Building UIs with DART

Yohan BESCHI – Java Developer

@yohanbeschi

+Yohan Beschi

So@t

Building UIs with Dart

web_ui

&

Programmatic Components

Building UIs - Javascript ?

Building UIs - Java ?

Building UIs with Java - But how ?

vaadin }>

Programmatic Components with GWT

```
CellTable<User> table = new CellTable<User>();

TextColumn<User> idColumn = new TextColumn<User>() {
  @Override
  public String getValue(User user) {
              return user.id;
  }
};

TextColumn<User> firstNameColumn = new TextColumn<User>() {
  @Override
  public String getValue(User user) {
              return user.firstName;
  }
};

TextColumn<User> lastNameColumn = new TextColumn<User>() {
  @Override
  public String getValue(User user) {
              return user.lastName;
  }
};

TextColumn<User> ageColumn = new TextColumn<User>() {
  @Override
  public String getValue(User user) {
              return user.age;
  }
};

idColumn.setSortable(true);
firstNameColumn.setSortable(true);
lastNameColumn.setSortable(true);
ageColumn.setSortable(true);

table.addColumn(idColumn, "ID");
table.addColumn(firstNameColumn, "First name");
table.addColumn(lastNameColumn, "Lats name");
table.addColumn(ageColumn, "Age");

ListDataProvider<User> dataProvider = new ListDataProvider<User>();
dataProvider.addDataDisplay(table);

List<User> list = dataProvider.getList();
for (User user : USERS) {
  list.add(user);
}

ListHandler<User> columnSortHandler = new ListHandler<Tester.User>(list);
columnSortHandler.setComparator(idColumn,
              new Comparator<Tester.User>() {
                public int compare(User o1, User o2) {
                            if (o1 == o2) {
                              return 0;
                            }

                            if (o1 != null) {
                              return (o2 != null) ? o1.id.compareTo(o2.id) : 1;
                            }
                            return -1;
                }
              });
columnSortHandler.setComparator(firstNameColumn,
              new Comparator<Tester.User>() {
                public int compare(User o1, User o2) {
                            if (o1 == o2) {
                              return 0;
                            }

                            if (o1 != null) {
                              return (o2 != null) ? o1.firstName.compareTo(o2.firstName) : 1;
                            }
                            return -1;
                }
              });
columnSortHandler.setComparator(lasteNameColumn,
              new Comparator<Tester.User>() {
                public int compare(User o1, User o2) {
                            if (o1 == o2) {
                              return 0;
                            }

                            if (o1 != null) {
                              return (o2 != null) ? o1.lasteName.compareTo(o2.lasteName) : 1;
                            }
                            return -1;
                }
              });
columnSortHandler.setComparator(ageColumn,
              new Comparator<Tester.User>() {
                public int compare(User o1, User o2) {
                            if (o1 == o2) {
                              return 0;
                            }

                            if (o1 != null) {
                              return (o2 != null) ? o1.age.compareTo(o2.age) : 1;
                            }
                            return -1;
                }
              });
table.addColumnSortHandler(columnSortHandler);
table.getColumnSortList().push(firstNameColumn);
```

Programmatic Components with GWT

```
CellTable<User> table = new CellTable<User>();

TextColumn<User> idColumn = new TextColumn<User>() {
  @Override
  public String getValue(User user) {
              return user.id;
  }
};

TextColumn<User> firstNameColumn = new TextColumn<User>() {
  @Override
  public String getValue(User user) {
              return user.firstName;
  }
};

TextColumn<User> lastNameColumn = new TextColumn<User>() {
  @Override
  public String getValue(User user) {
              return user.lastName;
  }
};

TextColumn<User> ageColumn = new TextColumn<User>() {
  @Override
  public String getValue(User user) {
              return user.age;
  }
};

idColumn.setSortable(true);
firstNameColumn.setSortable(true);
lastNameColumn.setSortable(true);
ageColumn.setSortable(true);

table.addColumn(idColumn, "ID");
table.addColumn(firstNameColumn, "First name");
table.addColumn(lastNameColumn, "Lats name");
table.addColumn(ageColumn, "Age");

ListDataProvider<User> dataProvider = new ListDataProvider<User>();
dataProvider.addDataDisplay(table);

List<User> list = dataProvider.getList();
for (User user : USERS) {
  list.add(user);
}

ListHandler<User> columnSortHandler = new ListHandler<Tester.User>(list);
columnSortHandler.setComparator(idColumn,
              new Comparator<Tester.User>() {
                public int compare(User o1, User o2) {
                            if (o1 == o2) {
                              return 0;
                            }

                            if (o1 != null) {
                              return (o2 != null) ? o1.id.compareTo(o2.id) : 1;
                            }
                            return -1;
                }
              });
columnSortHandler.setComparator(firstNameColumn,
              new Comparator<Tester.User>() {
                public int compare(User o1, User o2) {
                            if (o1 == o2) {
                              return 0;
                            }

                            if (o1 != null) {
                              return (o2 != null) ? o1.firstName.compareTo(o2.firstName) : 1;
                            }
                            return -1;
                }
              });
columnSortHandler.setComparator(lasteNameColumn,
              new Comparator<Tester.User>() {
                public int compare(User o1, User o2) {
                            if (o1 == o2) {
                              return 0;
                            }

                            if (o1 != null) {
                              return (o2 != null) ? o1.lasteName.compareTo(o2.lasteName) : 1;
                            }
                            return -1;
                }
              });
columnSortHandler.setComparator(ageColumn,
              new Comparator<Tester.User>() {
                public int compare(User o1, User o2) {
                            if (o1 == o2) {
                              return 0;
                            }

                            if (o1 != null) {
                              return (o2 != null) ? o1.age.compareTo(o2.age) : 1;
                            }
                            return -1;
                }
              });
table.addColumnSortHandler(columnSortHandler);
table.getColumnSortList().push(firstNameColumn);
```

More than 100 lines

The Dart Way

```
Table<User> table = new Table (sorting:true)
  ..addColumn('ID', new TextCell((User o) => o.id))
  ..addColumn('First name', new TextCell((User o) => o.firstName))
  ..addColumn('Last name', new TextCell((User o) => o.lastName))
  ..addColumn('Age', new TextCell((User o) => o.age))
  ..setData(objs);
```

The Dart Way

```
Table<User> table = new Table (sorting:true)
  ..addColumn('ID', new TextCell((User o) => o.id))
  ..addColumn('First name', new TextCell((User o) => o.firstName))
  ..addColumn('Last name', new TextCell((User o) => o.lastName))
  ..addColumn('Age', new TextCell((User o) => o.age))
  ..setData(objs);
```

6 lines

Dart in few words

- Language
  - Object Oriented
  - Optionaly typed
  - Top-level functions
  - Functions as First-Class Objects
- Ecosystem
  - Client VM => Dartium
  - Server VM
  - Dart2js
  - DartEditor
  - Pub
  - DartDoc

UIs

Objectives

- School 1
- School 2
  - Grade 2.1
  - Grade 2.2
    - Person 2.2.1
    - Person 2.2.2
- School 3
  - Grade 3.1
- School 4

```
<ul>
  <li>School 1</li>
  <li>School 2
    <ul>
      <li>Grade 2.1</li>
      <li>Grade 2.2
        <ul>
          <li>Person 2.2.1</li>
          <li>Person 2.2.2</li>
        </ul>
      </li>
    </ul>
  </li>
  <li>School 3
    <ul>
      <li>Grade 3.1</li>
    </ul>
  </li>
  <li>School 4</li>
</ul>
```

Classes

```
class School {
  String schoolName;
  List<Grade> grades;

  School(this.schoolName,
         [this.grades]);
}
```

```
class Grade {
  String schoolGrade;
  List<Student> students;

  Grade(this.schoolGrade,
        [this.students]);
}
```

```
class Student {
  String firstname;
  String lastname;

  Student(this.firstname,
          this.lastname);
}
```

The Old-Fashioned Way

The Old-Fashioned Way

```
void main() {
  String tree = '<ul>';

  for (School school in schools) {
    tree += '<li>${school.schoolName}';

    // Grades

    tree += '</li>';
  }

  tree += '</ul>';

  query('body').insertAdjacentHtml('afterBegin', tree);
}
```

The Old-Fashioned Way

```
var grades = school.grades;
if (grades != null) {
  tree += '<ul>';

  for (Grade grade in grades) {
    tree += '<li>${grade.schoolGrade}';

    // Students

    tree += '</li>';
  }

  tree += '</ul>';
}
```

The Old-Fashioned Way

```
var students = grade.students;
if (students != null) {
  tree += '<ul>';

  for (Student student in students) {
    tree +=
        '<li>${student.firstname} ${student.lastname}</li>';
  }

  tree += '</ul>';
}
```

The Old-Fashioned Way

```
void main() {
  String tree = '<ul>';

  for (School school in schools) {
    tree += '<li>${school.schoolName}';

    var grades = school.grades;
    if (grades != null) {
      tree += '<ul>';

      for (Grade grade in grades) {
       tree += '<li>${grade.schoolGrade}';

      var students = grade.students;
      if (students != null) {
        tree += '<ul>';

        for (Student student in students) {
          tree += '<li>${student.firstname}
                      ${student.lastname}</li>';
        }

        tree += '</ul>';
      }
      tree += '</li>';
   }
```

```
      tree += '</ul>';
    }

    tree += '</li>';
  }

  tree += '</ul>';

  query('body')
    .insertAdjacentHtml('afterBegin', tree);
}
```

Introducing reusable components

Is there a pattern here ?

```
var grades = school.grades;
if (grades != null) {
  tree += '<ul>';

  for (Grade grade in grades) {
    tree += '<li>${grade.schoolGrade}';

    // Students

    tree += '</li>';
  }

  tree += '</ul>';
}
```

Is there a pattern here ?

```
var grades = school.grades;

if (grades != null) {
  tree += '<ul>';









  tree += '</ul>';
}
```

Is there a pattern here ?

```
var grades = school.grades;

if (grades != null) {
  tree += '<ul>';

  for (Grade grade in grades) {
    tree += '<li>${grade.schoolGrade}';



    tree += '</li>';
  }

  tree += '</ul>';
}
```

Is there a pattern here ?

```
var grades = school.grades;

if (grades != null) {
  tree += '<ul>';

  for (Grade grade in grades) {
    tree += '<li>${grade.schoolGrade}';

    // Do the same with children

    tree += '</li>';
  }

  tree += '</ul>';
}
```

Recursive Pattern

```
String doSomething(/* parameters */) {
  String tree = '';

  var grades = school.grades;
  if (grades != null) {
    tree += '<ul>';

    for (Grade grade in grades) {
      tree += '<li>${grade.schoolGrade}';
      tree += doSomething(/* parameters */);
      tree += '</li>';
    }

    tree += '</ul>';
  }

  return tree;
}
```

Side note – Functions & sugar syntax

```
int length(String s) {
    return s.length;
}
```

Side note – Functions & sugar syntax

```
int length(String s) {
    return s.length;
}
```

```
int length(String s)
  => s.length;
```

Easy use of reusable components

```
void main() {
  final Tree tree = new Tree(...);
}
```

Easy use of reusable components

```
void main() {
  final Tree tree = new Tree(...);
  tree.setData(schools);
  tree.addTo('body', 'afterBegin');
}
```

Easy use of reusable components

```
void main() {
  final Tree tree = new Tree(
    [new TreeConfig((School s) => s.schoolName,
                    (School s) => s.grades),
     new TreeConfig((Grade g) => g.schoolGrade,
                    (Grade g) => g.students),
     new TreeConfig((Student s) =>
                      '${s.firstname} ${s.lastname}')]
  );
  tree.setData(schools);
  tree.addTo('body', 'afterBegin');
}
```

Easy use of reusable components

```
class School {
  String schoolName;
  List<Grade> grades;

  School(this.schoolName,
         [this.grades]);
}
```

```
class Grade {
  String schoolGrade;
  List<Student> students;

  Grade(this.schoolGrade,
        [this.students]);
}
```

```
class Student {
  String firstname;
  String lastname;

 Student(this.firstname,
         this.lastname);
}
```

Implementing a reusable components

```
typedef dynamic Accessor(dynamic data);

class TreeConfig {
  Accessor _value;
  Accessor _children;

  TreeConfig(Accessor this._value,
             [Accessor this._children]);

  Accessor get value => _value;
  Accessor get children => _children;
}
```

Implementing a reusable components

```
typedef dynamic Accessor(dynamic data);

class TreeConfig {
  Accessor _value;
  Accessor _children;

  TreeConfig(Accessor this._value,
             [Accessor this._children]);

  Accessor get value => _value;
  Accessor get children => _children;
}
```

Implementing a reusable components

```
typedef dynamic Accessor(dynamic data);

class TreeConfig {
  Accessor _value;
  Accessor _children;

  TreeConfig(Accessor this._value,
             [Accessor this._children]);

  Accessor get value => _value;
  Accessor get children => _children;
}
```

Implementing a reusable components

```
typedef dynamic Accessor(dynamic data);

class TreeConfig {
  Accessor _value;
  Accessor _children;

  TreeConfig(Accessor this._value,
             [Accessor this._children]);

  Accessor get value => _value;
  Accessor get children => _children;
}
```

Implementing a reusable components

```
class Tree {
  List<TreeConfig> treeConfigs;

  String tree;

  Tree(this.treeConfigs);

  String setData(final List data) {
    // Build tree
  }

  void addTo(String selector,
             [String where = 'afterEnd']) {
    query(selector).insertAdjacentHtml(where, this.tree);
  }
}
```

Implementing a reusable components

```
class Tree {
  List<TreeConfig> treeConfigs;

  String tree;

  Tree(this.treeConfigs);

  String setData(final List data) {
    // Build tree
  }

  void addTo(String selector,
             [String where = 'afterEnd']) {
    query(selector).insertAdjacentHtml(where, this.tree);
  }
}
```

Implementing a reusable components

```
class Tree {
  List<TreeConfig> treeConfigs;

  String tree;

  Tree(this.treeConfigs);

  String setData(final List data) {
    // Build tree
  }

  void addTo(String selector,
             [String where = 'afterEnd']) {
    query(selector).insertAdjacentHtml(where, this.tree);
  }
}
```

Implementing a reusable components

```
class Tree {
  List<TreeConfig> treeConfigs;

  String tree;

  Tree(this.treeConfigs);

  String setData(final List data) {
    // Build tree
  }

  void addTo(String selector,
             [String where = 'afterEnd']) {
    query(selector).insertAdjacentHtml(where, this.tree);
  }
}
```

Implementing a reusable components

```
class Tree {
  List<TreeConfig> treeConfigs;

  String tree;

  Tree(this.treeConfigs);

  String setData(final List data) {
    // Build tree
  }

  void addTo(String selector,
             [String where = 'afterEnd']) {
    query(selector).insertAdjacentHtml(where, this.tree);
  }
}
```

Implementing a reusable components

```
String buildOneLevelTree(final List data,
                         final List<TreeConfig> treeNodes,
                         [final int depth = 0]) {
  String tree = '';

  if (data != null && !data.isEmpty) {

  }

  return tree;
}
```

Implementing a reusable components

```
String buildOneLevelTree(final List data,
                         final List<TreeConfig> treeNodes,
                         [final int depth = 0]) {
  String tree = '';

  if (data != null && !data.isEmpty) {
    final TreeConfig treeNode = treeNodes[depth];

    tree += '<ul>';

    for (dynamic element in data) {

    }

    tree += '</ul>';

  }

  return tree;
}
```

Implementing a reusable components

```
String buildOneLevelTree(final List data,
                         final List<TreeConfig> treeNodes,
                         [final int depth = 0]) {
  String tree = '';

  if (data != null && !data.isEmpty) {
    final TreeConfig treeNode = treeNodes[depth];

    tree += '<ul>';

    for (dynamic element in data) {

      tree += '<li>${treeNode.value(element)}';

      tree += '</li>';

    }

    tree += '</ul>';
  }

  return tree;
}
```

Implementing a reusable components

```
String buildOneLevelTree(final List data, final List<TreeConfig> treeNodes,
                         [final int depth = 0]) {
  String tree = '';

  if (data != null && !data.isEmpty) {
    final TreeConfig treeNode = treeNodes[depth];

    tree += '<ul>';

    for (dynamic element in data) {
       tree += '<li>${treeNode.value(element)}';

       if (treeNode.children != null) {
         tree += buildOneLevelTree(treeNode.children(element),
                                   treeNodes, depth + 1);
       }

       tree += '</li>';
    }

    tree += '</ul>';
  }

  return tree;
}
```

Implementing a reusable components

```
class Tree {
  List<TreeConfig> treeConfigs;
  String tree;

  Tree(this.treeConfigs);

  String setData(final List data) {
    this.tree = buildOneLevelTree(data, this.treeConfigs);
    return this.tree;
  }

  String buildOneLevelTree(final List data,
                           final List<TreeConfig> treeNodes,
                           [final int depth = 0]) {
      // Implementation
  }

  void addTo(String selector,
             [String where = 'afterEnd']) {
    query(selector).insertAdjacentHtml(where, this.tree);
  }
}
```

Are we done yet ?

Getting ride of Strings

```
Element buildOneLevelTree(final List data, final List<TreeConfig> treeNodes,
                          [final int depth = 0]) {
  Element tree; // String tree = '';

  if (data != null && !data.isEmpty) {
    final TreeConfig treeNode = treeNodes[depth];

    tree = new UListElement(); // tree += '<ul>';

    for (dynamic element in data) {
      final LIElement li = new LIElement(); // <li>;
      li.text = treeNode.value(element);    // ${treeNode.value(element)}

      if (treeNode.children != null) {
        final UListElement ulChild = //
                      buildOneLevelTree(treeNode.children(element), treeNodes, depth + 1);

        if (ulChild != null) {  //
          li.append(ulChild);   // tree += buildOneLevelTree(...)
        }                       //
      }

      tree.append(li);  // tree += '<li>${treeNode.value(element)}';
    }
  }

  return tree;
}
```

Getting ride of Strings

```
class Tree {
  List<TreeConfig> treeConfigs;
  Element tree; // String tree;

  Tree(this.treeConfigs);

  Element setData(final List data) {
    this.tree = buildOneLevelTree(data, this.treeConfigs);
    return this.tree;
  }

  Element buildOneLevelTree(final List data,
                            final List<TreeConfig> treeNodes,
                            [final int depth = 0]) {
    // Implementation
  }

  void addTo(String selector,
             [String where = 'afterEnd']) {
    query(selector).insertAdjacentElement(where, this.tree);
  }
}
```

web_ui

web_ui

- Based on HTML5 Web Components Spec
- Syntax and uses similar to JSP tags
- Template Engine – Compilation needed
- Reusable components
- CSS encapsulation
- Data-binding
- Complex for real life use-cases
- Doesn’t solve layouting problems

web_ui and Single-Page Webapps

```
<!DOCTYPE html>

<html>
  <head>
    <title>01_web_ui</title>
  </head>

  <body>
    <script type="application/dart" src="01_web_ui.dart">
       </script>
    <script src="packages/browser/dart.js"></script>
  </body>
</html>
```

web_ui - The template

```
<!DOCTYPE html>

<html>
  <body>

  </body>
</html>
```

web_ui - The template

```
<!DOCTYPE html>

<html>
  <body>
    <element>

    </element>
  </body>
</html>
```

web_ui - The template

```
<!DOCTYPE html>

<html>
 <body>
  <element name="x-click-counter">

  </element>
 </body>
</html>
```

web_ui - The template

```
<!DOCTYPE html>

<html>
 <body>
  <element name="x-click-counter" constructor="CounterComponent">

  </element>
 </body>
</html>
```

web_ui - The template

```
<!DOCTYPE html>

<html>
 <body>
  <element name="x-click-counter" constructor="CounterComponent" extends="div">

  </element>
 </body>
</html>
```

web_ui - The template

```
<!DOCTYPE html>

<html>
 <body>
  <element name="x-click-counter" constructor="CounterComponent" extends="div">
   <template>

   </template>
  </element>
 </body>
</html>
```

web_ui - The template

```
<!DOCTYPE html>

<html>
 <body>
  <element name="x-click-counter" constructor="CounterComponent" extends="div">
   <template>
    <div>
     <button>Click me</button><br />
     <span>(click count: {{count}})</span>
    </div>
   </template>
  </element>
 </body>
</html>
```

web_ui - The template

```
<!DOCTYPE html>

<html>
 <body>
  <element name="x-click-counter" constructor="CounterComponent" extends="div">
   <template>
    <div>
     <button>Click me</button><br />
     <span>(click count: {{count}})</span>
    </div>
   </template>
  </element>
 </body>
</html>
```

web_ui - The template

```
<!DOCTYPE html>

<html>
 <body>
  <element name="x-click-counter" constructor="CounterComponent" extends="div">
   <template>
    <div>
     <button>Click me</button><br />
     <span>(click count: {{count}})</span>
    </div>
   </template>
   <script type="application/dart" src="xclickcounter.dart"></script>
  </element>
 </body>
</html>
```

web_ui – Extending WebComponent

```
class CounterComponent {


}
```

web_ui – Extending WebComponent

```
class CounterComponent extends WebComponent {


}
```

web_ui – Extending WebComponent

```
class CounterComponent extends WebComponent {
  @observable
  int count = 0;

}
```

web_ui – Extending WebComponent

```
class CounterComponent extends WebComponent {
  @observable
  int count = 0;

  void increment(Event event) {
    count++;
  }
}
```

web_ui – Extending WebComponent

```
class CounterComponent extends WebComponent {
  @observable
  int count = 0;

  void increment(Event event) {
    count++;
  }

  void inserted() {
    this.query('button').onClick.listen(increment);
  }
}
```

web_ui and Single-Page Webapps

```
<!DOCTYPE html>

<html>
  <head>
    <title>01_web_ui</title>
    <link rel="components" href="xclickcounter.html">
  </head>

  <body>
    <script type="application/dart" src="01_web_ui.dart">
       </script>
    <script src="packages/browser/dart.js"></script>
  </body>
</html>
```

web_ui – The application

```
void main() {


}
```

web_ui – The application

```
void main() {
  var element = new Element.html(
    '<x-click-counter id="click_counter"></x-click-counter>'
  );
}
```

web_ui – The application

```
void main() {
  var element = new Element.html(
    '<x-click-counter id="click_counter"></x-click-counter>'
  );
  var counter = new CounterComponent()
    ..host = element
    ..count = 25;
}
```

web_ui – The application

```
void main() {
  var element = new Element.html(
    '<x-click-counter id="click_counter"></x-click-counter>'
  );
  var counter = new CounterComponent()
    ..host = element
    ..count = 25;

  var lifecycleCaller = new ComponentItem(counter)
    ..create();
  query('body').append(counter.host);
  lifecycleCaller.insert();
}
```

web_ui – The application

```
void main() {
  var element = new Element.html(
    '<x-click-counter id="click_counter"></x-click-counter>'
  );
  var counter = new CounterComponent()
   ..host = element
   ..count = 25;

  var lifecycleCaller = new ComponentItem(counter)..create();
  query('body').append(counter.host);
  lifecycleCaller.insert();

  var button = new ButtonElement()
    ..text = 'Update'
    ..onClick.listen((e) {
      counter.count = 100;

      watchers.dispatch();
    });
  query('body').append(button);
}
```

A word about Layouts

Menu and content dynamic

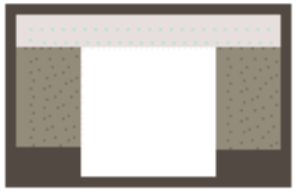
Menu fixed, Content dynamic

Menu and content dynamic

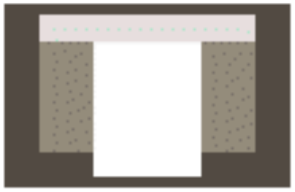
3 columns, all dynamic

4 columns, all dynamic

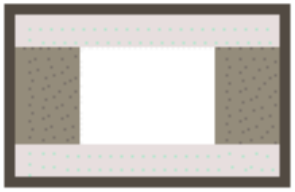
Menu floating



Menu fixed, content & header dynamic



3 columns fixed centered

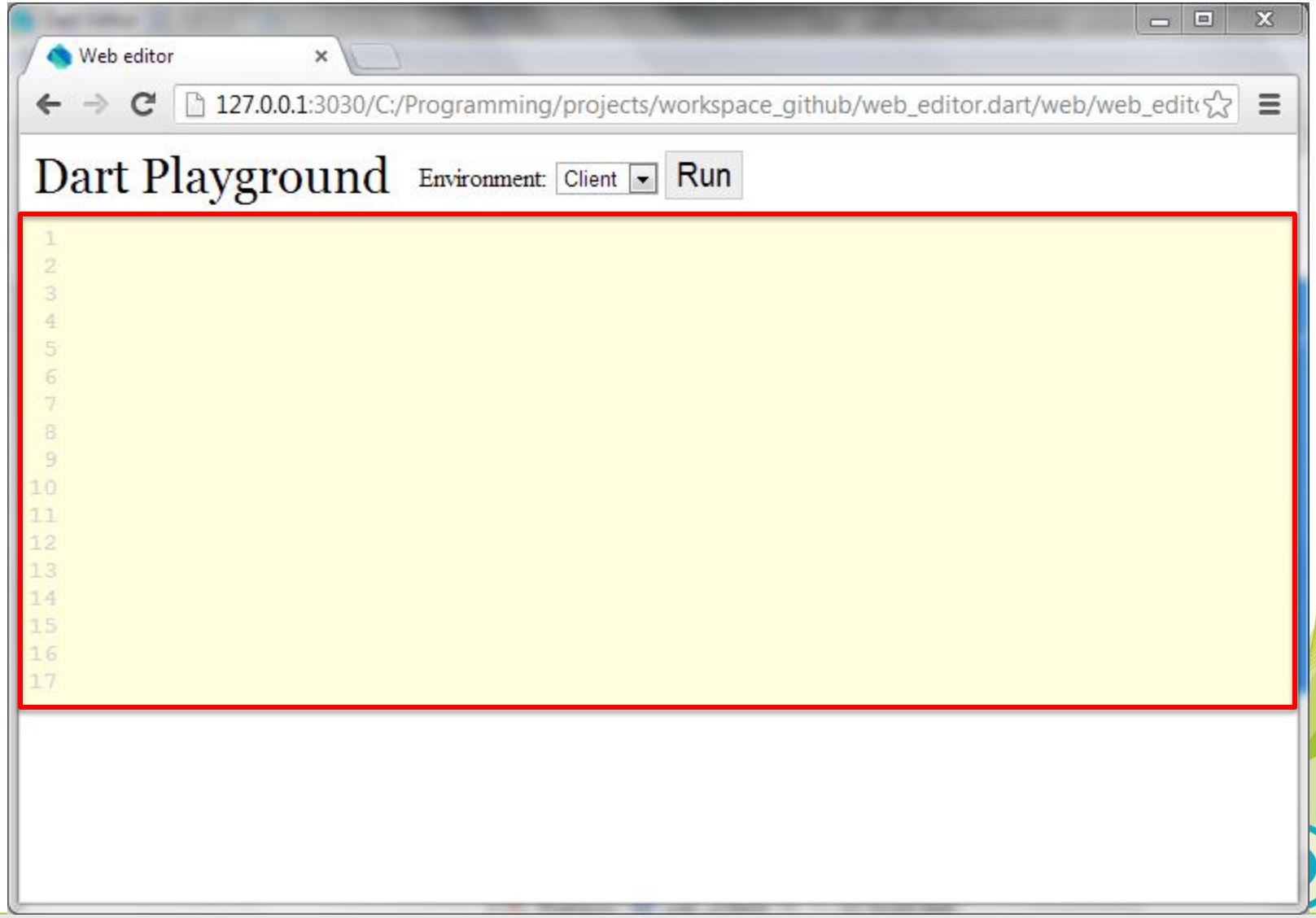


dynamic with header and footer

A word about Layouts

A word about Layouts

A word about Layouts



A word about Layouts

A word about Layouts

```
builder()
  ..div({'id' : 'banner'})
    ..div({'id' : 'head'}, 'Dart Playground')
    ..div({'id' : 'controls'})
      ..span(null, 'Environment: ')
      ..addElement(listboxEnv)
      ..addElement(runButton)
    ..end() // controls
  ..end() // banner
  ..div({'id':'wrap'})
    ..addElement(e(linedTextarea.element))
  ..end() // wraps
  ..addElement(output);
```

A word about Layouts

```
builder()
  ..div({'id' : 'banner'})
    ..div({'id' : 'head'}, 'Dart Playground')
    ..div({'id' : 'controls'})
      ..span(null, 'Environment: ')
      ..addElement(listboxEnv)
      ..addElement(runButton)
    ..end() // controls
  ..end() // banner
  ..div({'id':'wrap'})
    ..addElement(e(linedTextarea.element))
  ..end() // wraps
  ..addElement(output);
```

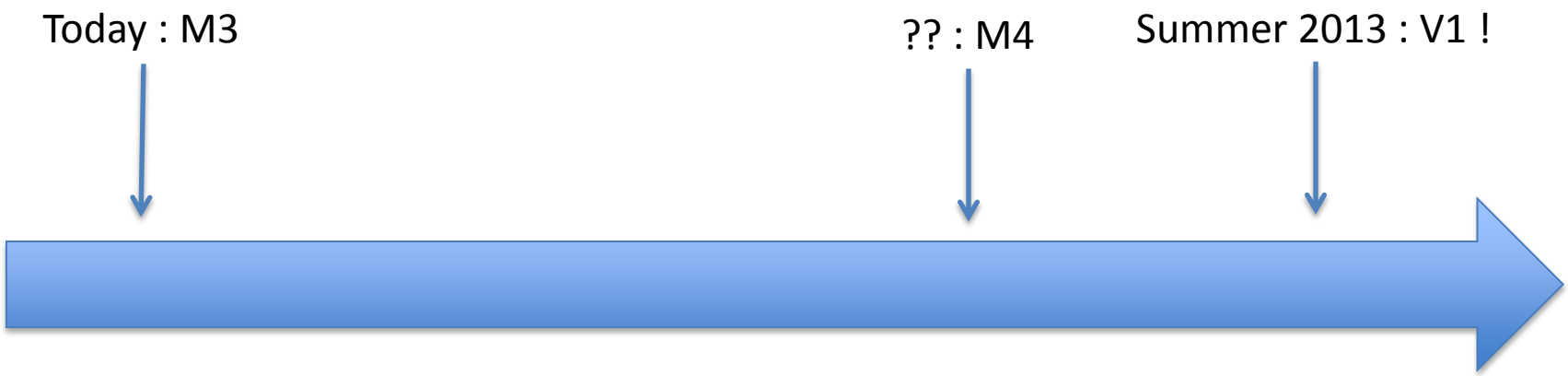
A word about Layouts

```
builder()
  ..div({'id' : 'banner'})
    ..div({'id' : 'head'}, 'Dart Playground')
    ..div({'id' : 'controls'})
      ..span(null, 'Environment: ')
      ..addElement(listboxEnv)
      ..addElement(runButton)
    ..end() // controls
  ..end() // banner
  ..div({'id':'wrap'})
    ..addElement(e(linedTextarea.element))
  ..end() // wraps
  ..addElement(output);
```

A word about Layouts

```
builder()
  ..div({'id' : 'banner'})
    ..div({'id' : 'head'}, 'Dart Playground')
    ..div({'id' : 'controls'})
      ..span(null, 'Environment: ')
      ..addElement(listboxEnv)
      ..addElement(runButton)
    ..end() // controls
  ..end() // banner
  ..div({'id':'wrap'})
    ..addElement(e(linedTextarea.element))
  ..end() // wraps
  ..addElement(output);
```

Roadmap



The Future of Dart ?

Want to know more ?

DartLangFR

- Mailing-list : dartlangfr (https://groups.google.com/forum/?fromgroups=&hl=en#!forum/dartlangfr)
- Google+ : DartlangFR (https://plus.google.com/u/0/communities/104813951711720144450)
- Twitter : @dartlang_fr
- Blog : dartlangfr.net

DartLang

- Official website: www.dartlang.org
- Mailing-list : dartlang (https://groups.google.com/a/dartlang.org/forum/?fromgroups&hl=en#!forum/misc)
- Google+ : Dart (https://plus.google.com/+dartlang)
- Google+ : Dartisans (https://plus.google.com/communities/114566943291919232850)
- Twitter : @dart_lang
- Blog : blog.dartwatch.com
- Newsletter : Dart weekly

Github

- Paris JUG
  - https://github.com/yohanbeschi/parisjug_20130409.dart
- DevoxxFR 2013
  - https://github.com/yohanbeschi/devoxxfr_20130327.dart
- Widgets
  - https://github.com/yohanbeschi/pwt_proto.dart
- Web Editor for Dart
  - https://github.com/yohanbeschi/web_editor.dart

Thank You

Questions ?