

# **Comp 324/424 - Client-side Web Design**

---

Spring Semester 2017 - Week 11

Dr Nick Hayward

# Contents

---

- Server-side considerations
- Data storage
  - *Redis*
  - *MongoDB - Part I*
- Data visualisation
  - *intro*
  - *types*

# Server-side considerations - data storage

---

## *intro*

- tested Node.js, created a server for hosting our files and routes with ExpressJS
  - *read JSON from the server*
  - *updated our JSON on the server-side*
- works well as long as we do not need to restart, repair, update etc our server
- data lost with restart etc...
- need to consider a persistent data storage
  - *independent from the application*
- NoSQL options such as Redis and MongoDB
- integration with Node.js

# Server-side considerations - data storage

---

## SQL or NoSQL

- common database usage and storage
  - *often thought solely in terms of SQL, or structured query language*
- SQL used to query data in a relational format
- relational databases, for example MySQL or PostgreSQL, store their data in tables
  - *provides a semblance of structure through rows and cells*
  - *easily cross-reference, or relate, rows across tables*
- a relational structure to map authors to books, players to teams...
  - *thereby dramatically reducing redundancy, required storage space...*
- improvement in storage capacities, access...
  - *led to shift in thinking, and database design in general*
- started to see introduction of non-relational databases
  - *often referred to simply as **NoSQL***
- with NoSQL DBs
  - *redundant data may be stored*
  - *such designs often provide increased ease of use for developers*
- some NoSQL examples for specific use cases
  - *eg: fast reading of data more efficient than writing*
  - *specialised DB designs*

# Server-side considerations - data storage

---

## Redis - intro

- Redis provides an excellent example of NoSQL based data storage
- designed for fast access to frequently requested data
- improvement in performance often due to a reduction in perceived reliability
  - *due to in-memory storage instead of writing to a disk*
- able to flush data to disk
  - *performs this task at given points during uptime*
  - *for majority of cases considered an in-memory data store*
- stores this data in a **key-value** format
  - *similar in nature to standard object properties in JavaScript*
- Redis often a natural extension of conventional data structures
- Redis is a good option for quick access to data
  - *optionally caching temporary data for frequent access*

# Server-side considerations - data storage

---

## Redis - installation

- On OS X, use the Homebrew package manager to install Redis

```
brew install redis
```

- Windows port maintained by the Microsoft Open Tech Group - Redis
  - or use Windows package manager - <https://chocolatey.org/>
- for Linux - download, extract, and compile Redis

```
$ wget http://download.redis.io/releases/redis-3.0.5.tar.gz
$ tar xzf redis-3.0.5.tar.gz
$ cd redis-3.0.5
$ make
```

# Server-side considerations - data storage

---

## Redis - server and CLI

- start the Redis server with the following command,

```
redis-server
```

- interact with our new server directly using the CLI tool,

```
redis-cli
```

- store some data in Redis using the set command
  - create a new key for *notes*, and then set its value to *0*
  - if value is set, Redis returns *OK*

```
set notes 0
```

- retrieve a value using the get command
  - returns our set value of *0*

```
get notes
```

# Image - Client-side and server-side computing

---

```
Drs-MacBook-Air-2:~ ancientlives$ redis-cli
127.0.0.1:6379> set notes 0
OK
127.0.0.1:6379> get notes
"0"
127.0.0.1:6379> █
```

Redis CLI - set and get



# Server-side considerations - data storage

---

## Redis - server and CLI

- also manipulate existing values for a given key
  - eg: *increment and decrement a value, or simply delete a key*
- increment key notes value by 1

```
incr notes
```

- decrement key notes value by 1

```
decr notes
```

- we can then increment or decrement by a specified amount

```
// increment by 10
incrby notes 10
// decrement by 5
decrby notes 5
```

- delete our key

```
// single key deletion
del notes
// multiple keys deletion
del notes notes2 notes3
```

# Image - Client-side and server-side computing

---

```
Drs-MacBook-Air-2:~ ancientlives$ redis-cli
127.0.0.1:6379> set notes 0
OK
127.0.0.1:6379> get notes
"0"
127.0.0.1:6379> incr notes
(integer) 1
127.0.0.1:6379> incr notes
(integer) 2
127.0.0.1:6379> get notes
"2"
127.0.0.1:6379> decr notes
(integer) 1
127.0.0.1:6379> get notes
"1"
127.0.0.1:6379> incrby notes 10
(integer) 11
127.0.0.1:6379> get notes
"11"
127.0.0.1:6379> decrby notes 5
(integer) 6
127.0.0.1:6379> get notes
"6"
```

Redis CLI - increment and decrement

# Server-side considerations - data storage

---

## *Redis and Node.js setup*

- test Redis with our Node.js app
- new test app called 424-node-redis1

```
| - 424-node-redis1
|   | - app
|   |   | - assets
|   | - node_modules
|   - package.json
| - server.js
```

- create new file, `package.json` to track project
  - eg: *dependencies, name, description, version...*

# Server-side considerations - data storage

---

## *Redis and Node.js - package.json*

```
{
  "name": "424-node-redis1",
  "version": "1.0.0",
  "description": "test app for node and redis",
  "main": "server.js",
  "dependencies": {
    "body-parser": "^1.14.1",
    "express": "^4.13.3",
    "redis": "^2.3.0"
  },
  "author": "ancientlives",
  "license": "ISC"
}
```

- we can write the `package.json` file ourselves or use the interactive option

```
npm init
```

- then add extra dependencies, eg: Redis, using

```
npm install redis --save
```

- use `package.json` to help with app management and abstraction...

# Server-side considerations - data storage

---

## *Redis and Node.js - set notes value*

- add Redis to our earlier test app
- import and use Redis in the `server.js` file

```
...  
var express = require("express"),  
    http = require("http"),  
    bodyParser = require("body-parser"),  
    jsonApp = express(),  
    redis = require("redis");  
...
```

- create client to connect to Redis from Node.js

```
//create client to connect to Redis  
redisConnect = redis.createClient();
```

- then use Redis, for example, to store access total for notes on server

```
redisConnect.incr("notes");
```

- check Redis command line for change in notes value

```
get notes
```

# Server-side considerations - data storage

---

## Redis and Node.js - get notes value

- now set the counter value for our notes
  - *add our counter to the application to record access count for notes*
- use the get command with Redis to retrieve the incremented values for the notes key

```
redisConnect.get("notes", function(error, notesCounter) {  
    //set counter to int of value in Redis or start at 0  
    notesTotal.notes = parseInt(notesCounter,10) || 0;  
});
```

- get accepts two parameters - error and return value
- Redis stores values and strings
  - *convert string to integer using `parseInt()`*
  - *two parameters - return value and `base-10` value of the specified number*
- value is now being stored in a global variable `notesTotal`
  - *declared in `server.js`*

```
var express = require("express"),  
    http = require("http"),  
    bodyParser = require("body-parser"),  
    jsonApp = express(),  
    redis = require("redis"),  
    notesTotal = {};
```

# Server-side considerations - data storage

---

## Redis and Node.js - get notes value

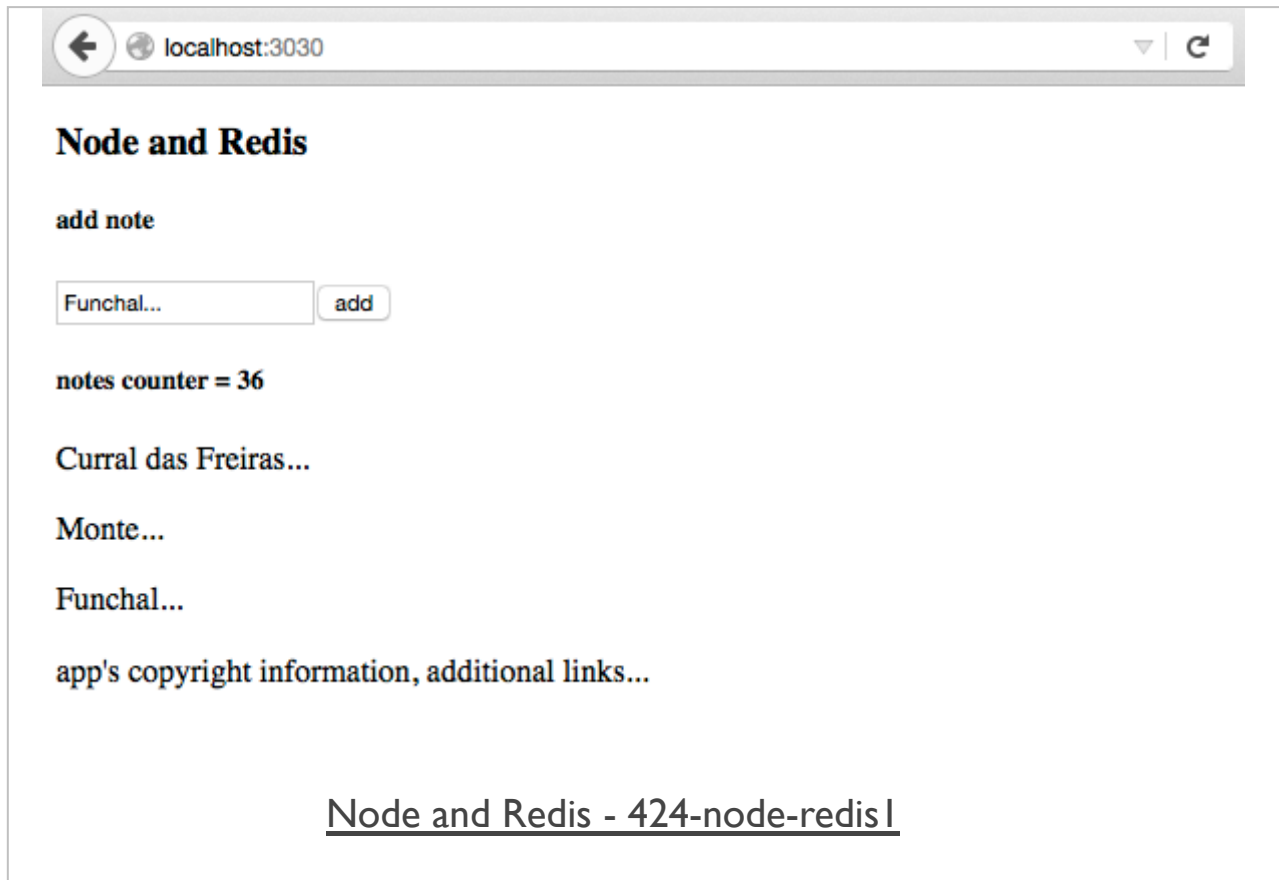
- store notes counter value in Redis
- create new route in `server.js`
  - *monitor the returned JSON for the counter*

```
//json get route
jsonApp.get("/notesTotal.json", function(req, res) {
  res.json(notesTotal);
});
```

- start using it with our application
  - *load by default, within event handler...*
- render to DOM
- store as a internal log record
- link to create note event handler...
- DEMO - 424-node-redis I

# Image - Client-side and server-side computing

---





# Server-side considerations - data storage

---

## *MongoDB - intro*

- MongoDB is another example of a NoSQL based data store
  - *a database that enables us to store our data on disk*
- unlike MySQL, for example, it is not in a relational format
- MongoDB is best characterised as a **document-oriented** database
- conceptually may be considered as storing objects in collections
- stores its data using the BSON format
  - *consider similar to JSON*
  - *use JavaScript for working with MongoDB*

# Server-side considerations - data storage

---

## *MongoDB - document oriented*

- SQL database, data is stored in tables and rows
- MongoDB, by contrast, uses **collections** and **documents**
- comparison often made between a collection and a table
- **NB:** a document is quite different from a table
- a document can contain a lot more data than a table
- a noted concern with this document approach is duplication of data
- one of the trade-offs between NoSQL (MongoDB) and SQL
- SQL - goal of data structuring is to normalise as much as possible
- thereby avoiding duplicated information
- NoSQL (MongoDB) - provision a data store, as easy as possible for the application to use

# Server-side considerations - data storage

---

## MongoDB - BSON

- BSON is the format used by MongoDB to store its data
- effectively, JSON stored as binary with a few notable differences
  - eg: *ObjectId* values - data type used in MongoDB to uniquely identify documents
  - created automatically on each document in the database
  - often considered as analogous to a primary key in a SQL database
- *ObjectId* is a large pseudo-random number
- for nearly all practical occurrences, assume number will be unique
- might cease to be unique if server can't keep pace with number generation...
- other interesting aspect of *ObjectId*
  - they are partially based on a timestamp
  - helps us determine when they were created

# Server-side considerations - data storage

---

## **MongoDB - general hierarchy of data**

- in general, MongoDB has a three tiered data hierarchy

### 1. database

- *normally one database per app*
- *possible to have multiple per server*
- *same basic role as DB in SQL*

### 2. collection

- *a grouping of similar pieces of data*
- *documents in a collection*
- *name is usually a noun*
- *resembles in concept a table in SQL*
- *documents do not require the same schema*

### 3. document

- *a single item in the database*
- *data structure of field and value pairs*
- *similar to objects in JSON*
- *eg: an individual user record*

# Server-side considerations - data storage

---

## *MongoDB - install and setup*

- install on Linux
- install on Mac OS X
  - again, we can use **Homebrew** to install MongoDB

```
// update brew packages  
brew update  
  
// install MongoDB  
brew install mongodb
```

- then follow the above OS X install instructions to set paths...
- install on Windows

# Server-side considerations - data storage

---

## *MongoDB - a few shell commands*

- issue following commands at command line to get started - OS X etc

```
// start MongoDB server - terminal window 1
mongod

// connect to MongoDB - terminal window 2
mongo
```

- switch to, create a new DB (if not available), and drop a current DB as follows

```
// list available databases
show dbs

// switch to specified db
use 424db1

// show current database
db

// drop current database
db.dropDatabase();
```

- DB is not created permanently until data is created and saved
  - *insert a record and save to current DB*
- only permanent DB is the local test DB, until new DBs created...

# Server-side considerations - data storage

---

## *MongoDB - a few shell commands*

- add an initial record to a new 424db1 database.

```
// select/create db
use 424db1

// insert data to collection in current db
db.notes.insert({
...   "travelNotes": [{
...     "created": "2015-10-12T00:00:00Z",
...     "note": "Curral das Freiras..."
...   }]
... })
```

- our new DB 424db1 will now be saved in Mongo
- we've created a new collection, notes

```
// show databases
show dbs

// show collections
show collections
```

# Server-side considerations - data storage

---

## **MongoDB - test app**

- now create a new test app for use with MongoDB
- create and setup app as before
  - *eg: same setup pattern as Redis test app*
- add **Mongoose** to our app
  - *use to connect to MongoDB*
  - *helps us create a schema for working with DB*
- update our `package.json` file
  - *add dependency for Mongoose*

```
// add mongoose to app and save dependency to package.json  
npm install mongoose --save
```

- test server and app as usual from app's working directory

```
node server.js
```



# Server-side considerations - data storage

---

## MongoDB - Mongoose schema

- use **Mongoose** as a type of bridge between Node.js and MongoDB
- works as a client for MongoDB from Node.js applications
- serves as a useful data modeling tool
  - *represent our documents as objects in the application*
- a data model
  - *object representation of a document collection within data store*
  - *helps specify required fields for each collection's document*
  - *known as a schema in Mongoose, eg: `NoteSchema`*

```
var NoteSchema = mongoose.Schema({  
  "created": Date,  
  "note": String  
});
```

- using schema, build a model
  - *by convention, use first letter uppercase for name of data model object*

```
var Note = mongoose.model("Note", NoteSchema);
```

- now start creating objects of this model type using JavaScript

```
var funchalNote = new Note({  
  "created": "2015-10-12T00:00:00Z",  
  "note": "Curral das Freiras..."  
});
```

- then use the Mongoose object to interact with the MongoDB
  - *using functions such as `save` and `find`*

# Server-side considerations - data storage

---

## **MongoDB - test app**

- with our new DB setup, our schema created
  - *now start to add notes to our DB, 424db1, in MongoDB*
- in our `server.js` file
  - *need to connect Mongoose to 424db1 in MongoDB*
  - *define our schema for our notes*
  - *then model a note*
  - *use model to create a note for saving to 424db1*

```
...  
  
//connect to 424db1 DB in MongoDB  
mongoose.connect('mongodb://localhost/424db1');  
  
//define Mongoose schema for notes  
var NoteSchema = mongoose.Schema({  
  "created": Date,  
  "note": String  
});  
  
//model note  
var Note = mongoose.model("Note", NoteSchema);  
  
...
```

# Server-side considerations - data storage

---

## MongoDB - test app

- then update app's post route to save note to 424db1

```
//json post route - update for MongoDB
jsonApp.post("/notes", function(req, res) {
  var newNote = new Note({
    "created":req.body.created,
    "note":req.body.note
  });
  newNote.save(function (error, result) {
    if (error !== null) {
      console.log(error);
      res.send("error reported");
    } else {
      Note.find({}, function (error, result) {
        res.json(result);
      })
    }
  });
});
```

# Server-side considerations - data storage

---

## MongoDB - test app

- update our app's get route for serving these notes

```
//json get route - update for mongo
jsonApp.get("/notes.json", function(req, res) {
  Note.find({}, function (error, notes) {
    //add some error checking...
    res.json(notes);
  });
});
```

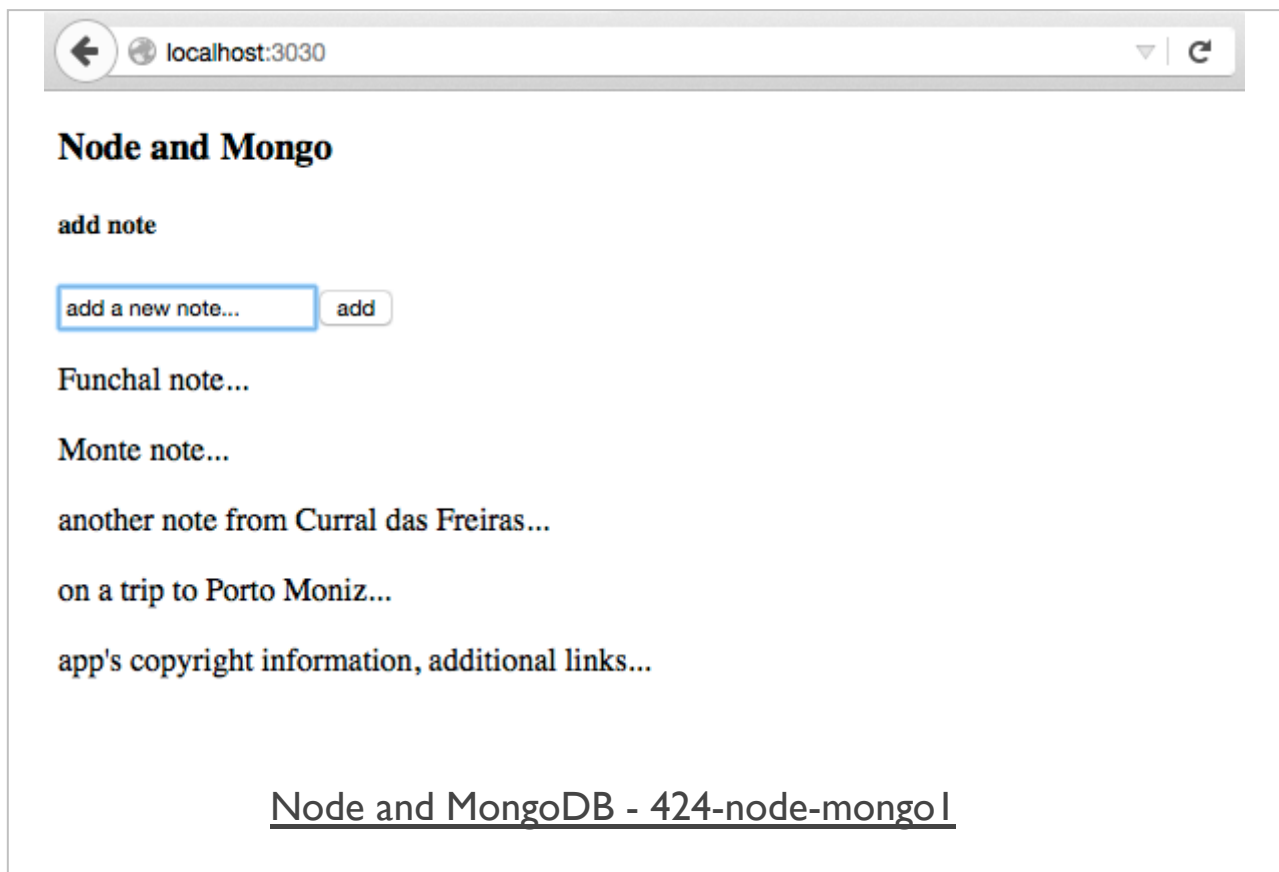
- modify buildNotes ( ) function in json\_app.js to get return correctly

```
...
//get travelNotes
var $travelNotes = response;
...
```

- now able to enter, save, read notes for app
- notes data is stored in the 424db1 database in MongoDB
- notes are loaded from DB on page load
- notes are updated from DB for each new note addition
- DEMO - 424-node-mongo I

# Image - Client-side and server-side computing

---



# Data visualisation

---

## *intro - part I*

- data visualisation - study of how to visually communicate and analyse data
- covers many disparate aspects
  - *including infographics, exploratory tools, dashboards...*
- already some notable definitions of data visualisation
- one of the better known examples,

*"Data visualisation is the representation and presentation of data that exploits our visual perception in order to amplify cognition."*

*(Kirk, A. "Data Visualisation: A successful design process." Packt Publishing. 2012.)*

- several variants of this general theme exist
  - *the underlying premise remains the same*
- simply, data visualisation is a visual representation of the underlying data
- visualisation aims to impart a better understanding of this data
  - *by association, its relevant context*

# Data visualisation

---

## *intro - part 2*

- an inherent flip-side to data visualisation
- without a correct understanding of its application
  - *it can simply impart a false perception, and understanding, on the dataset*
- run the risk of creating many examples of standard **areal unit** problem
  - *perception often based on creator's base standard and potential bias*
- inherently good at seeing what we want to see
- without due care and attention visualisations may provide false summations of the data

# Data visualisation

---

## types - part I

- many different ways to visualise datasets
  - *many ways to customise a standard infographic*
- some standard examples that allow us to consider the nature of visualisations
  - *infographics*
  - *exploratory visualisations*
  - *dashboards*
- perceived that data visualisation is simply a variation between
  - *infographics, exploratory tools, charts, and some data art*

### I. infographics

- *well suited for representing large datasets of contextual information*
- *often used in projects more inclined to exploratory data analysis,*
- *tend to be more interactive for the user*
- *data science can perceive infographics as improper data visualisation because*
- *they are designed to guide a user through a story*
- *the main facts are often already highlighted*
- **NB:** *such classifications often still only provide tangible reference points*



# Data visualisation

---

## types - part 2

### 2. exploratory visualisations

- *more interested in the provision of tools to explore and interpret datasets*
- *visualisations can be represented either static or interactive*
- *from a user perspective these charts can be viewed*
- *either carefully*
- *simply become interactive representations*
- *both perspectives help a user discover new and interesting concepts*
- *interactivity may include*
- *option for the user to filter the dataset*
- *interact with the visualisation via manipulation of the data*
- *modify the resultant information represented from the data*
- *often perceived as more objective and data oriented than other forms*

### 3. dashboards

- *dense displays of charts*
- *represent and understand a given issue, domain...*
- *as quickly and effectively as possible*
- *examples of dashboards*
- *display of server logs, website users, business data...*

# Data visualisation

---

## **Dashboards - intro**

- dashboards are dense displays of charts
- allow us to represent and understand the key **metrics** of a given issue
  - *as quickly and effective as possible*
  - *eg: consider display of server logs, website users, and business data...*
- one definition of a dashboard is as follows,

*"A dashboard is a visual display of the most important information needed to achieve one or more objective; consolidated and arranged on a single screen so the information can be monitored at a glance."*

*Few, Stephen. Information Dashboard Design: The Effective Visual Communication of Data. O'Reilly Media. 2006.*

- dashboards are visual displays of information
  - *can contain text elements*
  - *primarily a visual display of data rendered as meaningful information*

# Data visualisation

---

## *Dashboards - intro*

- information needs to be consumed quickly
- often simply no available time to read long annotations or repeatedly click controls
- information needs to be visible, and ready to be consumed
- dashboards are normally presented as a complementary environment
- an option to other tools and analytical/exploratory options
- design issues presented by dashboards include effective distribution of available space
- compact charts that permit quick data retrieval are normally preferred
- dashboards should be designed with a purpose in mind
- generalised information within a dashboard is rarely useful
- display most important information necessary to achieve their defined purpose
- a dashboard becomes a central view for collated data
- represented as meaningful information

# Data visualisation

---

## *Dashboards - good practices*

- to help promote our information
  - *need to design the dashboard to fully exploit available screen space*
- need to use this space to help users absorb as much information as possible
- some visual elements more easily perceived and absorbed by users than others
- some naturally convey and communicate information more effectively than others
- such attributes are known as **pre-attentive attributes of visual perception**
- for example,
  - *colour*
  - *form*
  - *position*

# Data visualisation

---

## **Dashboards - visual perception**

### ■ pre-attentive attributes of visual perception

#### 1. Colour

- *many different colour models currently available*
- *most useful relevant to dashboard design is the HSL model*
- *this model describes colour in terms of three attributes*
  - *hue*
  - *saturation*
  - *lightness*
- *perception of colour often depends upon context*

#### 2. Form

- *correct use of length, width, and general size can convey quantitative dimensions*
- *each with varying degrees of precision*
- *use the Laws of Prägnanz to manipulate groups of similar shapes and designs*
- *thereby easily grouping like data and information for the user*

#### 3. Position

- *relative positioning of elements helps communicate dashboard information*
- *laws of Prägnanz teach us*
- *position can often infer a perception of relationship and similarity*
- *higher items are often perceived as being better*
- *items on the left of the screen traditionally seen first by a western user*

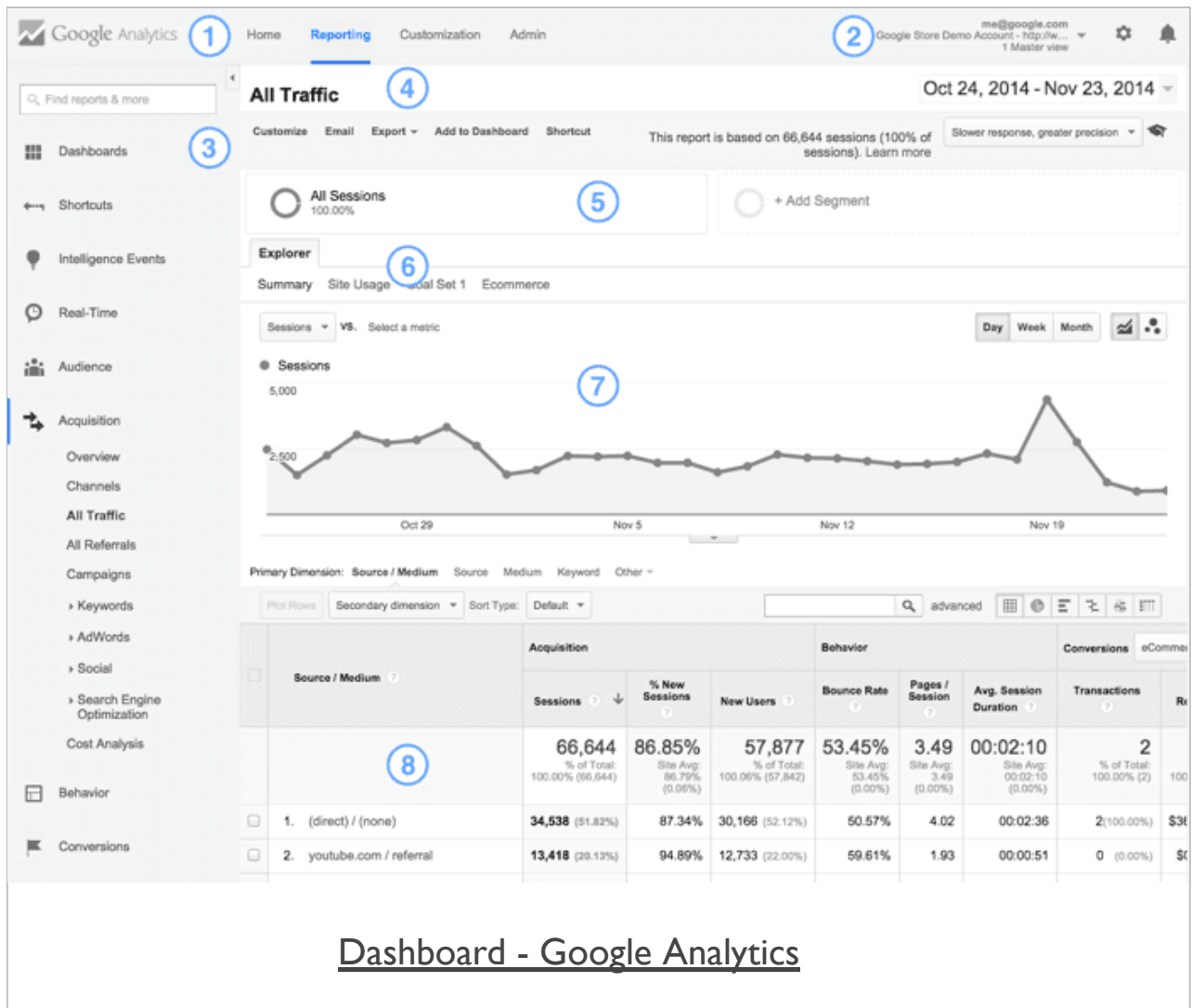
# Data visualisation

---

## ***Building a dashboard***

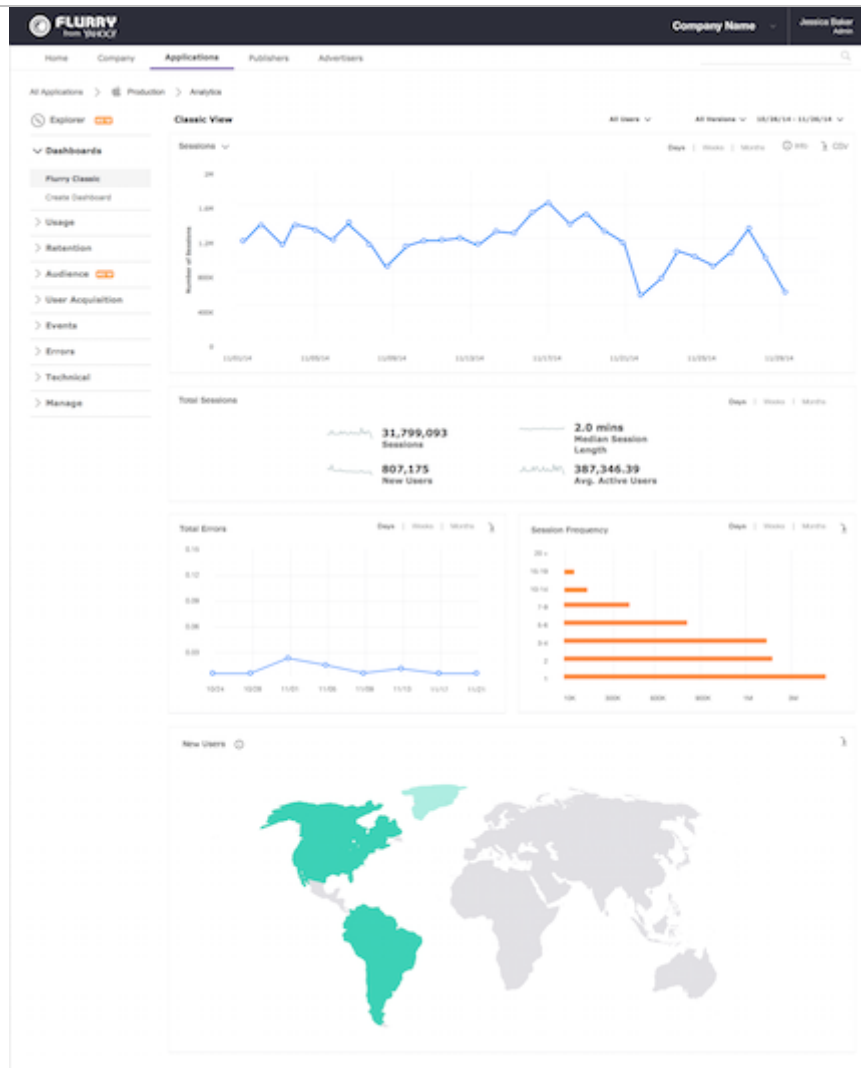
- need to clearly determine the questions that need to be answered
  - *given the information collated and presented within the dashboard*
- need to ensure that any problems can be detected on time
- be certain why we actually need a dashboard for the current dataset
- then begin to collect the requisite data to help us answer such questions
  - *data can be sourced from multiple, disparate datasets*
- chosen visualisations help us tell this story more effectively
- present it in a manner appealing to our users
- need to consider information visualisations familiar to our users
  - *helps reduce any potential user's cognitive overload*
- carefully consider organisation of data and information
- organise the data into logical units of information
  - *helps present dashboard information in a meaningful manner*
- dashboard sections should be organised
  - *to help highlight and detect any underlying or prevailing issues*
  - *then present them to the user*

# Image - Google Analytics



Dashboard - Google Analytics

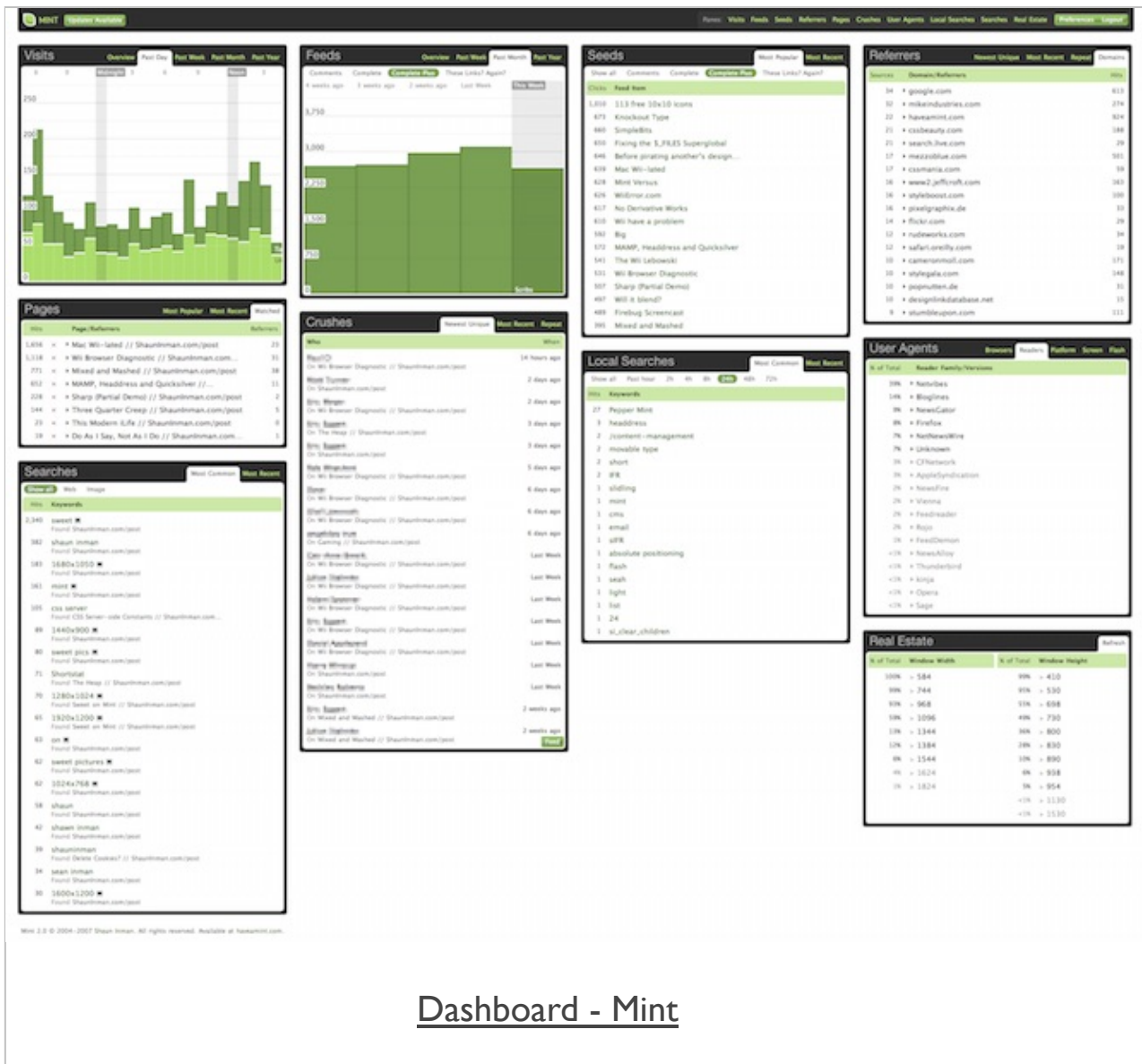
# Image - Yahoo Flurry



Dashboard - Yahoo Flurry



# Image - Mint



## Dashboard - Mint

# Demos

---

## Redis

- `424-node-redis l`

## MongoDB

- `424-node-mongo l`

# References

---

- Chocolatey for Windows
  - *Chocolatey package manager for Windows*
- Homebrew for OS X
  - *Homebrew - the missing package manager for OS X*
- Kirk, A. *Data Visualisation: A successful design process*. Packt Publishing. 2012.
- MongoDB
  - *MongoDB - For Giant Ideas*
  - *MongoDB - Getting Started (Node.js driver edition)*
  - *MongoDB - Getting Started (shell edition)*
- Mongoose
  - *MongooseJS Docs*
- Node.js
  - *Node.js home*
  - *Node.js - download*
  - *ExpressJS*
  - *ExpressJS body-parser*
- Redis
  - *redis.io*
  - *redis commands*
  - *redis - npm*
  - *try redis*
  - *Windows support*
- W3 Selector API