Building Web Applications with React

Chapter 4: React and Rest

Chapter Objectives

In this chapter, we will:

- Introduce REST
- Make `GET` requests to retrieve data
- `POST` data to the server for updates

Chapter Concepts

Introducing REST

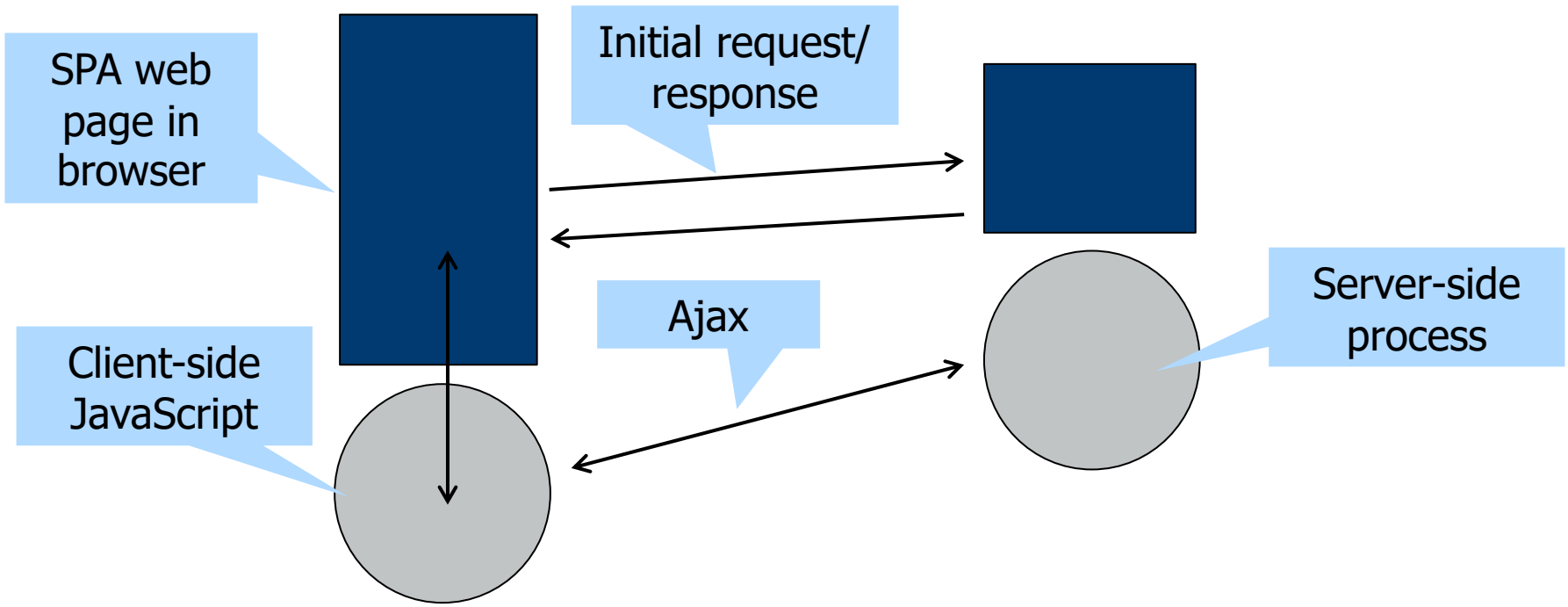
Retrieving Data

Modifying Data

Chapter Summary

SPAs and HTTP Requests

- Recall that in Single Page Applications:
  - Initial page is loaded via a normal web request
  - Subsequent interaction via Ajax
    - HTTP requests made on a background thread



What Is Ajax?

- Ajax is a term loosely based on:
  - Asynchronous
    - Happens on a background thread
  - JavaScript
    - The client-side scripting language making the request
  - Xml
    - The `XmlHttpRequest` object
      - Responsible for making the requests
- Originally, XML was the primary payload for Ajax requests
  - Modern browsers use JavaScript Object Notation (JSON)
    - Simpler and less verbose
    - Very similar to JavaScript object literals
    - Easily converted into JavaScript objects
      - Via `JSON.parse()` method of the browser

JSON Example

```
[ {
    "bookId": 67,
    "title": "Tom's Midnight Garden",
    "author": "Philippa Pearce"
}, {
    "bookId": 66,
    "title": "The Borribles Go For Broke",
    "author": "Michael de Larrabeiti"
}, {
    "bookId": 3,
    "title": "The Ennead",
    "author": "Jan Mark"
}, {
    "bookId": 1,
    "title": "The Lord Of The Rings",
    "author": "J R Tolkien"
} ]
```

As with JavaScript, `[]` = array, `{}` = object

Everything is a string, except numbers

JSON property names are double-quoted strings

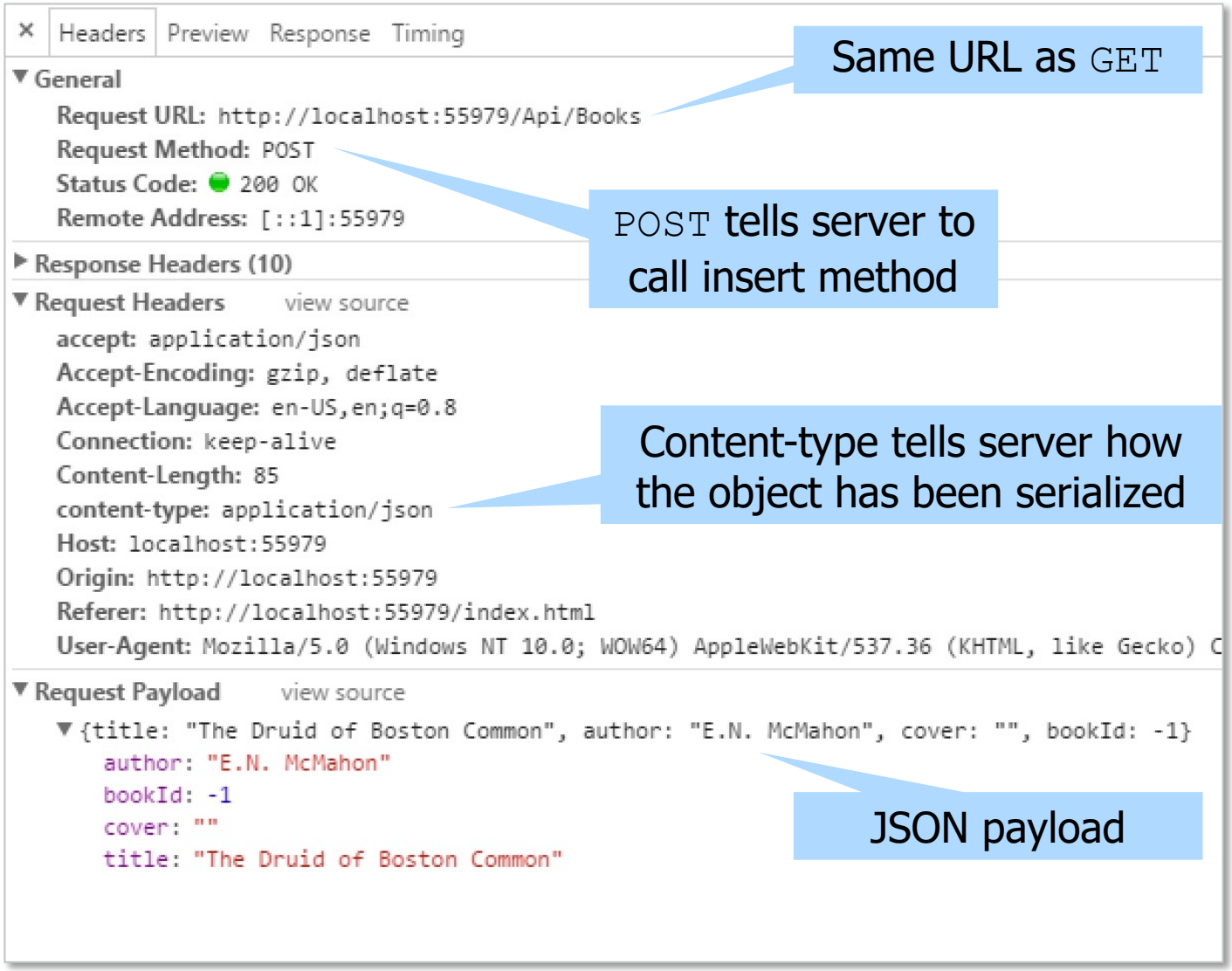
What Is REST?

- REST stands for Representational State Transfer
  - An alternative to 'traditional' web services
  - Much simpler, with no complicated XML vocabulary
- Uses HTTP verbs
  - `GET`, `POST`, `PUT`, `DELETE`
- Can return any payload
  - XML
  - HTML
  - Plain text
  - JSON
- Typically, React applications will make RESTful Ajax calls
  - And receive JSON data from the server

Anatomy of a REST Call

- REST maps HTTP verbs to server-side CRUD operations
  - Create = `HTTP POST`
  - Retrieve = `HTTP GET`
  - Update = `HTTP PUT (MERGE/PATCH)`
  - Delete = `HTTP DELETE`
- Actions on the server are accessed via simple hierarchical URLs
- `GET` requests
  - `/Api/Books`
    - Retrieves data for all books
  - `/Api/Reviews/67`
    - Retrieves reviews for bookId 67
- `POST` requests
  - /`Api/Books`
    - Insert new book
    - URL is same as for `GET`
      - Use of `POST` method tells the server what action to take

REST `HTTP POST` Illustrated



REST Technologies

- There are two parts to any REST communication
  - Client-side and server-side
- Any server-side technology can be used to create REST services
  - Java
  - Python
  - Node
  - .NET…
- Unlike some SPAs, React does not provide a REST client
  - Developers could use a third-party library
    - Or simply take advantage of the browser `fetch()` method
- This course will use `fetch`

Chapter Concepts

Introducing REST

Retrieving Data

Modifying Data

Chapter Summary

Making a `GET` Request

1. Pass the REST URL to the `fetch()` method

```
fetch("/Api/Reviews/" + bookId)
```

2. Chain a `then()` method and retrieve JSON from the response

```
.then(function (response) {
        return response.json();
})
```

3. Retrieve the parsed JSON from the promise returned by `response.json()`

```
reviewApi.fetchAllBooks().then(function (data) {
   // do something with data here. Data is the parsed json
});
```

- Implication: `response.json` might not contain the requested data
  - Can check `response.ok` before accessing JSON data

Coding the REST Request

1. Inside the `bookAPI` file

```
export const fetchAllBooks = () => {
    return fetch('/api/Books').then((response) => {
       return response.json();
    });
}
```

`fetch()` makes the HTTP request

2. Inside a class component

```
getBooks() {
    return api.fetchAllBooks().then((response) => {
        this.setState({ books: response });
    });
}
```

The return from `fetch()` is a promise, so `setState()` is called inside a `then()`

Hooks and REST Requests

3. Inside a functional component

```
const [reviews, setReviews] = useState([]);


useEffect(() => {
     api.fetchReviews(bookId).then((response) => {
        setReviews(response);
    });
  },
  [bookId]);
```

`useEffect()` hook to control timing of REST

`useState()` hook for state

Calling the `useState()` hook

Optional second array argument ensures effect will only run when argument changes. Without this, it would run on every render, causing an infinite loop.

Exercise 4.1: Retrieving Data with REST

- In this exercise, you will retrieve book data from a RESTful web service and display it inside your React application
- Please refer to the Exercise Manual

Chapter Concepts

Introducing REST

Retrieving Data

Modifying Data

Chapter Summary

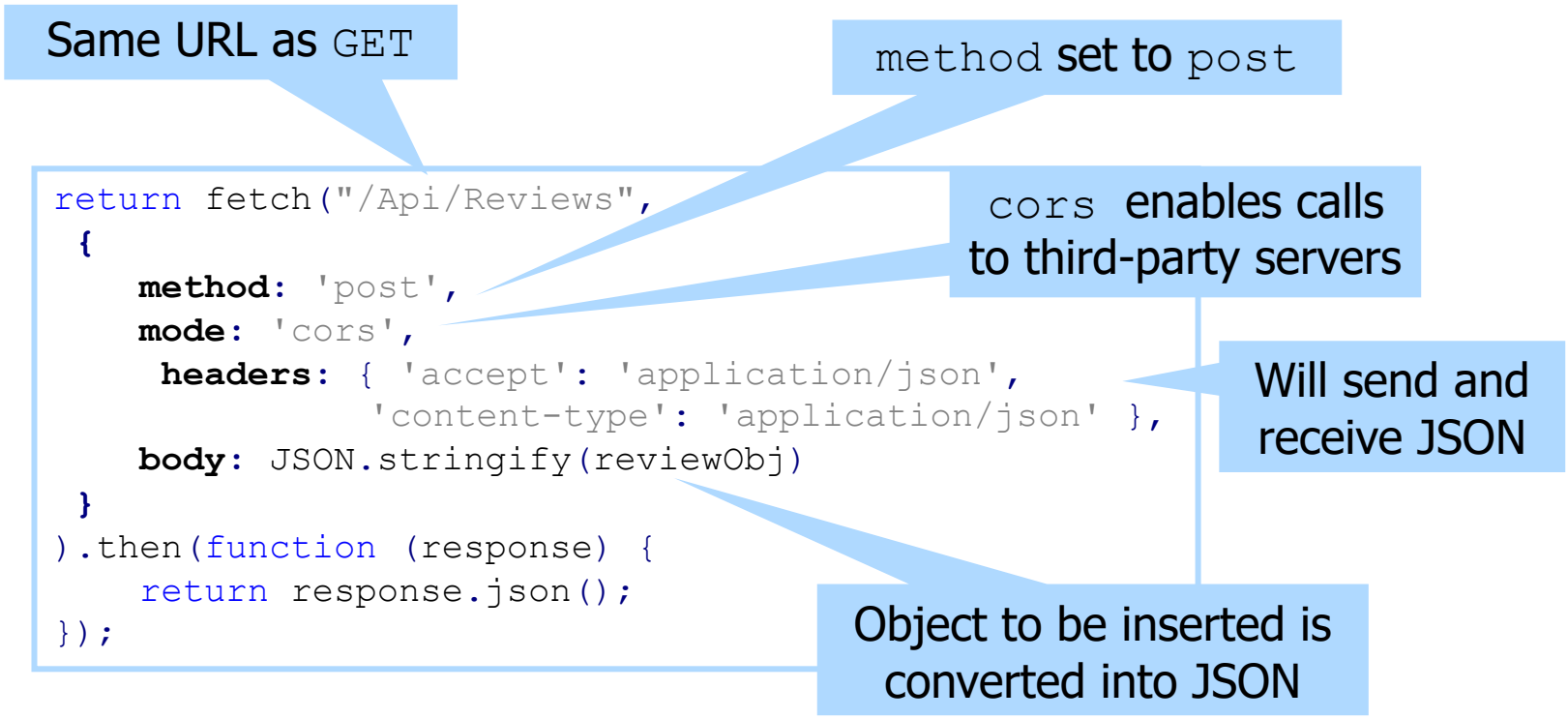
Modifying Data with REST

- REST requests for insert, update, and deletion:
  - Use different verbs
    - `POST` for addition
    - `PUT MERGE` or `PATCH` for updates
  - Have a request body
    - The data to be sent to the server
      - Typically a JSON object
  - Specify appropriate HTTP headers
    - Data-type of request body
    - Data-type of expected response

The Options Object

- The `fetch()` method accepts an options object as the second argument
  - Not required for `GET` requests that use default settings
- Options object is used to set HTTP properties
  - HTTP headers
  - HTTP method
  - Request body
- For an insert, needs the following settings:
  - Method: `post`
  - Headers:
    - Accept: `json`
    - Content-type: `json`
  - Body: `JSON.stringified` object

The Options Object—Code



Updates on the Client

- Two choices for managing updates in SPAs
  - Mixed client and server updating
  - Server-only
- Mixed approach
  - Update server and store independently
    - Client: `push()` new data into local arrays as it is created
    - Server: send data using REST
      - No immediate refresh of client data from server
  - Advantage:
    - Performance: no need to wait for HTTP round trip
- Server-only approach
  - Replace local data with fresh data from server after updating
  - Advantage:
    - More reliable: will include other users' updates

Exercise 4.2: Inserting Data with REST

- In this exercise, you will add a new book by making a RESTful call to a web service
- Please refer to the Exercise Manual

Chapter Concepts

Introducing REST

Retrieving Data

Modifying Data

Chapter Summary

Chapter Summary

In this chapter, we have:

- Introduced REST
- Made `GET` requests to retrieve data
- `POST`ed data to the server for updates