

# Porting a Swing application to DukeScript using NetBeans

Ioannis Kostaras  
NetBeans Day Athens  
21 April 2017

# Agenda

TodoDS

- Prerequisites
- Port to DukeScript

## *References:*

- **TodoDS** <http://wiki.netbeans.org/ToDoDS>

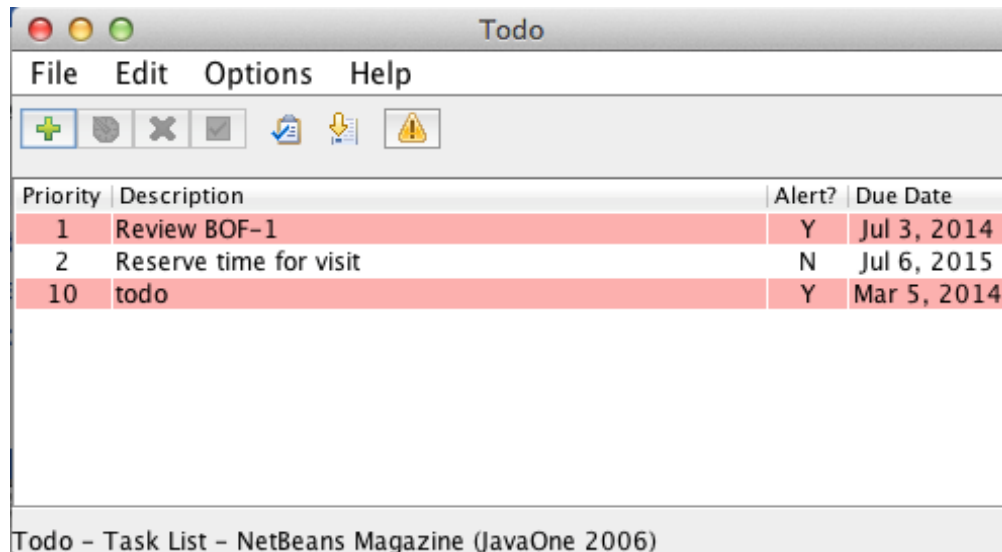
# Prerequisites

TodoDS

- [ToDo Swing application](#)
  - [NetBeans 8.0.2](#) or later
  - [JDK 7](#) or later
  - [HSQL DB](#)
  - [DukeScript](#) plugin for NetBeans
- 
- Create a Project Group (*Optional*)

# To-do Swing application

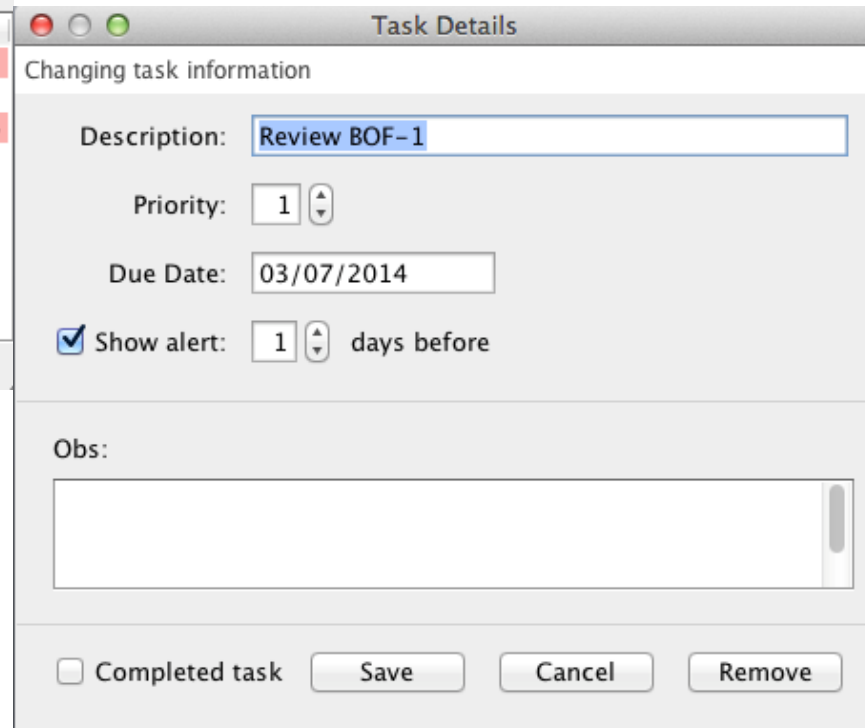
TodoDS



The 'Todo' application window has a title bar with standard Mac OS window controls (red, yellow, green buttons). Below the title bar is a menu bar with 'File', 'Edit', 'Options', and 'Help'. Underneath the menu bar is a toolbar with icons for adding a task (+), deleting a task (trash), toggling a task (X), marking a task as done (checkbox), and a warning icon. The main area contains a table with the following data:

Priority	Description	Alert?	Due Date
1	Review BOF-1	Y	Jul 3, 2014
2	Reserve time for visit	N	Jul 6, 2015
10	todo	Y	Mar 5, 2014

At the bottom of the window, there is a status bar that reads 'Todo - Task List - NetBeans Magazine (JavaOne 2006)'.



The 'Task Details' dialog box has a title bar with standard Mac OS window controls. The main area is titled 'Changing task information'. It contains the following fields and controls:

- Description: A text field containing 'Review BOF-1'.
- Priority: A numeric field with a value of '1' and up/down arrow buttons.
- Due Date: A date field containing '03/07/2014'.
- Show alert: A checked checkbox followed by a numeric field with a value of '1' and up/down arrow buttons, and the text 'days before'.
- Obs: A large empty text area for observations.
- At the bottom, there are four buttons: 'Completed task' (with an unchecked checkbox), 'Save', 'Cancel', and 'Remove'.

# Requirements

TodoDS

- ❑ Tasks should have a *priority*, so users can focus on higher-priority tasks first.
- ❑ Tasks should have a *due date*, so users can focus on tasks that are closer to their deadline.
- ❑ Tasks that are either late or near their deadlines should have visual cues.
- ❑ Tasks can be marked as *completed*, but this doesn't mean they have to be deleted or hidden.

# Requirements (cont.)

TodoDS

- The to-do application consists of two main windows:
  - A task list window and
  - A task editing form

FILE EDIT OPTIONS

+ - ✓ ↓ !

PRI	TASK	DVE

STATUS

TASK:

PRI:

DVE:

OBS:

SAVE CANCEL

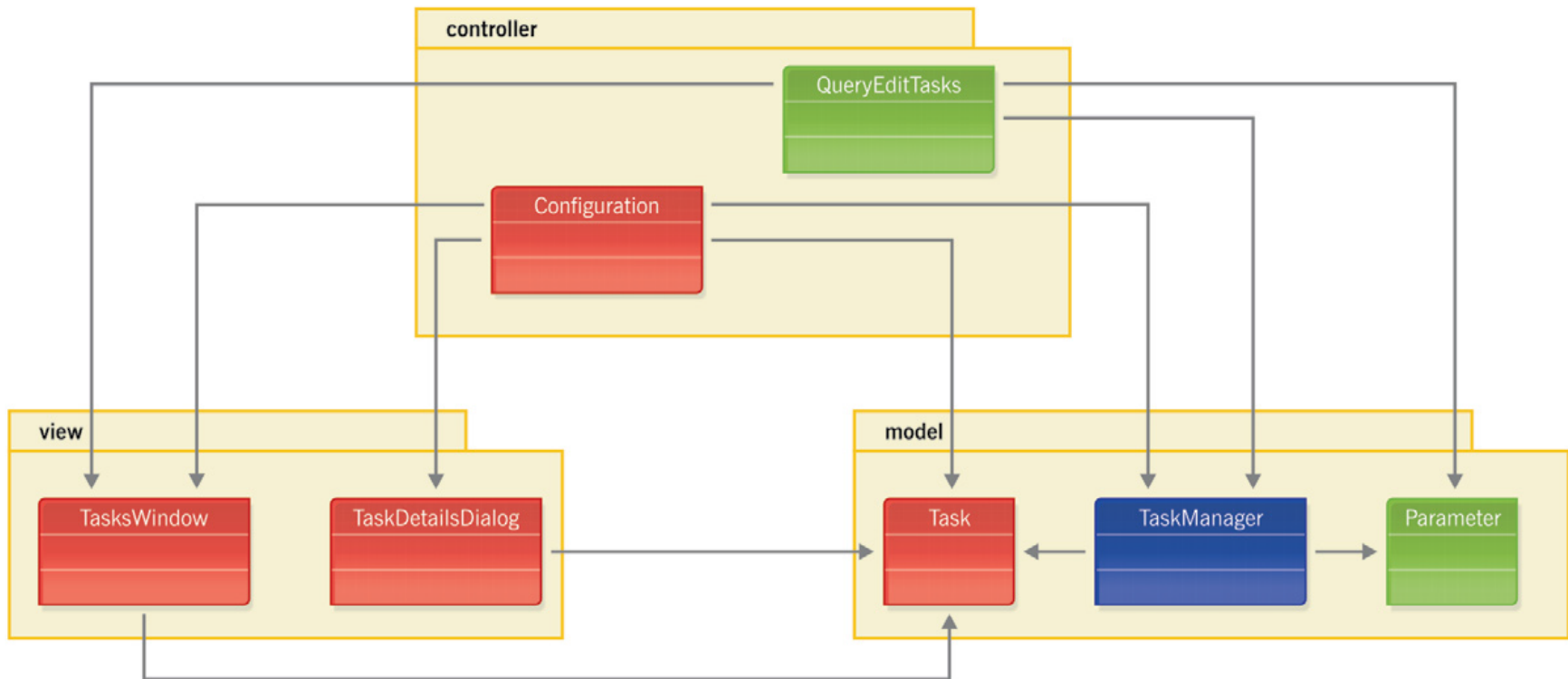
# Steps

TodoDS

1. Build a "static" visual prototype of the GUI.
2. Build a "dynamic" prototype of the application, coding user interface events and associated business logic and creating customized GUI components as needed.
3. Code the persistence logic by modeling the classes and the database.

# Swing ToDo application Architecture

TodoDS





# TODO DS

# What is DukeScript

TodoDS

- DukeScript is a new technology for creating cross-platform mobile, desktop and web applications. It allows you to write your logic in Java and render the result to a number of clients, which can be web browser, portable devices etc.
- DukeScript applications are plain Java applications that internally use HTML5 technologies and JavaScript for rendering. This way developers only need to write clean Java code and can still leverage the latest developments in modern UI technology.

# How does it work

TodoDS



# Pros & Cons

TodoDS

- + Write in Java
- + Write once run everywhere (web, JavaFX, Android, iOS, ...)
- + API similar to JavaFX
- Not a lot of documentation available
- Need to learn a new API

# Technologies to master

---

TodoDS

- HTML(5)
- CSS(3)
- JavaScript
- Knockout.js
- DukeScript
- Model-View-ViewModel (MVVM)

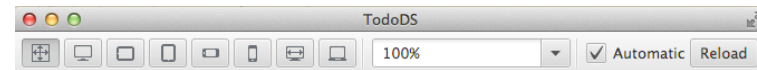
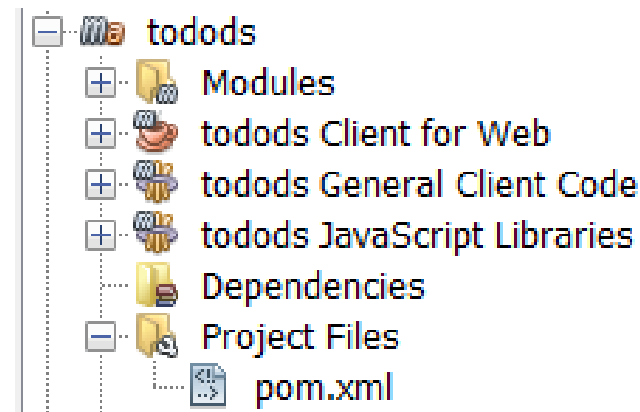
- In NetBeans install the DukeScript plugin
  - Tools → Plugins → Available Plugins
    - ☐ DukeScript Project Wizard
    - ☐ Mine Sweeper

# Create a new DukeScript application

TodoDS

## ➤ In NetBeans:

- File → New Project
  - ☐ Categories: DukeScript
  - ☐ Projects: DukeScript Application
- Provide an artifact & group id
- Choose your platform: Browser
- Select *Knockout for Java Example* template
- Right click on it and choose **Build with Dependencies.**
- Execute *todo General Client Code*
- Contains:
  - ☐ 2 Java files: `Main` and `DataModel`
  - ☐ 1 HTML file: `index.html`
  - ☐ 1 empty CSS file: `index.css`



# STEP 1

**Build a Static Prototype of the GUI**



# Step 1: Build a Static Prototype of the GUI

TodoDS

- DukeScript claims to have a clean separation of design and development. With DukeScript it is possible to completely outsource the UI design to a designer with no knowledge of DukeScript, or a specific set of tools.
- Dukescript uses HTML for the framework's UI and there are plenty of tools to build HTML UIs with the help of CSS and it is a well known technology to UI designers.
- Using a nice `.css` we can produce the following prototype:



## Tasks

Priority	Description	Alert?	Due Date
10	Finish TodoDS article!	true	10/03/2017
5	Book conference room.	false	01/04/2017

**There are 1 task(s) with alerts today.**











# Step 1: Build a Static Prototype of the GUI

TodoDS

- Copying the icons from the original Todo Swing application and with some more HTML editing, our tasks-list page is almost ready:



## Tasks

   				
Priority	Description	Alert?	Due Date	
10	Finish TodoDS article!	true	10/03/2017	  
5	Book conference room.	false	01/04/2017	  

There are 1 task(s) with alerts today.

# Step 1: Build a Static Prototype of the GUI

TodoDS

- Create `edit.html` for the edit task form and link the two pages together using standard HTML.

## Create/Edit Task

Description:

Priority:

Due Date:

☐ Show alert:  days before

Obs:

☐ Completed Task

# STEP 2

## Build a Dynamic Prototype

# Step 2: Build a Dynamic Prototype

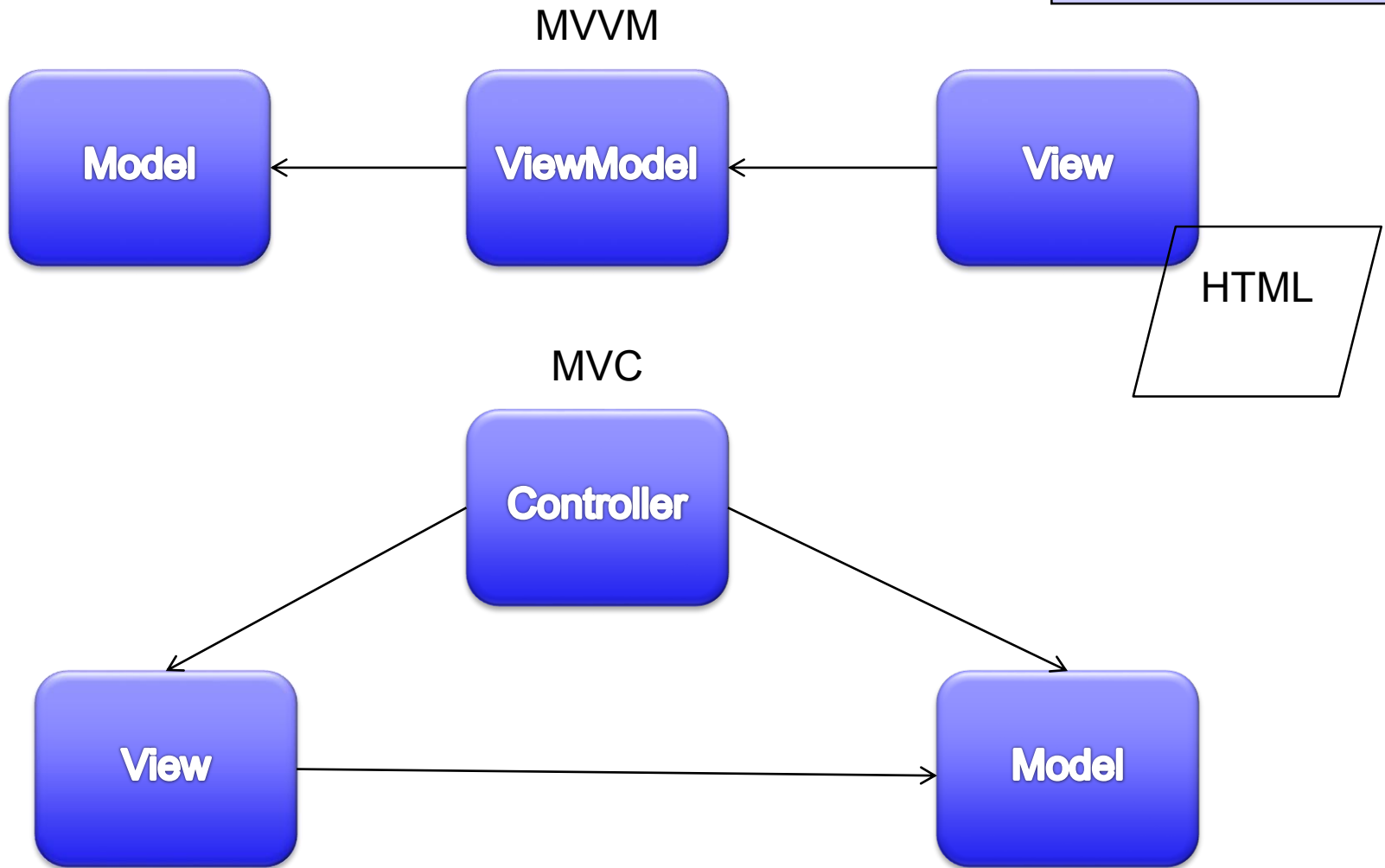
TodoDS

- implement as much user interaction as possible without using persistent storage or implementing complex business logic.

Swing Todo	TodoDS
MVC	MVVM
Value Object (VO): Task	ViewModel: Task
DAO: TaskManager	ViewModel: TaskList

# Model-View-ViewModel (MVVM)

TodoDS



## Step 2: todods.TODODS.Main

TodoDS

```
public final class Main {  
    private Main() { }  
    public static void main(String... args) throws  
Exception {  
        BrowserBuilder.newBrowser().  
            loadPage("pages/index.html").  
            loadClass(Main.class).  
            invoke("onPageLoad", args).  
            showAndWait();  
        System.exit(0);  
    }  
    public static void onPageLoad() throws Exception {  
        ViewModel.onPageLoad();  
    }  
}
```

## Step 2: todods.TODODS.DataModel

TodoDS

```
@Model(className = "Data", targetId="",
    properties = { })

final class DataModel {
    private static Data ui;
    /** * Called when the page is ready. */
    static void onPageLoad() throws
    Exception {
        ui = new Data();
        ui.applyBindings();
    }
}
```



## Step 2: todods.TODODS.ViewModel

TodoDS

```
@Model(className = "Task", targetId="", properties = {
    @Property(name = "id", type = int.class),
    @Property(name = "description", type = String.class),
    @Property(name = "priority", type = int.class),
    @Property(name = "dueDate", type = String.class),
    @Property(name = "alert", type = boolean.class),
    @Property(name = "daysBefore", type = int.class),
    @Property(name = "obs", type = String.class),
    @Property(name = "completed", type = boolean.class) })

final class ViewModel {
    static void onPageLoad() throws Exception {
        Task task = new Task(); task.setPriority(10);
        task.setDescription("Finish TodoDS article!");
        task.setAlert(true); task.setDueDate("10/03/2017");
        task.applyBindings();
    }
}
```

## Step 2: todods.TODODS.ViewModel

T O D O S

```
@Model(className = "Task", targetId="", properties = {
    @Property(name = "id", type = int.class),
    @Property(name = "description", type = String.class),
    @Property(name = "priority", type = int.class),
    @Property(name = "dueDate", type = String.class),
    @Property(name = "alert", type = boolean.class),
    @Property(name = "daysBefore", type = int.class),
    @Property(name = "obs", type = String.class),
    @Property(name = "completed", type = boolean.class) })

final class ViewModel {
    static void onPageLoad() throws Exception {
        Task task = new Task(); task.setPriority(10);
        task.setDescription("Finish TodoDS article!");
        task.setAlert(true); task.setDueDate("10/03/2017");
        task.applyBindings();
    }
}

// <div class="rTableCell" data-bind="text: priority"></div>
```

## Step 2: todods.TODO.S.ViewModel

T O D O S











```
@Model(className = "TaskList", targetId = "", properties = {
@Property(name = "tasks", type = Task.class, array = true) })
final class ViewModel {
    static void onPageLoad() throws Exception {
        TaskList taskList = new TaskList();
        taskList.add(new Task(...));
        taskList.applyBindings();
    }
    @Model(className = "Task", targetId = "", properties = {...})
    public static class TaskModel { }
}

<div class="rTable" data-bind="foreach: tasks">
-----
<div class="rTable">
    <div class="rTableHeading"> ... </div>
    <!-- ko foreach: tasks -->
<div class="rTableRow" > ... </div> <!-- /ko -->
```

# Step 2: Build a Dynamic Prototype of the GUI

TodoDS

## Tasks

   				
Priority	Description	Alert?	Due Date	
10	Finish TodoDS article!	true	10/03/2017	  
Priority	Description	Alert?	Due Date	
7	Book venue!	false	11/01/2017	  

There are 1 task(s) with alerts today.

## Step 2: add user interaction

```
public void removeTask(final int id) {  
    tasks.removeIf(task -> id == task.getId());  
} →
```

### @Function

```
public static void removeTask(TaskList tasks, Task data) {  
    tasks.getTasks().remove(data);  
}
```

```
-----  
<a class="button" data-bind="click: $parent.removeTask">  
      
</a>
```

## Step 2: add user interaction (cont.)

```
@ComputedProperty
public static int numberOfTasksWithAlert(List<Task> tasks)
{
    return listTasksWithAlert(tasks).size();
}

private static List<Task> listTasksWithAlert(List<Task>
tasks) {
return
    tasks.stream().filter(Task::isAlert).collect(toList());
}
```

Java 8 not supported  
by bck2brwsr yet

```
-----
<div class="rTableFoot">There are
    <label data-bind="text:
        $data.numberOfTasksWithAlert"/></label> task(s) with
    alerts today.
</div>
```

➤ Properties available only in View:

- `$root`: refers to the top-level ViewModel
- `$data`: refers to the ViewModel object of the current context (can be omitted)
- `$parent`: refers to the parent ViewModel object (useful for nested loops)
- `$index`: contains the current item's index in the array

- Applications written with DukeScript typically are single pages, and the scope of a Model is a single page.
- Still we need a way to mimic the behaviour that you typically get in a web application with several linked HTML-pages, like our `index.html` and `edit.html`.
- To overcome this problem we use [Knockout templates](#).
- The template binding has a `name` parameter. Knockout will look for a script tag with the same `id` as specified by the `name` parameter:

```
<div data-bind="template: {name: 'task'}"></div>
<script type="text/html" id="task">
  <h2>Tasks</h2>
  ...
</script>
```

Make sure that the content of the script tag won't be executed as Javascript.



# Templates (2/4)

```
<div data-bind="template: {name: 'task',  
  if: !edited()} "></div>  
<div data-bind="template: {name: 'editor', if: edited(),  
  data: edited()} "></div>  
<script type="text/html" id="task">  
  <h2>Tasks</h2>  
  
  ...  
</script>  
<script type="text/html" id="editor">  
  ...  
</script>
```

# Templates (3/4)

```
@Property(name = "selected", type = Task.class),
@Property(name = "edited", type = Task.class)

@Function static void addNew(TaskList tasks) {
    tasks.setSelected(null);
    tasks.setEdited(new Task());
}

@Function static void edit(TaskList tasks, Task data) {
    tasks.setSelected(data);
    tasks.setEdited(data.clone());
}
```

# Templates (4/4)

```
@Function
static void commit(TaskList tasks) {
    final Task task = tasks.getEdited();
    if (task == null) return;
    final Task selectedTask = tasks.getSelected();
    if (selectedTask != null) {
        tasks.getTasks().set(tasks.getTasks().indexOf(selectedTask),
task);
    } else {
        tasks.getTasks().add(task);
    }
    tasks.setEdited(null);
}

@Function
static void cancel(TaskList tasks) {
    tasks.setSelected(null);
    tasks.setEdited(null);
}
```

# Validation (1/3)

TodoDS

```
private static boolean validate(Task task) {  
    String invalid = null;  
    if (task.getValidate() != null) {  
        invalid = task.getValidate();  
    }  
    return invalid == null;  
}
```

TasksViewModel

...

@Function

```
static void commit(TaskList tasks) {  
    final Task task = tasks.getEdited();  
    if (task == null || !validate(task)) {  
        return;  
    }  
}
```

...

# Validation (2/3)

TodoDS

```
@ComputedProperty
static String validate(String description, int priority,
    String dueDate, int daysBefore) {
    String errorMsg = null;
    if (description == null || description.isEmpty()) {
        errorMsg = "Specify a description";
    }
    ...
    if (errorMsg == null && (daysBefore < 0 || daysBefore
    > 365)) {
        errorMsg = "Days before must be an integer in the
    range 0-365";
    }
    return errorMsg;
}
```

TaskModel

# Validation (3/3)

TodoDS

## Create/Edit Task

Description:

Priority:

Due Date:

☒ Show alert:  days before

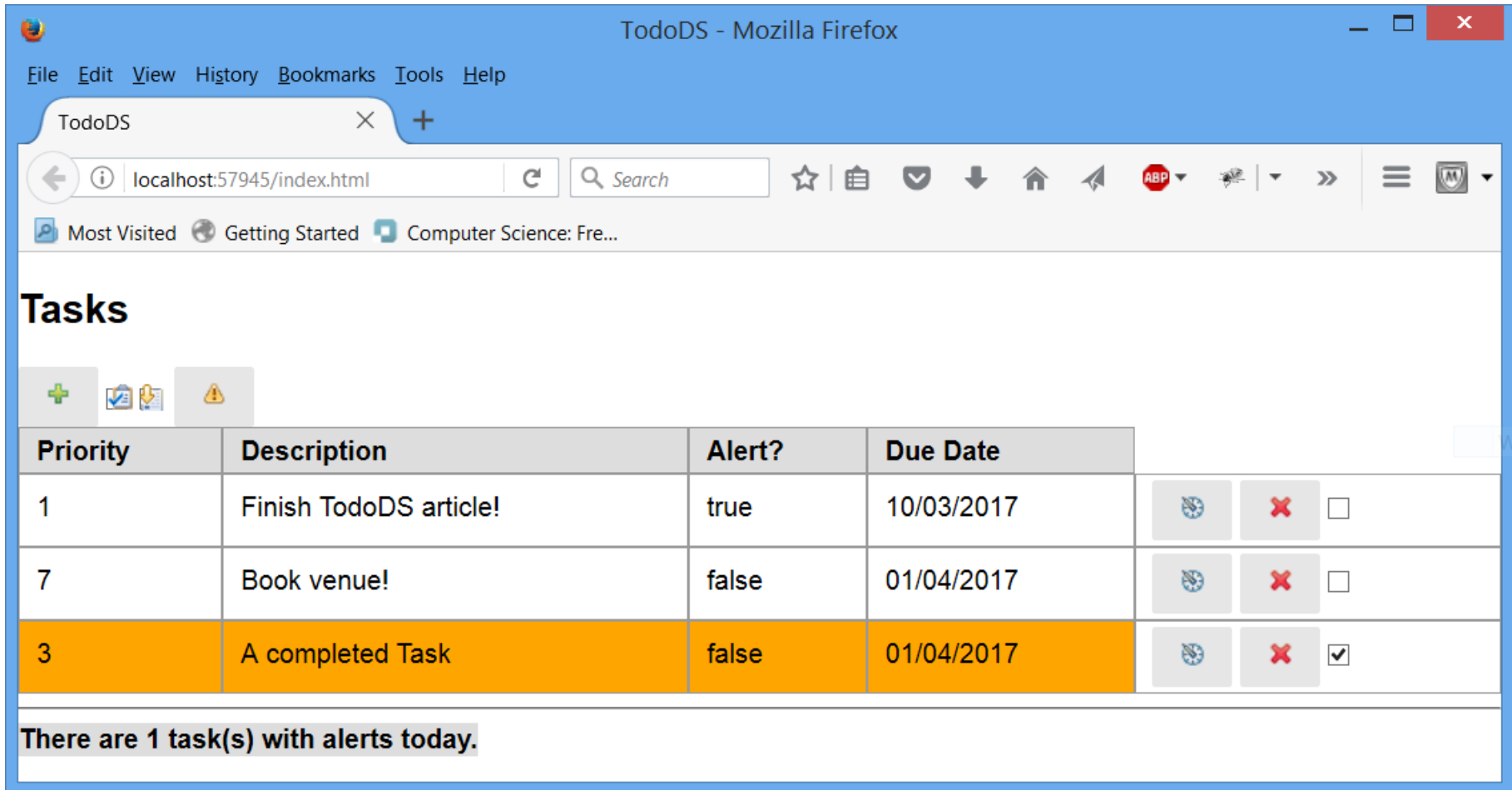
Obs:

☐ Completed Task

Priority must be an integer in the range 1-10

# In the browser

TodoDS



The screenshot shows the TodoDS application in a Mozilla Firefox browser window. The address bar displays 'localhost:57945/index.html'. The page title is 'TodoDS'. The application interface includes a menu bar (File, Edit, View, History, Bookmarks, Tools, Help) and a toolbar with various icons. Below the toolbar, there is a section titled 'Tasks' with a sub-header 'Tasks'. Under this header, there are three icons: a green plus sign, a document with a checkmark, and a yellow warning triangle. The main content is a table with four columns: 'Priority', 'Description', 'Alert?', and 'Due Date'. The table contains three rows of tasks. The first row has a priority of 1, description 'Finish TodoDS article!', alert status 'true', and due date '10/03/2017'. The second row has a priority of 7, description 'Book venue!', alert status 'false', and due date '01/04/2017'. The third row has a priority of 3, description 'A completed Task', alert status 'false', and due date '01/04/2017'. To the right of the table, there are three columns of icons: a circular arrow, a red 'X', and a checkbox. The first two rows have the red 'X' icon, while the third row has the checkbox icon checked. Below the table, there is a summary text: 'There are 1 task(s) with alerts today.'

Priority	Description	Alert?	Due Date			
1	Finish TodoDS article!	true	10/03/2017			<input type="checkbox"/>
7	Book venue!	false	01/04/2017			<input type="checkbox"/>
3	A completed Task	false	01/04/2017			<input checked="" type="checkbox"/>

There are 1 task(s) with alerts today.

# STEP 3

## Add Persistence



# Step 3: Add Persistence Logic

TodoDS

1. Add `hsqldb.jar` to your project's repository:
  - a) Right-click on *Runtime Dependencies*
  - b) Add Dependency
  - c) Fill the form as shown in figure
  - d) Click **Add**
  - e) **Clean and Build** or **Build with Dependencies**

The screenshot shows the 'Add Dependency' dialog box in NetBeans. The fields are filled as follows:

- Group ID: `org.hsqldb`
- Artifact ID: `hsqldb`
- Version: `2.3.4`
- Scope: `runtime` (dropdown menu)
- Type: `jar`
- Classifier: (empty)

The 'Search' tab is selected. Below the search bar, there is a 'Query' field and a 'Search Results' list. At the bottom right are 'Add' and 'Cancel' buttons.

# Step 3: Add Persistence Logic

TodoDS

- Copy persistent `TaskManager` from the original *Todo* application
- Make it a singleton
- Copy `Parameters`, `ModelException`, `ValidationException` and `DatabaseException`
- Adapt the `ViewModel` to use `TaskManager`

```
static void onPageLoad(PlatformServices services) throws
    Exception {
    TaskList taskList = new TaskList();
    taskList.setSelected(null);
    taskList.setEdited(null);
    List<Task> tasks =
    TaskManager.getInstance().listAllTasks(true);
    for (Task task : tasks) {
        taskList.getTasks().add(task);
    }
    taskList.applyBindings();
}
```

# Common Misconceptions about DukeScript

- “DukeScript is just a GWT clone”
  - GWT is a web toolkit that you can use it for writing web applications. You write Java, and it's compiled to JavaScript. Then it's typically deployed to a server and you run it in a browser.
  - DukeScript's is pure client technology: You write your application and it's business logic in Java which is compiled to Java bytecode. The bytecode is running in a normal JVM (Desktop, Dalvik, RoboVM, TeaVM etc).
- “DukeScript has no access to JavaScript”
  - Totally wrong; you can use any JavaScript library available for:
    - ❑ The view part (just reference the JavaScript library in the HTML as usual and use it in the View)
    - ❑ The Java part (use [@JavaScriptResource](#) and the [JavaScriptBody Annotation](#) to provide a typesafe way to call it's JavaScript functions from Java. See [online example](#)).

- “DukeScript is just for the client”
  - This is true, but there are annotations to make communication to the server as easy as possible (e.g. with the [@OnReceive](#) Annotation you can define a JSON communication endpoint in your view model). DukeScript ViewModel classes natively support JSON.

# References

- Lozano F. (2006), "A complete App using NetBeans 5", NetBeans Magazine, Issue 1, May,  
[http://netbeans.org/download/magazine/01/nb01\\_completeapp.pdf](http://netbeans.org/download/magazine/01/nb01_completeapp.pdf)
- Eppler A. (2016), *Java everywhere: Write Once Run Everywhere with DukeScript*, [LeanPub](#).
- Eppler A. (2015), "Java Everywhere: Write Once Run Anywhere with DukeScript", [JavaCodeGeeks](#).
- Eppler A. (2015), "[Common Misconceptions about DukeScript](#)".
- Kostaras I. (2016), [TodoDS](#)
- Kostaras I. (2015), [Port Your Java Applets](#)
- Hodson R. (2012), *Knockout.js Succintly*, Syncfusion.

# References

- Bauer et. al. (2016), Java Persistence with Hibernate, 2nd Ed., Manning.
- Coehlo H., Kiourtzoglou B., Java Persistence API Mini Book, JavaCodeGeeks.
- Goncalves A. (2013), Beginning Java EE 7, Apress.
- Goncalves A. (2013), "Generating Database Schemas with JPA 2.1", [Antonio's blog](#).
- Janssen T. (2016), "Standardized schema generation and data loading with JPA 2.1", [Thoughts-on-java](#).
- Keith M. & Schincariol M. (2013), Pro JPA 2 - Mastering the Java™ Persistence API, 2nd Ed., Apress.

# Q&A

TodoDS

