A QUICK INTRODUCTION

ZOTONIC DRIVE THROUGH

OVERVIEW — WHAT IS ZOTONIC?

- CMS/Framework
- Erlang
- PostgreSQL
- Sweet spot
- Batteries included

OVERVIEW — CMS / FRAMEWORK

- Extensible, modular system.
- All functionality is implemented using modules.
  - Or a site, which is a special module.
- Lots included in the base system.
  - Admin / media handling / SEO / SSL / etc.
- Virtual hosting - many sites on a single system.

OVERVIEW — ERLANG

- Functional programming language.
- Used in telephone switches, WhatsApp, etc.
- Multi-threaded, can handle millions of processes.
- Hot code upgrade.
- Rock solid.
- “Let it crash” error handling, gives much simpler code.
  - Process isolation and supervision.
  - Program the “happy path.”

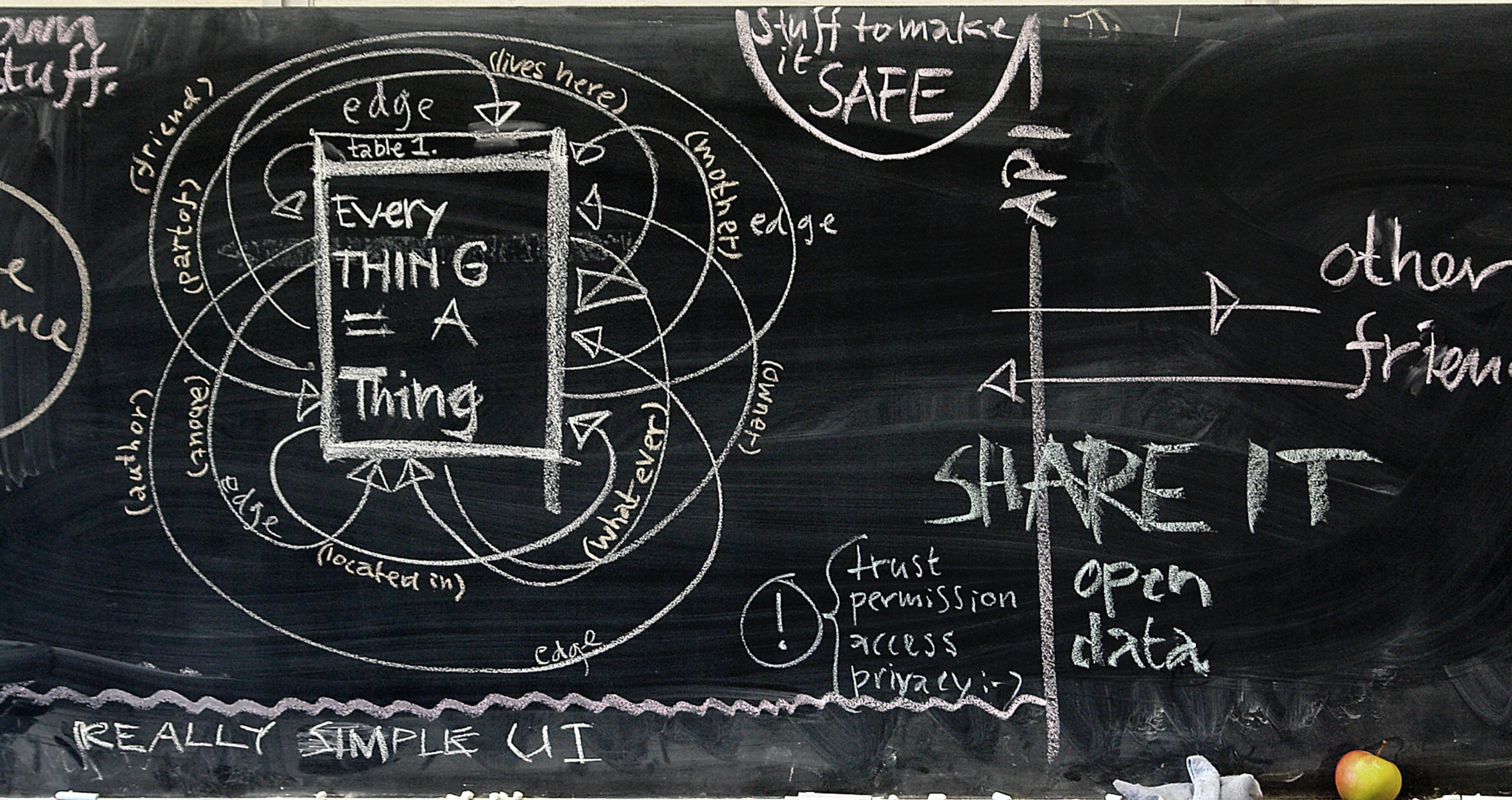
OVERVIEW — POSTGRESQL

- Stable, very stable.
- Scales nicely with the number of CPU cores.
- Good join performance (not so good in MySQL).
- Support for JSON, Geo, full text search.
- No plans to support other databases.
- Schema (or database) per site.
- Sites can run without a database.

OVERVIEW — SWEET SPOT

- Semantic data.
  - More later in the datamodel overview.
- Separation between data and representation.
- Many parallel TCP connections.
  - Limit is the amount of memory.
- Single machine.
  - Upscales very good, no need yet for distributed.

DATA MODEL


edge
table 1.
Every
THING
=
A
Thing
(lives here)
(friend of)
(part of)
(mother)
edge
(author)
(owner)
edge
(located in)
(whatever)
edge
stuff to make it SAFE
other
friend
SHARE IT
open data
!
trust
permission
access
privacy :-)
REALLY SIMPLE UI

DATA MODEL

- Resources
- Categories
- Edges
- Predicates
- Medium
- Searching
- Identity
- Access control

RESOURCES / CATEGORIES

- Everything is a thing.
- A thing is called a “resource” (semantic web terminology) or “page” (for editors and other normal humans).
- One table holding all resources in serialized form.
- Category of a resource defines *what* it is (person, article, keyword).
- Categories are organized in a hierarchy.
  - Example: news is an article, an article is a text.
- Categories themselves are resources (“things”) of the category “category.”

EDGES / PREDICATES

- Edges are directed connections between resources.
- Every edge has a label, the *predicate*.
- An edge defines a meaningful relation between resources.
  - Example: A *book* (subject) has as *author* (predicate) a *person* (object).
- A predicate is a resource, of the category *predicate*. The resource describes and names the predicate

MEDIA – IMAGES / VIDEO / DOCUMENTS

- Every resource can contain a single media item.
- This media item can be a file, embed code, or something else.
- Usually we place resources with a media item in the *media* category, with sub-categories *image*, *video*, *audio* and *document*.
- Built-in support for resizing, image manipulation, video processing, EXIF handling, and more.

SEARCHING — PIVOT / FACET

- Resources are stored in serialized blobs.
- Pivoting is the process to extract properties from those blobs and place them in indexed columns and tables.
  - Pivoting is done after a resource has been updated.
    - Define special pivot tables per project.
    - Template for the default pivot columns.
  - Facets are used for searching, they are used to *drill down* in search results.
    - Template to define the facets per site.

IDENTITY

- A person / user is a resource.
- Password and username are stored in the identity table, not in the serialized data.
- As are other identities:
  - Extra email addresses (primary *is* in the serialized data).
  - Tokens from services like Facebook, Twitter etc.
  - Login secrets

ACCESS CONTROL

- Access control defines what a user can see and do.
- Access control is done at a low level, in the models (later more about models).
- Default access control module:
  - Places users into user groups.
  - Places resources into content groups.
  - Gives rights (view, edit, delete, link) to user groups on content groups.
  - Gives rights to upload media, per mime-type and size.
  - Gives rights to *use* certain modules.

REQUEST HANDLING

REQUEST HANDLING

- Dispatching
- Controllers
- Templates
- Models

DISPATCHING

- Dispatching is the process of mapping the incoming HTTP request to a site and controller.
- Site configurations define the hostnames a site handles.
- Dispatch rules define the mapping from an URL path to a controller (and arguments).
- In reverse, dispatch rules are also used to *generate* URLs for resources and other pages or links.
- Dispatch rules are defined in files (later more).

CONTROLLER

- HTTP requests are handled by the HTTP protocol handler Cowmachine.
- Controllers define callbacks for the HTTP protocol handling. Examples: is_authorized, resource_exists, process.
- A controller is an Erlang module and defined in a site or Zotonic module.
- Most used controllers are:
  - controller_page: serves HTML pages for resources.
  - controller_template: serves HTML pages from templates.

TEMPLATES

- Templates are used to generate HTML pages. Based on Django syntax.
- Templates can *extend* other templates.
- Templates can *overrule* same named templates in other modules.
- Templates pull their information from *models*, controllers pass minimal information, like the *id* of the resource being handled.
- Notable templates:
  - base.tpl: implements the main page structure, including head and css/js.
  - page.tpl: used for displaying a generic resource.
  - page.categoryname.tpl or page.name.somename.tpl to display a specific category or uniquely named resource.

MODELS

- Used to access data.
- Accessible from:
  - Templates: m.modelname.foo
  - HTTP: https://example.test/api/model/modelname/get/foo
  - MQTT: model/modelname/get/foo
- Main models:
  - rsc: access resource data and properties, example: m.rsc[id].title
  - edge: access edge information
  - search: perform all kinds of searches on resources or other data

COTONIC


SEUM OF SCOTLAND
TOILETS
FLODDEN WALL
MAGDELEN CHAPEL
CASTLE TERRACE CAR PARK

COTONIC – INFRASTRUCTURE IN THE BROWSER

- Javascript framework.
- MQTT message bus between browser and server.
- Topic tree in browser, connects components and browser tabs.
- Bridge topics to send messages in browser to server and from server to browser.
- Manages web workers for authentication, file uploads and more.
- Elm uses MQTT topics to access server and client models.

WIRES

WIRES – SIMPLE ACTIONS WITHOUT JAVASCRIPT

- Tags in templates, attaches actions to elements.
  - {% wire id="..." type="submit" action=... %}
- Attaches to forms, button, and other elements.
- Actions like confirms, postback (to server), dialogs, show/hide etc.
- Postback events route to *delegate* Erlang modules event functions.

MESSAGE BUSES

MESSAGE BUSES

- Notification bus in Erlang
  - Extension mechanism for low level, modules.
  - Trusted messages.
  - z_notifier and observe, with notify, first, fold, and map.
- MQTT bus for outside data.
  - Untrusted user facing messages.
  - MQTT topic tree with publish and subscribe.

DIRECTORY STRUCTURE — CODE ORGANIZATION

DIRECTORY STRUCTURE — CODE ORGANIZATION

- Sites and modules are Erlang OTP applications
- Zotonic is an *umbrella* application, containing multiple other applications (sites, modules).
  - apps: core modules and applications
  - apps_user: sites, extra modules

DIRECTORY STRUCTURE — SITE / MODULE

- Erlang OTP application.
- Makefile for building.
- rebar.config to define dependencies, compile options.
- *priv* directory for assets, static files, templates.
- *src* directory for Erlang code
- Zotonic module indexer looks for all templates, models, filters, actions etc. in the site and module directories.

DIRECTORY STRUCTURE — PRIV

- *priv* directory contains all static non erlang files.
  - priv/zotonic_site.config for site configuration.
    - priv/config.d/... for extra site configuration.
  - priv/dispatch/... for dispatch files.
  - priv/lib/... for images, css, javascript
  - priv/lib-src/... for scss and Makefiles to generate priv/lib/ files.
  - priv/templates/...
  - priv/translation/... for .po files

DIRECTORY STRUCTURE — SRC

- *src* directory contains all erlang files, main content:
  - src/appname.app.src
  - src/mod_foobar.erl (or src/sitename.erl)
  - src/models/m_mymodel.erl
  - src/controllers/controller_foobar.erl
  - src/filters/filter_myfilter.erl
  - src/support/… for extra Erlang source files.
- Erlang file names MUST be unique. Use site/module specific prefixes.