti:

import datetime

date = datetime.datetime.now() - datetime.timedelta(days=3\*365)

item.open(where={'customer': 44, 'invoicedate\_\_gt': date})

-------------------------------------------------------------------

import datetime

date = datetime.datetime.now() - datetime.timedelta(days=3\*365)

item.set\_where({'customer': 44, 'invoicedate\_\_gt': date})

item.open()

-------------------------------------------------------------------

import datetime

date = datetime.datetime.now() - datetime.timedelta(days=3\*365)

item.set\_where(customer=44, invoicedate\_\_gt=date)

item.open()

#### Eventi

##### on\_apply(self, delta, params, connection)

Napišite *on\_apply* event handler kada vam je potrebno da prepišete standardni način upisa pri izvršavanju *apply* metoda na klijentu ili serveru.

*on\_apply* event handler ima sledeće parametre:

* item - reference na objekat,
* delta - delta sadrži objektov change log,
* params - parametere prosledjene na server iz *apply* metoda,
* connection – konekcija koja će biti korišćena da upiše promene u bazu podataka.

*delta* parametar sadrži promene koje moraju biti sačuvane u bazi podataka. Po sebi, ova opcija je kopija objekta, i njegov dataset je objektov *change log*. Priroda promena zapisa može biti dobijena korišćenjem sledećih metoda:

* *rec\_inserted*
* *rec\_modified*
* *rec\_deleted*

svaki od kojih vraća *True*, ako je zapis dodat, promenjen ili obrisan, respektivno. Ako objekat ima detalje, *delta* takodje ima odgovarajući detalj objekat, koji smešta promene na detalj objektu.

Napomena: Primetite da kada je zapis obrisan iz dataseta objekta i taj zapis ima zapise detalja, *change log* će će zadržati samo taj obrisani zapis, informacija o obrisanim zapisima detalja nisu smeštene. Da dodate te obrisane zapisa detalja, pozovite *delta* *update\_deleted* metod. Ne treba da otvarate *delta* *detail* pošto je kursor prešao na drugi zapis.

Polja *delta* dataseta imaju *old\_value* atribut koji može biti iskorišćen za dobijanje vrednosti polja pre nego što su promene napravljene.

Kada *on\_apply* event handler nije definsan *apply\_delta* metod je izvršen, koji generiše sql upite i izvršava ih. Posle tog vraća informaciju o rezultatu procesiranja, što smešta id-ove novih zapisa. Klijent na osnovu informacija ažurira objektov change log i vrednosti primarnih ključeva novih zapisa.

Kada *on\_apply* event handler vrati *None* *apply\_delta* je izvršen.

Možete napraviti dodatno procesiranje *delte*. U sledećem kodu, vrednost date polja je postavljena na tekući datum pre nego što su promene primenjene na tabelu baze podataka.

import datetime

def on\_apply(item, delta, params, connection):

for d in delta:

d.edit()

d.date.value = datetime.datetime.now()

d.post()

Napomena:

Primetite da promene napravljene na ovaj nain klijent ne vidi. Možete koristiti *refresh\_record* ili *refresh\_page* da prikašete te promene.

U sledećem kodu, dak upisuje promene napravljene na invoices, aplikacija ažurira vrednost *tracks\_sold* polja za trekove u invoces. Sve je obavljeno u transakciji:

def on\_apply(item, delta, params, connection):

tracks = item.task.tracks.copy()

changes = {}

delta.update\_deleted()

for d in delta:

for t in d.invoice\_table:

if not changes.get(t.track.value):

changes[t.track.value] = 0

if t.rec\_inserted():

changes[t.track.value] += t.quantity.value

elif t.rec\_deleted():

changes[t.track.value] -= t.quantity.value

elif t.rec\_modified():

changes[t.track.value] += t.quantity.value - t.quantity.old\_value

ids = list(changes.keys())

tracks.set\_where(id\_\_in=ids)

tracks.open()

for t in tracks:

q = changes.get(t.id.value)

if q:

t.edit()

t.tracks\_sold.value += q

t.post()

tracks.apply(connection)

U prethodnom primeru *on\_apply* event handler vraća None tako da se izvršava *apply\_delta* metod.

U generalnom slučaju:

def on\_apply(item, delta, params, connection):

# execute some code before changes are written to the database

result = item.apply\_delta(delta, params, connection)

# execute some code after changes are written to the database

return result

##### on\_open(item, params)

Napišite *on\_open* event handler kada je potrebno da prepišite standardan način dobijanja zapisa dataseta za vreme izvršenja *open* metoda na klijentu ili serveru.

*on\_open* event handler ima sledeće parametre:

* item - referenca na objekat,
* params – rečnik koji sadrži parametre prosledjene na server open metodom:
  + \_\_expanded – odgovara *expanded* parametru server open metoda / *expanded* atributu *options* parametru klijent open metoda,
  + \_\_fields – lista imena polja,
  + \_\_filters - lista objekata, od kojih svaki je lista sa sledećim članovima:
    - *ime* polja
    - *filter* *konstantu* iz filtriranja zapisa
    - *filter* *vrednost*
  + \_\_funcs – rečnik funkcija
  + \_\_order – lista objekata. Od kojih svaki je lista sa sledećim članovima:
    - ime polja
    - boolen vrednost, ako je *True* poredak je opadajući
  + \_\_offset – odgovara offset parametru *open* metoda
  + \_\_limit – odgovara *limit* parametru *open* metoda
  + \_\_client\_request – je *True* kada zahtev dolazi sa klijenta.

*params* može takodje uključiti korisnički definisane parametre prosledjene open metodi.

Ispod je primer *params* koji klijent *open* metod invoices objekta šalje na server:

{

'\_\_fields': [u'id', u'deleted', u'customer', u'firstname', u'date',

u'subtotal', u'taxrate', u'tax', u'total',

u'billing\_address', u'billing\_city', u'billing\_country',

u'billing\_postal\_code', u'billing\_state'],

'\_\_filters': [[u'customer', 7, [6]]],

'\_\_expanded': True,

'\_\_limit': 11,

'\_\_offset': 0,

'\_\_order': [[u'date', True]]

}

{

'\_\_fields': [u'id'],

'\_\_funcs': {u'id': u'count'},

'\_\_filters': [],

'\_\_expanded': False,

'\_\_offset': 0,

'\_\_order': [],

'\_\_summary': True

}

Server aplikacija generiše sql upit, zasnovan na *params* i izvršava ga. Server vraća klijentu rezultujuće zapise i *error* poruku ako se dogodi za vreme izvršenja.

### *class* Detail

*Detail* klasa nasledjuje atribute, metode i evente *Item* klasu.

#### Atributi

##### master

Koristite *master* atribut da dobijete referencu na *master* objekat *detalja*.

### *class* Reports

Reports klasa se koristi za kreiranje grupe objekata task stabla koja je vlasnik izveštaja projekta. Ispod su eventi. Klasa nasledjuje *AbstractItem* klasu.

#### Eventi

##### on\_convert\_report(*report*)

Okvir konvertuje izveštaje interno, koristeći LibreOffice. Koristite *on\_convert\_report* event ako želite da koristite neki drugi servis ili promenite neke parametre konverzije izveštaja. *report* parametar je report koji pokreće event.

*Primer*

import os

from subprocess import Popen, STDOUT, PIPE

def on\_convert\_report(report):

try:

if os.name == nt:

import \_winreg

regpath = SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths\\soffice.exe

root = \_winreg.OpenKey(\_winreg.HKEY\_LOCAL\_MACHINE, regpath)

s\_office = \_winreg.QueryValue(root, )

else:

s\_office = soffice

convertion = Popen([s\_office, '--headless', '--convert-to', report.ext,

report.report\_filename, '--outdir', os.path.join(report.task.work\_dir, 'static', 'reports') ],

stderr=STDOUT,stdout=PIPE)

out, err = convertion.communicate()

converted = True

except Exception as e:

print(e)

### *class* Report

Ispod su atributi, metode i eventi klase. Klasa nasledjuje atribute i metode od *AbstractItem* klase.

#### Atributi

##### report\_filename

Kada je *template* atribut izveštaja spec., *generate* metod upisuje sadržaj generisanog izveštaja u fajl u report folderu static foldera i postavlja vrednost *report\_filename* atributa na ime sačuvanog fajla. Njegova vrednost može biti korišćena u *on\_after\_generate*.

##### report\_url

*Generate* metod šalje klijentu vrednost smeštenu u ovaj atribut kao *url* generisanog fajla. Kada je *template* atribut izveštaja spec., vrednost je postavljena od *generate* metoda pošto je sačuvan generisan sadržaj. Inače programer ga mora sam postaviti.

##### template

*report\_filename* atribut smešta ime fajla šablona izveštaja. Obično je to postavljeno u AppBbuildera kada je izveštaj kreiran. Može biti promenjen dinamički na serveru u *on\_before\_generate* event handleru ili biti prazno, ako je neophodno da kreira, na primer, neki txt fajl.

#### Metodi

##### generate(*self*)

Ovaj metod se koristi interno za generisanje sadržaja izveštaja. Kada server dobije zahtev sa klijenta da generiše izveštaj, prvo kreira kopiju izveštaja i potom kopija poziva metod. Ovaj metod pokreće *on\_before\_generate event*. Ako je šablon izveštaja definisan, parsira ga i pokreće *on\_parsed* i *on\_generate* evente. U *on\_generate* event handleru programer bi trebao da napiše kod koji će da generiše sadržaj izveštaja i sačuva ga u *ods* fajl, koristeći *print\_band* metod da štampa bandove.

Kada je izveštaj generisan i vrednost extension atributa izveštaja postavljena na klijentu, nije jednaka *ods*, server pokušava da konveruje ods fajl. Za konverziju fajla, prvo proverava da li grupa izveštaja (vlasnik izveštaja) ima *on\_convert\_report* event handler.ako je taj event hendler definisan koristi ga za konverziju. Inače koristi za konverziju LibreOffice instaliran na server u headless modu.

Posle toga aplikacija sačuva generisani izveštaj u fajl u report folder static foldera, postavi vrednost *report\_filename* atributa na ime sačuvanog fajla, generiše vrednost *report\_url* atributa i pokreće *on\_after\_generate* event. Pošto je izveštaj generisan i sačuvan u report folderu static foldera server šalje klijentu *url* fajla izveštaja.

Ako je atribut template izveštaja nije postavljen, server pokreće *on\_generate* i posle toga *on\_after\_generate* evente. U tom slučaju trebalo bi sačuvati generisani sadržaj u sam fajl i postaviti vrednost *report\_url* atribut.

##### hide\_columns(*self*, *col\_list*)

Koristite *hide\_columns* metod da sakrijete neke kolone definisane u šablonu izveštaja. *col\_list* parametar spec. koje kolone će biti skrivene. Ova je inigera ili slova, definišući poziciju kolona izveštaja. Koristite ovaj metod u *on\_parsed* event handleru.

*Example*

def on\_parsed(report):

report.hide\_columns(['A', 'C')

# report.hide\_columns([1, 3])

##### print\_band(*self*, *band*, *dic=None*)

Koristite *print\_band* metod da postavite vrednosti progamibilnih ćelija banda definisanih u šablonu izveštaja. I dodajte band u sadržaj izveštaja.

Metod ima sledeće parametre:

* band – spec. ime banda koji će biti štampan,
* dic – rečnik, koji sadrži vrednosti pridružene programabilim ćelijama banda.

*Primer*

Sleeći kod generiše sadržaj Customer izveštaja u Demo aplikaciji:

def on\_generate(report):

cust = report.task.customers.copy()

cust.open()

report.print\_band('title')

for c in cust:

firstname = c.firstname.display\_text

lastname = c.lastname.display\_text

company = c.company.display\_text

country = c.country.display\_text

address = c.address.display\_text

phone = c.phone.display\_text

email = c.email.display\_text

report.print\_band('detail', locals())

#### Eventi:

##### ****on\_after\_generate(report)****

***on\_after\_generate*** event je pokrenut od generate metoda, kada je izveštaj generisan i sačuvan u fajl čije ime je smešteno u *report\_filename* atribut. ***report*** parametar je izveštaj koji pokreće event.

##### on\_before\_generate(report)

***on\_before\_generate*** event je pokrenut od *generate* metoda, pre generisanje izveštaja. ***report*** parametar je izveštaj koji pokreće event.

##### on\_generate(report)

***on\_generate*** event je pokrenut od *generate* metoda. Napišite *on\_generate* event handler da generišete sadržaj izveštaja. Koristite *print*\_*band* metod da štampate bandove, definisane u šablonu izveštaja. ***report*** parametar je izveštaj koji je pokrenuo event.

##### on\_parsed(report)

***on\_parsed*** event je pokrenut od *generate* metoda, pošto je šablon izveštaja parsiran. Koristite ovaj event hendler da sakrijete neke kolone definisane u šablonu izveštaja pozivanjem *hide*\_*columns*. ***report*** parametar je izveštaj koji je pokrenuo event.

### *class* Field

#### Atributi i svojstva

##### display\_text

Predstavlja vrednost polja kao string. *display*\_*text* svojstvo je string prikaz vrednosti polja samo za čitanje da bi se prikazao korisnicima. Ako je dodeljen event hendler *on\_get\_field\_text***, *display\_text*** je vrednost koju vraća ovaj event hendler. Inače, *display\_text* je vrednost svojstva *lookup*\_*text* za lookup polja i svojstva *text* konvertovana u skladu sa postavkama jezika za druga polja.

*display*\_*text* je string predstava svojstva vrednosti polja kada se ne uređuje. Kada se polje uređuje, koristi se svojstvo *text*.

*Primer*

def on\_generate(report):

cust = report.task.customers.copy()

cust.open()

report.print\_band('title')

for c in cust:

firstname = c.firstname.display\_text

lastname = c.lastname.display\_text

company = c.company.display\_text

country = c.country.display\_text

address = c.address.display\_text

phone = c.phone.display\_text

email = c.email.display\_text

report.print\_band('detail', locals())

##### field\_caption

*Field\_caption* atribut spec. ime polja koje se pojavljuje korisnicima.

##### field\_name

Spec. ime polja koje se koristi u kodu.

##### field\_size

Identifikuje veličinu polja text tipa.

##### field\_type

Identifikuje tip objekta polja. Koristite *field\_type* atribut da saznate tip podatka koje polje sadrži. To je jedna od sledećih vrednosti:

* text,
* integer,
* float,
* currency,
* date,
* datetime,
* boolean,
* blob

##### lookup\_text

Koristite *lookup\_text* svojstvo da dobijete lookup vrednost lookup polja konvertovanu u string. Ako je polje lookup polje daje *lookup\_text*, inače vraća vrednost *text* svojstva.

##### lookup\_value

Koristite *lookup\_value* svojstvo da dobijete lookup vrednost lookup polja. Ako je polje lookup polja daje lookup vrednost, inače daje vrednost *value* svojstva.

##### owner

Identifikuje objekat kome polje pripada.

##### raw\_value

Predstavlja podatke field objekta. Koristite *raw\_value* read only svojstvo da čitate podataka direktno iz dataseta objekta. Druga svojstva kao što je *value* i *text* koriste konverziju. *Value* svojstvo konvertuje *null* vrednost u 0 za numerička polja i u prazan string za polja tipa *text*.

##### read\_only

Odredjuje da li je polje modifikovano. Postavite *read\_only* na *True* da sprečite polje da bude modifikovane u kontrolama svesnim podataka.

##### required

Spec. da li neprazna vrednost za polje zahtevana. Koristite *required* da odredite da li polje zahteva vrednost ili polje može biti prazno. Kada je *required* svojstvo *True*, pokušaj da *post* *null* vrednost će izazvati da izuzetak bude podignut.

##### text

Koristite *text* svojstvo da dobijete ili postavite vrednost polja. Da dobijete vrednost text svojstva polja.

##### value

Koristite *value* svojstvo da dobijete ili postavite vrednost value svojstva polja.

###### Dobijanje vrednosti

Kada je vrednost podataka polja *null*, polje konvertuje na 0, ako je *field\_type* *integer*, *float* ili *currency*, ili prazan string ako je *field\_type* *text*. Za llokup polja, vrednost ovog svojstva je integer koji je vrednost id polja odgovarajućeg zapisa lookup objekta. Da dobijete lookup vrednost polja koristite lookup\_value svojstvo.

###### Postavljanje vrednosti

Kada je nova vrednost pridružena, polje proverava da li je tekuća vrednost različita od one nove. Ako jeste:

* postavi *new\_value* atributa na tu vrednost,
* pokreće *on\_before\_field\_changed* event ako je definisan za polje,
* promeni podatak polja na vrednost *new\_value* atributa i postavi ga na null,
* označi objekat kao promenjen, tako da *is\_modified* metod će vratiti *True,*
* pokreće *on\_field\_changed* event ako je definisan za polje,
* ažurira kontrole svesne podataka.

### *class* Filter

#### Atributi i svojstva

##### filter\_name

Spec. ime filtera koje se koristi u kodu.

##### owner

Identifikuje objekat koji je vlasnik filtera.

##### value

Koristite *value* svojstvo da dobijete ili postavite vrednost filtera.

*Primer*

function on\_view\_form\_created(item) {

item.filters.invoicedate1.value =

new Date(new Date().setYear(new Date().getFullYear() - 1));

}