Server Side Web Frameworks

Functionality provided by server-side web frameworks may include the following:

- Tying dynamic functionality to specific URLs.
- Support for the basic HTTP request-response lifecycle.
- Some kind of application context.
- Integration with a templating framework.
- Support for a persistence layer.

Contexts

Persistence Persistence layer, used for storing long-term information which may be shared by multiple application servers.

Application Global to entire application. Good place to remember information like database connections and read-only info like configuration.

Session Information associated with a user (often tied to user via a cookie). Problematic if only in memory; avoid except for non-critical information.

Request Context associated with an individual request-response cycle. Can often store this information in some kind of request object.

Client Information stored on client. Browsers provide cookies, local storage and session storage.

Express.js Overview

- Built on top of nodejs HTTP facilities.
- Allows dispatching HTTP requests based on a modular routing framework.
- Allows inserting *middleware* into the applications request-response cycle.
- Allows plugging in multiple **template engines** to generate dynamic content.

Routing

- Routing allows specifying that a **handler** function (single or multiple) run when the URL for a web request matches a particular route.
- Matching route can be a fixed string or a pattern or a regular expression.
- Handler function takes a `req` and `res` argument to access the incoming `Request` and outgoing `Response`. Optionally, it can also take a `next` argument; calling `next()` allows chaining to the next set of handlers for a route which matches the request URL.

Middleware

- Like many other JavaScript frameworks, express.js is highly modular. It provides a friendlier layer over node's `http` module and relegates most functionality to library functions hooked into as *middleware*.
- Allows inserting arbitrary code into request-response cycle. Can make changes to the request and response objects, end the cycle, call the next middleware (using `next()`). Three basic kinds:

Application	Used for all requests which satisfy mount-point.
Router	Just like application, but a router can be mounted at a specific mountpoint. Used for factoring behavior by URL prefixes.
Error	Used for errors. Handler **must** have 4 arguments with the first argument being the error object.

Users Web Service

Provide a web service for maintaining users. Provides following methods.

`GET /users` Return information for all current users.

`GET /users/`*userId* Return information for existing user *userId*.

`POST /users` Create new user specified using JSON request body.

`PUT /users/`*userId* Replace information for existing user *userId* by JSON request body.

`PATCH /users/`*userId* Update information for existing user *userId* by JSON request body.

`DELETE /users/`*userId* Delete information for existing user *userId*.

Underlying User-Store

Modification of earlier user-store. Changes include:

- Deal with only a single user rather than a list of users.
- No default id (defaulting to configured git email).
- Read method changes to allow any filter (including none).
- Services provides are `create`, `delete`, `load`, `read`, `replace` and `update`.
- All handlers are wrapped for server errors.

Code

- Top-level index.js. Start web server using:

```
$ ./index.js 1234 simpsons.json
```

  to run on port 1234 loading data from `simpsons.json`.
- Web services user-ws.js.
- Underlying store user-store.js.
- Can access using REST clients like Restlet.