brief contents

PART 1 THE BASICS 1

1 ▪ What is a single-page application? 3
2 ▪ The role of MV* frameworks 22
3 ▪ Modular JavaScript 52

PART 2 CORE CONCEPTS 83

4 ▪ Navigating the single page 85
5 ▪ View composition and layout 106
6 ▪ Inter-module interaction 129
7 ▪ Communicating with the server 156
8 ▪ Unit testing 186
9 ▪ Client-side task automation 209

appendix A Employee directory example walk-through 229
appendix B Review of the XMLHttpRequest API 259
appendix C Chapter 7 server-side setup and summary 266
appendix D Installing Node.js and Gulp.js 277

contents

foreword *xv*
preface *xvii*
acknowledgments *xviii*
about this book *xix*
about the cover illustration *xxiii*

PART 1 THE BASICS 1

1 *What is a single-page application?* *3*

1.1 SPA in a nutshell 4

No browser refreshes *7* ▪ *Presentation logic in the client* *7*
Server transactions *7*

1.2 A closer look 8

An SPA starts with a shell *8* ▪ *From traditional pages to views* *9* ▪ *The birth of a view* *10* ▪ *View swapping for zero reload navigation* *11* ▪ *Fluidity through dynamic updates* *12*

1.3 Benefits of SPAs over traditional web applications 12

1.4 Rethinking what you already know 14

1.5 Ingredients of a well-designed SPA 15

Organizing your project *15* ▪ *Creating a maintainable, loosely coupled UI* *17* ▪ *Using JavaScript modules* *18* ▪ *Performing SPA navigation* *19* ▪ *Creating view composition and layout* *19* ▪ *Enabling module communication* *20*

Communicating with the server 20 ▪ Performing unit testing 20 ▪ Using client-side automation 20

1.6 Summary 20

2 ***The role of MV* frameworks 22***

2.1 What is MV*? 24

Traditional UI design patterns 25 ▪ MV and the browser environment 27*

2.2 Common MV* concepts 29

Meet the frameworks 30 ▪ Meet our MV project 30 Models 32 ▪ Bindings 36 ▪ Templates 40 ▪ Views 44*

2.3 Why use an MV* framework? 44

Separation of concerns 45 ▪ Routine tasks simplified 46 Productivity gains 47 ▪ Standardization 47 Scalability 48

2.4 Choosing a framework 48

2.5 Chapter challenge 50

2.6 Summary 51

3 ***Modular JavaScript 52***

3.1 What is a module? 53

Module pattern concepts 53 ▪ The module's structure 54 The revealing module pattern 55

3.2 Why modular programming? 56

Avoiding name collisions 56 ▪ Protecting your code's integrity 65 ▪ Hiding complexity 67 ▪ Reducing the impact of code changes 68 ▪ Organizing your code 68 Understanding disadvantages of the module pattern 69

3.3 The module pattern dissected 69

Achieving privacy 69 ▪ Creating the public API 70 Allowing for global imports 73 ▪ Creating the module's namespace 73

3.4 Module loading and dependency management 74

Script loaders 74 ▪ Asynchronous Module Definition 75 Using AMD modules with RequireJS 77

3.5 Chapter challenge 81

3.6 Summary 81

PART 2 CORE CONCEPTS 83

4 *Navigating the single page 85*

4.1 What is a client-side router? 86

Traditional navigation 86 ▪ SPA navigation 86

4.2 Routes and their configuration 88

Route syntax 90 ▪ Route entries 90 ▪ Route parameters 91 Default routes 93

4.3 How do client-side routers work? 93

The fragment identifier method 94 ▪ The HTML5 History API method 95 ▪ Changes for the HTML5 History API method 97

4.4 Putting it all together: implementing routing in SPAs 98

The faculty list (default route) 99 ▪ The main contact route 101 ▪ Faculty office hours (parameterized route) 102

4.5 Chapter challenge 104

4.6 Summary 105

5 *View composition and layout 106*

5.1 Introducing our project 107

5.2 Reviewing layout design concepts 108

Views 108 ▪ Regions 109 ▪ View composition 110 Nested views 111 ▪ Routes 112

5.3 Considering alternatives for advanced composition and layout 113

Pros 113 ▪ Cons 114

5.4 Designing the application 114

Designing the base layout 115 ▪ Designing the default content 117 ▪ Using a view manager for complex designs 122 Creating nested views with their own states 125

5.5 Chapter challenge 127

5.6 Summary 128

6 *Inter-module interaction 129*

6.1 Review of module concepts 131

Modules encapsulate code 131 ▪ APIs provide controlled access to internal functionality 133 ▪ SRP means designing with a single purpose 134 ▪ Code reuse helps your project scale 135

6.2 Inter-module interaction methods 136

Inter-module interaction through dependencies 136 Dependency method pros and cons 138 ▪ Inter-module interaction through publish/subscribe 138 ▪ Pub/sub pros and cons 141

6.3 Project details 142

Searching 144 ▪ Displaying product information 150

6.4 Chapter challenge 155

6.5 Summary 155

7 ***Communicating with the server 156***

7.1 Understanding the project requirements 158

7.2 Exploring the communication process 158

Choosing a data type 158 ▪ Using a supported HTTP request method 159 ▪ Converting the data 160

7.3 Using MV* frameworks 162

Generating requests 162 ▪ Processing results with callbacks 165 ▪ Processing results with promises 166 Promise error handling 171

7.4 Consuming RESTful web services 172

What is REST? 172 ▪ REST principles 172 ▪ How MV frameworks help us be RESTful 174*

7.5 Project details 174

Configuring REST calls 174 ▪ Adding product items to the cart 177 ▪ Viewing the cart 179 ▪ Updating the cart 181 Removing products from the cart 183

7.6 Chapter challenge 184

7.7 Summary 184

8 ***Unit testing 186***

8.1 Understanding the project 187

8.2 What is a unit test? 187

Benefits of unit testing 188 ▪ Creating better unit tests 189

8.3 Traditional unit testing 192

Getting started with QUnit 193 ▪ Creating your first unit tests 196 ▪ Testing code built with MV objects 200 Testing changes to the DOM 205 ▪ Adding other testing frameworks to the mix 206*

8.4 Chapter challenge 208
8.5 Summary 208

9 *Client-side task automation* 209
9.1 Common uses for task runners 210
Live browser reloads 210 ▪ *Automatic JavaScript and CSS preprocessing* 211 ▪ *Automatic code analysis from linters* 211 *Continuous unit testing* 211 ▪ *File concatenation* 212 *Code minification* 212 ▪ *Continuous integration* 212
9.2 Choosing a task runner 212
9.3 Our project 213
Introducing Gulp.js 214 ▪ *Creating your first task* 215 *Creating a code analysis task* 216 ▪ *Creating a browser-reload task* 218 ▪ *Automating unit testing* 220 ▪ *Creating a build process* 222
9.4 Chapter challenge 227
9.5 Summary 227

appendix A *Employee directory example walk-through* 229
appendix B *Review of the XMLHttpRequest API* 259
appendix C *Chapter 7 server-side setup and summary* 266
appendix D *Installing Node.js and Gulp.js* 277

index 281