# UnCommon Web ile "Merhaba Dünya"

Zekeriya KOÇ <zkoc@metkoi.com>

**Özet**

Bu yazı, Common Lisp ile geliştirilmiş olan UnCommon Web geliştirme çatısına basit bir örnek aracılığı ile giriş seviyesinde bilgi vermek amacındadır.

# İçindekiler

# 1 Giriş - Kurulum

Ne zamandır fırsat bulamadığım UnCommon Web geliştirme çatısı ile bir kaç deneme yapabildim sonunda. Kurcalamaya başlamak için önce UCW'i kurmam gerekti doğal olarak. Oldukça fazla sayıda olan bağımlı olduğu paketleri tek tek kurmak yerine http://common-lisp.net/project/UCW/UCW-boxset.tar.gz adresinden UCW-boxset paketini indirdim. Windows sistemimde çeşitli hatalar aldığımdan sanal makine üzerindeki Debian sistemime kurdum. Kurdum derken ev klasörümde arşiv dosyasını açtım sadece. Gerisi UCW-boxset klasöründeki "start.lisp" dosyasını Lisp sistemine yüklemekten ibaret zaten. Veritabanı erişimi için (malum web programlama veritabanı olmadan olmaz) clsql paketini kullandım. Emacs içinde "M-x slime" komutu ile başlattığım oturumuma önce Uncommon web çatısını,

```
(load "/home/zekus/dev/lisp/UCW-boxset/start.lisp")
```

komutu ile yükledim. Ardından çok basit bir SQL veritabanı olan sqlite3 veritabanına erişmek için kullanacağım clsql-sqlite3 paketini yükledim. Verdiğim komut;

```
(asdf:operate 'asdf:load-op 'clsql-sqlite3)
```

start.lisp dosyasını yüklediğimde UCW'in gömülü web sunucusu 8080 numaralı porttan istekleri dinlemeye başlamış olmalıydı. Test etmek için tarayıcıdan "http://localhost:8080" adresine girdim. Örnek UCW uygulamaları karşıma gelince kurulumumun sorunsuz bir şekilde gerçekleştiğini anlamış oldum.

# 2 UnCommon Web

Şimdi kendimce yazdığım örnek uygulamanın satır satır neler yaptığına bir bakalım;

```
(in-package :it.bese.UCW-user)
(defvar *zekus-deneme-1* (make-instance 'cookie-session-application :url-prefix
"/zekus/"))
(register-application *default-server* *zekus-deneme-1*)
(defentry-point "deneme1.UCW"
  (:application *zekus-deneme-1*)
  ()
  (call 'deneme-component))
```

Burada ilk olarak çalışacağımız paketi belirtiyoruz ki UCW'in fonksiyonlarına, makrolarına, değişkenlerine paket ön ekini belirtmeden erişebilelim. Ardından *zekus-deneme-1* isimli global bir değişkene oluşturduğumuz uygulamayı atıyoruz. Uygulamamızın tarayıcıdan erişimi için kullanılacak olan ön eki de burada "/zekus/" olarak belirtiyoruz. UCW'in öntanımlı sunucusunun uygulamamızdan haberdar olması için uygulamayı kayıt ediyoruz (register-application ...). Ve son olarak uygulamamıza bir başlangıç noktası belirtiyoruz. Bu nokta

"deneme1.UCW" olarak belirlendi. Yani uygulamayı bitirip derlediğimizde tarayıcı adres çubuğuna "HTTP://localhost:8080/zekus/deneme1.UCW" yazarak uygulamamızı çalıştıracağız. Peki bunu yaptığımızda ne olacak? Hangi fonksiyonumuz sayfayı icra edecek? Bunun cevabı "defentrypoint" makrosunun gövdesinde verilmiş. Giriş noktası olarak "deneme-component" isimli bileşen kontrolü alacak ve sayfayı inşa edecek.

Buraya kadar uygulamamızın ilk tanımlamalarını yaptık. Şimdi giriş noktasından göreve çağrılan "deneme-component" isimli bileşeni tanımlayalım. Bunun için defcomponent isimli bir makro kullanıyoruz.

```
(defcomponent deneme-component (simple-window-component)
  ((mesaj :accessor mesaj :initarg :mesaj :initform nil)
   (sorgu-veri :accessor sorgu-veri :initarg :sorgu-veri :initform nil)
  (sorgu-kolon :accessor sorgu-kolon :initarg :sorgu-kolon :initform nil)))
```

Bileşenimiz "simple-window-component" isimli UCW'in sağladığı bir bileşenden türüyor ve üç tane yeni slot tanımlıyor. Bu kısım sadece bileşenimizin tanımı. Ama bileşenimizin kendini (sayfayı) nasıl inşa edeceğini "render" isimli bir metod ile belirleyeceğiz. Bu metod "deneme-component" bileşeninin "sanatçı" elleri. İşte kodu:

```
(defmethod render ((d deneme-component))
  (<:hr)
  (<:p (<:h1 "Merhaba Dünya!!!!!"))
  (<:hr)
  (<:h5 "by zekUs")
  (<UCW:a :action (deneme d) "Deneme")
  (<UCW:a :action (deneme1 d) "Deneme Sil")
  (<UCW:a :action (sorgula d) "Veritabanından Sorgula Getir")
  (<UCW:a :action (veritabani-sifirla d) "Veritabanı Sıfırla")
 (<UCW:a :href "HTTP://zekus.metkoi.com" :target "_a" "zekUs")
  (when (mesaj d)
    (<:br)(<:as-HTML (mesaj d)))
    (when (and (sorgu-veri d) (sorgu-kolon d))
      (<:table :border "1" :cellpadding "3" :cellspacing "0"
        (<:tr
        (dolist (kolon (sorgu-kolon d))
          (<:th (<:as-HTML kolon))))
          (dolist (satir (sorgu-veri d))
```

```
(<:tr
  (dolist (veri satir)
    (<:td (<:as-HTML veri))))))))
```

render metodu "deneme-component" isimli bileşenin o andaki bir örneğini parametre olarak alır(d). Yani bileşenin slotlarına bu metod içinden erişebileceğiz. YACLML isimli bir paket yardımı ile burada doğrudan HTML kodu yazıyoruz. Biraz alıştıktan sonra düz HTML yazmaktan çok farklı bir durum olmuyor. UCW'in TAL şablon tanımlama dilini kullanma özelliği de var fakat ben bunu henüz denemedim. Ayrıca Seaside kullanan Ramon Leon'un kullandığı programlama dili içinde (aynı sözdizimi ile)HTML yazmanın şablon kullanmaktan daha verimli olduğu yönündeki argümanlarını ikna edici bulduğumdan önce bu yöntemde biraz ilerlemek istiyorum. Yukarıdaki kodda UCW'i ilgilendiren bölüm tanımladığımız bağlantılara atadığımız "action"lar. Son örnekteki bağlantı klasik url ile yönlendirilen bir bağlantı fakat önceki ikisi "deneme-component" bileşenimiz için tanımlayacağımız "action"lara birer çağrı içeriyorlar. Örnek kodumuzun son parçası da bu "action"lara ait:

```
(defaction deneme ((d deneme-component))
  (setf (mesaj d) "denemeAction"))

(defaction deneme1 ((d deneme-component))
  (setf (mesaj d) nil))

(defaction sorgula ((d deneme-component))
  (clsql:connect '("/home/zekus/dev/lisp/UCW_code/deneme.db") :database-
type :sqlite3 :if-exists :old)
  (multiple-value-bind (veri kolon) (clsql:query "select * from tablo")
    (setf (sorgu-veri d) veri) (setf (sorgu-kolon d) kolon)))

(defaction veritabani-sifirla ((d deneme-component))
  (when (clsql:connected-databases)
    (clsql:disconnect)
    (setf (sorgu-kolon d) nil)
    (setf (sorgu-veri d) nil)))
```

Sayfadaki bağlantılara tıkladığınızda (href ile tanımlanan bağlantı haricinde) bu koddaki ilgili aksiyon devreye girer ve gövdesindeki kodu işletir. İşi bittiğinde kontrolü tekrar çağırıldığı bileşene teslim eder ve bileşen kendisini (sayfayı ya da sayfanın bir kısmını) inşa eder. Aksiyonlar da aynı render metodu gibi kendilerini çağıran bilşenenin o anki örneğini parametre olarak alırlar ve bileşen

slotlarında değişiklik yapabilirler(yukarıdaki kod bunu yapıyor).

Örnek koda genel olarak baktığımızda şunu görürüz, bir bileşen tanımlanmış. Bu bileşeni bir masaüstü programının "container" tipi bir bileşen olarak düşünebilirsiniz. Buna düğmeler eklenmiş (web bağlantıları). Ve her düğmeye basıldığında düğmenin altına yazılan kod çalışıyor (aksiyonlar). Arada web programlamaya özgü kontroller, problemler ve hamallıklarla uğraşmıyorsunuz ve probleme odaklanmanız kolaylaşıyor. Tabii ki bunlar benim şahsi düşüncelerim ama Seaside ile ilk kez farkettiğim bu harika yaklaşım Common Lisp ile de çok güçlü bir şekilde karşımızda.

Not: UCW ve Lisp hakkında çok derin bir bilgim yok. Bu yüzden olası hataları, önerilerinizi, eklemek istediklerinizi yorum olarak belirtmekten çekinmeyin.

Uncommon Web geliştirme çatısını kullanan bazı uygulamalara bu adreslerden ulaşabilir ve inceleyebilirsiniz: http://www.paragent.com/

Uncommon Web ile aynı programlama yaklaşımını kullanan, özgür bir Smalltalk gerçekleştirimi olan Squeak üzerinde Seaside web geliştirme çatısı kullanılarak yazılan bir uygulama: DabbleDB