Bristol Old Vic
Archive

Technical Documentation

Table of Contents

Introduction

This document is for the attention of any future developer(s) or technical staff working at or for the Bristol Old Vic on their archiving project.

When writing this document I have taken the assumption that the reader is competent with web framework development, Symfony in particular, the Doctrine ORM, and general MVC convention.

Should you meet the criteria above but have little to no experience using Symfony and/or Doctrine I would recommend the materials listed in 'Additional Reading'.

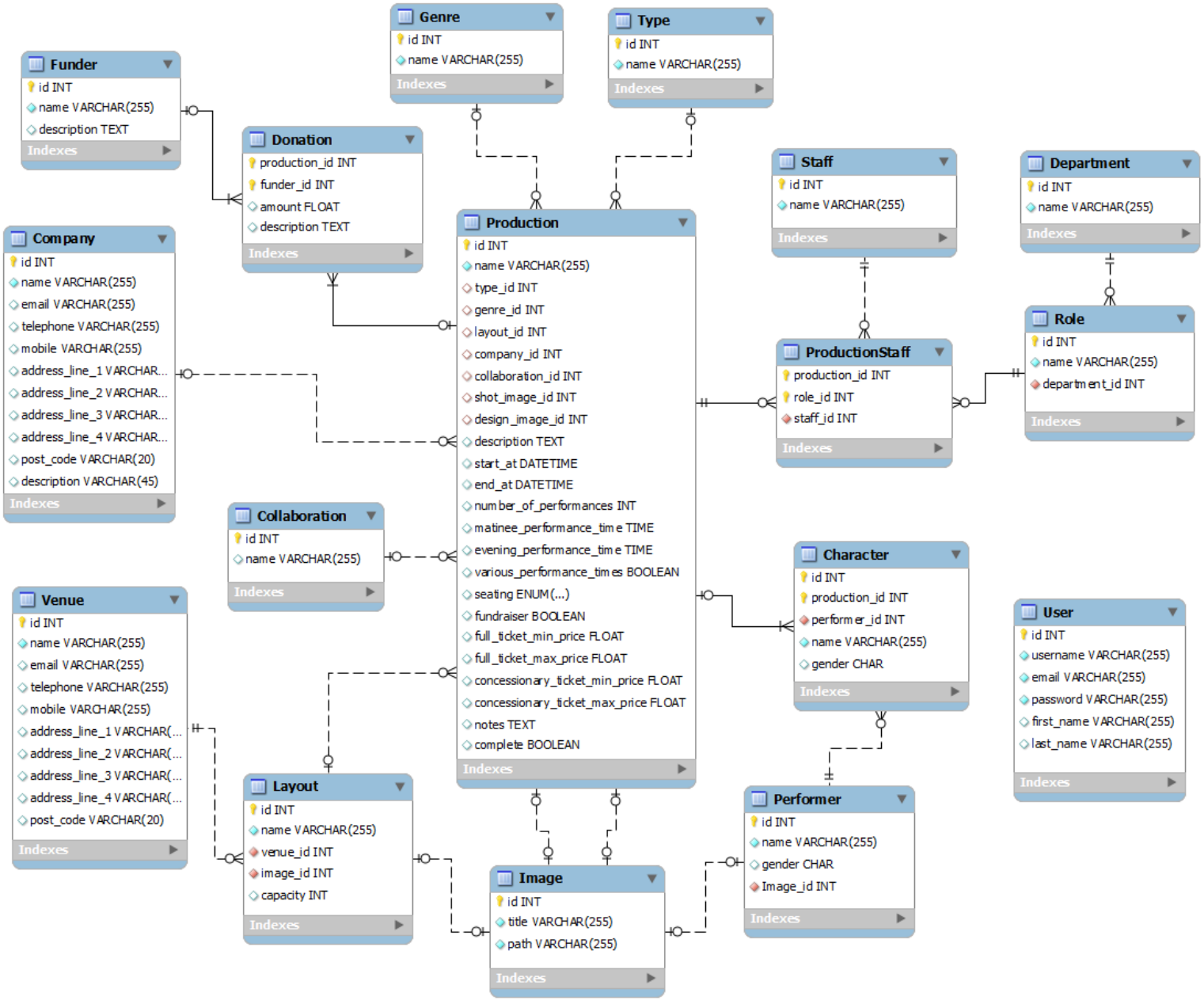
Resources

Live Archive:	http://archive.bristololdvic.org.uk
Demo Archive:	http://bristololdvic.demo.stevelacey.net
Source Code:	https://github.com/stevelacey/bristol-old-vic-archive

Additional Reading

Symfony Documentation:	http://www.symfony-project.org/doc/1_4
Doctrine ORM 1.2 Docs:	http://www.doctrine-project.org/projects/orm/1.2/docs/en

Entity Relationship Diagram



Data Modelling Commentary

This section documents any aspects of the data model [see: Entity Relationship Diagram] that is not inherently easy to understand or requires additional explanation.

Staff Production Roles

A staff member can have an infinite number of roles on a production, but a role on a production can only be filled by one staff member; i.e. there can only be one director, one producer, one stage assistant.

Should there be the need for multiple staff to be assigned the same role, i.e. 'Camera Man', multiple roles would have to be created, presumably numerically iterative; 'Camera Man 1,2,3' and so on. Additionally, roles needed to be customisable by the Bristol Old Vic hence the need for this data to be stored in the model rather than as fields on productions. To solve these requirements, the three-way link table structure was devised, allowing for the above criteria to be met.

Departments, whilst related to roles does not have much inherent meaning, they are primarily used as an arbitrary means of grouping on production forms, they are not integral to the dataset, more a convenience from a usability perspective.

Person, Staff & Performers

The need for multiple types of 'people' within the system made it suitable for the use of inheritance. Ideally, column aggregation would have been the best solution. However, Symfony handles concrete inheritance far more gracefully, with only downside being a redundant 'person' database table (omitted from the diagram). The table is unused by the archive and its existence allows for Doctrine to also create the Staff and Performer models inherited from the Person model- meaning the relationship to an Image and basic details such as 'name' and custom methods don't need repeating.

Any changes that need to be made to any aspect of both the Staff and Performer entities should be made on Person and allowed to cascade as appropriate.

Cascading Deletes

All deletes should cascade correctly across their related entities. Productions are never deleted based on a cascade. If companies or genres etc. are deleted, then the respective fields on production are set to null. If a production is deleted then the related staff, performers, donations etc. are all unbound correctly via deletion of rows within the appropriate link tables.

Images

The image model is relatively basic, for simplicity many-to-many relationships were not implemented, Bristol Old Vic at present has no need for setting an infinite number of images on any entities, simply:

- Performer
  - Photo
- Production
  - Shot (the poster)
  - Set design
- Staff
  - Photo
- Venue Layouts
  - Layout blueprint (seating plan / stage setup)

Plugins

Since this build is primarily a centralised administrative system, I took the objective of utilising as much built-in and plugin functionality to get stuff done rapidly, consistently and ensuring that it is easy to alter, maintain and cascade.

sfAdminDash

http://www.symfony-project.org/plugins/sfAdminDashPlugin

This plugin handles most of the UI, I stuck with the default Joomla-esque theme, and we've patched on some extra styles for our custom data views which can be found in the css folder in the web directory as well as the Bristol Old Vic branding.

Obviously, we swapped out the icons for the dashboard, but other than that the sfAdminDash web folder should be pretty much untouched should it need updating at a later date.

sfDoctrineGuard

http://www.symfony-project.org/plugins/sfDoctrineGuardPlugin

This is the standard authentication module for Symfony (with Doctrine). At the time of writing everything within the system requires authentication bar the login and forgotten passwords interfaces.

There is only one tier of user hierarchy, in which all users have administrative rights over the data stored in the system, including other users.

This is set to change as part of the future working, the plugin has good support for different levels of hierarchy, and I would recommend using the super-admin functionality for restricting user management.

For read-only functionality to be implemented the show interfaces for all entities would need creating, at present only edit interfaces are available but these should be too hard to generate and tweak.

sfImageTransformPlugin

http://www.symfony-project.org/plugins/sfImageTransformPlugin

This plugin handles all the image manipulation for the image object that relates to productions, staff/performers and venue layouts. Raw images are stored in web/uploads/images and a thumbnail cache generated by the plugin is maintained in web/uploads/thumbnails.

The thumbnails are organised by their dimensions and the directory is symlinked to web/images/thumbnails to allow for direct access once generated, allowing the htaccess to route users to the physical image rather than loading Symfony and triggering a re-generation of existing thumbnails.

The live environment is configured to use the ImageMagick adapter by default, GD is supported if IM is unavailable, but produces thumbnails of an inferior quality in my experience.

Data Views

Various entities have useful contextual information based on their relationships. Rather than implement new interfaces for this data I decided to patch it on to the side of the relative edit forms. To do this I have added an extra dash partial which is included directly after the form within the same dividing block.

Generally these sections contain information for example the productions a staff member has been involved in, and how they were involved, or all the productions that have been performed at a venue.

Forms

A few of the interfaces require embedded relations to be created on-the-fly. This has been implemented using embedded forms. To maximise usability we've based the number of forms that are displayed based on the common use scenarios for Bristol Old Vic. Production cast are usually around 5 people, hence there are 5 character forms as a minimum. Additionally, there is a minimum of 2 blank forms at any one time, thus if you were to create a production, and add 5 characters, post-save, 2 new forms would appear, and so on.

Similar rules apply for other relationships such as venue layouts. The minimum values are defined on a per-model basis in app.yml.

Reporting

The reporting functionality we have implemented is simply a print stylesheet, we've tried to hide all unnecessary UI to save ink and set some global styles to reposition content to suit the printed medium.

At the time of writing this is pretty basic, eventually they will probably need more comprehensive reporting functionality but for now at least it's acceptable and just worth keeping an eye on when adding new interfaces to ensure that the prints are still logically structured.

The pages that support direct browser printing are all list and edit views, most pages are compatible.

Future Works

There are a variety of future works that we discussed with Bristol Old Vic and decided were beyond the scope and time constraints imposed on our project, or just seem like a logical progression for the system:

- Implement a more advanced user hierarchy:
    - Adjust user model (and implement interfaces) so that users only have read-only access to data.
    - Implement administrator user level for managing data (at present all users can do this).
- Hyperlinks between production pages and their related entities i.e. staff and performers.
- Excel (CSV) export functionality.
- Public interfaces for non-private data.
- An XML / JSON API.

Hosting

The archive is currently hosted on a Rackspace cloud server, credentials of which are held by the Bristol Old Vic on-site technician Jason Barnes.

Backups

The server is configured to push entire backups of the MySQL database out to off-site Rackspace cloud file storage every half an hour, accessible via the same credentials at http://manage.rackspacecloud.com.

All source code is stored in Git, and as per its distributed nature, there are a few complete backups of the repository, the most conveniently available being the one on Github as listed in the resources section.

Security

The server is configured to automatically check for and install security updates from the Ubuntu repository. Additionally, there is firewall protection and port restriction in place minimising the risks to external attacks.

System Requirements

- PHP 5.3+
- PDO + MySQL Drivers
- A PHP Accelerator such as APC
- MySQL 5
- ImageMagick (or GD)

The archive currently resides on Ubuntu 10.04 LTS (Lucid Lynx), it should work with ease on any *nix-based operating system, changes would need to be made if you need to run it on Windows, at the very least to directory separators for image thumbnail generation, and cache deletion.