

Advance Web 2

by Sam Phoenix

FILE	2644_SAM_PHOENIX_ADVANCE_WEB_2_93284_770303932.DOCX		
TIME SUBMITTED	30-MAR-2018 10:18PM (UTC+0100)	WORD COUNT	3291
SUBMISSION ID	85888358	CHARACTER COUNT	21564

ADVANCED WEB, PART 2

I. Part A: Understanding	2
A. Page and Front Controllers	2
B. Template Views	2
C. MVC Frameworks	2
1. Symfony	3
2. CakePHP	3
3. Laravel.....	4
II. Part B: Implementation	6
A. About the app	6
B. Testing the app.....	6
III. References	16
IV. Appendix I: Code	17
A. Model.....	17
1. SQL data for BugCollection table	17
2. BugCollection.php	18
B. View	20
1. Collection.html.twig	20
C. Controller	21
1. BugCollectionController.php	21
D. Config	21
1. Doctrine configuration from config.yml.....	21
2. Database setup in parameters.yml	22

I. Part A: Understanding

This section explores different concepts of controllers and views in relation to the Model-View-Controller (MVC) design pattern and compares three different PHP frameworks that use this approach.

A. Page and Front Controllers

PHP was designed as a templating, server-side scripting language, and has evolved over the years to accommodate the growing demands for it. While more recent versions now include features such as type-hinting, there is still the problem of “spaghetti code” – logic tends to be embedded in the html of a page and duplicated across the web application.

Front controllers provide a central point in the application that handles all incoming and outgoing requests, such as connection to the database, security details and queries. It is easier to maintain, as there is only one file that contains the underlying code; it also provides more security as all processes in the app must run through this file. It is also easier for testing and debugging.

This method is used in the Symfony framework[1]. The `app.php` (or `app_dev.php` while in development) file handles all requests and responses in the app, using a routing configuration to forward URL addresses to the right controllers, which in return render the views to display the data.

B. Template Views

Standalone PHP can be used to cut down on the repetition of code while rendering files; e.g. using `include_once` to render the header and footer of a web page. However, if there are more than one page displaying information in a similar way, there is still a lot of duplication in the rendering of a view.

Template views solve this problem by offering a master view, which contains the main structure of the site including header, footer and navigation, along with code to allow overwriting of sections by sub-pages. The template engine Twig[2] provides extra security features, such as automatic output escaping, that in PHP you need to remember to write into every page where the user can submit information (i.e. with `htmlspecialchars`), as well as optimising the PHP code during compilation.

C. MVC Frameworks

While standalone PHP is fine for smaller applications, as web apps grow in size and complexity, they become difficult to maintain. Often there is a tight coupling between the view and the logic that displays the data, with SQL queries being run in the same page that shows the results.

MVC frameworks were built with the purpose of making development easier. The aim is to separate the logic of the application from the display, with controllers providing the link between the two. In developing complex apps, this has the additional benefit of allowing web developers to work on the functionality of the app, with web designers working on the code for the views, with no requirements for the latter to understand the former, or have any knowledge of the behaviour that will be rendered by the view.

Three popular frameworks in use are compared below:

1. **Symfony**

According to a framework reviewer[3]:

If I was going to mention one framework, which has been a rock-solid staple of the PHP community, it has to be Symfony.

SensioLabs, the company behind Symfony and other related components, has been in operation for over twenty years, and Symfony remains one of the most well-known and used MVC frameworks.

It uses the philosophy of “configuration over convention” – some conventions are applied, such as controller files needing to follow the convention [filename]Controller, but routing is set up via either yml config files, or annotations which take the appearance of comments within the PHP classes. There is, therefore quite a learning curve, although there is ample documentation to assist in setup and starting out.

Symfony is modular by design; additional components, or “bundles” may be installed when needed, mostly via SSH session using Composer, so knowledge of SSH terminal commands are needed to gain maximum benefit. However, its object-relational mapping (ORM) tool Doctrine provides some useful features, such as auto-generating model (entity) classes from an existing database, its own form of SQL (DQL) for interacting with the model, and a CRUD generator for providing simple controllers and views that are set up to handle most database operations.

Symfony is, however, quite tricky to install for development purposes. When using it for my test app, I ended up using my own web hosting account and its built-in app installer to install version 3.4. IDEs like PHPStorm do work well with Symfony, which makes development on a remote server much easier.

2. **CakePHP**

CakePHP follows the “convention over configuration” approach. According to the official website[4]:

We are big fans of convention over configuration. While it takes a bit of time to learn CakePHP's conventions, you save time in the long run. By following conventions, you get free functionality, and you liberate yourself from the maintenance nightmare of tracking config files. Conventions also make for a

very uniform development experience, allowing other developers to jump in and help.

In practice, this means that all file names and database tables need to be named a certain way to gain maximum benefit from this approach, although it means that CakePHP will provide the routing setup merely from recognising naming patterns. It also means that, with appropriately named database tables, model classes will be automatically generated as needed.

It offers many features out of the box, such as security for web forms and database handling, and they provide a downloadable “cookbook” that contains most of the documentation to help the user get started. However, if additional features are required, the extensions are somewhat lacking and outdated.

It has also lost popularity in recent years, as the review website states[3]:

CakePHP really stood out as a top framework back in 2014, however it has lost its edge over the years and got overshadowed by the ever-popular Laravel. That said, CakePHP still has many beloved features and mantras such as convention-over-configuration approach. Plenty of automagic for request handling and form validation. As well as and an excellent ORM featuring a Repository pattern.

One of the complaints about CakePHP is its support of database systems; it supports MySQL, but not MongoDB, which has been growing in popularity over the years.

3. Laravel

Laravel is a newer PHP framework that is based on the Symfony framework that has become a market leader. Because it is so popular, there is a lot of up to date support, including video tutorials and IDE plugins.

Laravel also favours the “configuration over convention” approach; however, the way routes are configured (for example) appear to be cleaner and generally easier to learn than that of Symfony.

As far as components, the review site states[3]:

I'd say that out of the box Laravel is the only framework that can truly allow a developer to easily bootstrap an entire application lifecycle (SDLC) from `git clone` to fully-fledged production deploy, using all the tooling within the framework's ecosystem.

A good example of this is the user authentication system. Symfony provides a basic security system, recommending the use of a third party bundle called FOSUserBundle to provide an easy way to generate a user registration system[5]. This works well in theory, but in practice, there were problems trying to get this working (see Figure 1).

Laravel, however, provides this out of the box with a very easy setup, as quoted below[6]:

***Want to get started fast?** Just run `php artisan make:auth` and `php artisan migrate` in a fresh Laravel application. Then, navigate your browser to `http://your-app.test/register` or any other URL that is assigned to your application. These two commands will take care of scaffolding your entire authentication system!*

The commands quoted above provide the models required, the views for all aspects of user management (registration, login, forgot password etc.), as well as the necessary controllers; similar to the way FOSUserBundle functions, except no extra downloading of modules is required, and it actually works.

For this and other reasons, such as cleaner syntax, Laravel provides an easier learning experience and faster development for users wishing to employ rapid application development.

II. Part B: Implementation

Below is discussion about the app being built. The app itself is a test bed for something larger, which will be developed later.

A. About the app

This app is the beginning of quite a complex, potentially large web app that I wish to develop. Since Symfony is designed to be assistive to developing such apps, it is ideal for this project.

This will be a helper app for players of The Sims 2 (which may be extended to cover others in the series), to aid them in keeping track of their Sims. The database to cover this is very large (and at time of writing, not even complete), so for simplicity's sake, only the code relevant to testing this app will be submitted.

Eventually, it will allow players to track all kinds of different player aspects related to gameplay, but for now, it merely displays the collectable bugs for owners of the FreeTime expansion.

B. Testing the app

Symfony turns out to be quite fiddly to get working, depending what the developer wants to do. It has some useful features, such as a templating engine (Twig), the ability to install components as and when you need them, and built in support for Doctrine, an ORM tool that supports MySQL and MongoDB.

However, being able to install components can sometimes cause more problems than they solve. When trying to make use of the FOSUserBundle, for instance, I was met with the following error:

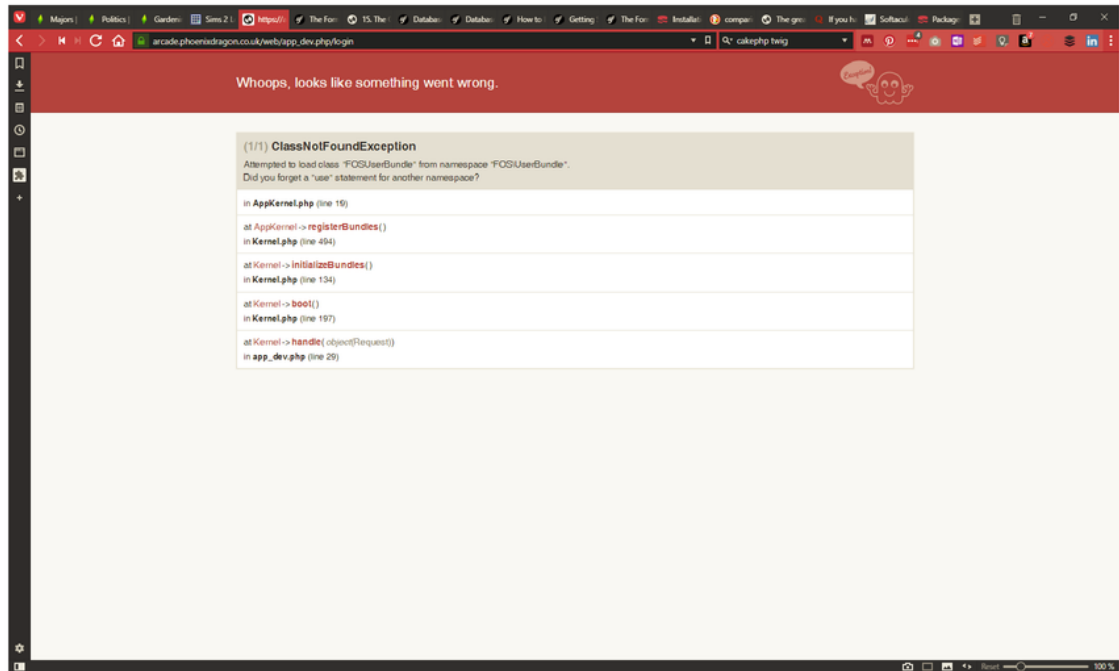


Figure 1: Trying to use a bundle in Symfony doesn't always work

Symfony seems to work on the principle of “configuration over convention”. While this can offer more flexibility, partly in the form of plugins, it also adds to the learning curve and to the amount of files installed and referenced on the server.

When the modules work well, however, it can make development a lot easier. Doctrine offers some automated tools to assist in working with databases. Figures 2 – 5 show some experimenting with the CRUD generator, which provides an interface via SSH (accessed through PuTTY) that asks a series of questions, before generating the controllers and views needed for a basic CRUD system – although as Figure 5 shows, the create option does not work; possibly because a form module was needed.

```
nlsrc3.supercp.com - PuTTY

Welcome to the Doctrine2 CRUD generator

This command helps you generate CRUD controllers and templates.

First, give the name of the existing entity for which you want to generate a CRUD
(use the shortcut notation like AcmeBlogBundle:Post)

The Entity shortcut name: AppBundle:Job

By default, the generator creates two actions: list and show.
You can also ask it to generate "write" actions: new, update, and delete.

Do you want to generate the "write" actions [no]? yes

Determine the format to use for the generated CRUD.

Configuration format (yaml, xml, php, or annotation) [annotation]:

Determine the routes prefix (all the routes will be "mounted" under this
```

Figure 2: Experimenting with Doctrine's CRUD generator for an empty database table

```
nlsrc3.supercp.com - PuTTY

CRUD generation

created ./src/AppBundle/Controller/JobController.php
created ./app/Resources/views/job/
created ./app/Resources/views/job/index.html.twig
created ./app/Resources/views/job/show.html.twig
created ./app/Resources/views/job/new.html.twig
created ./app/Resources/views/job/edit.html.twig
created ./src/AppBundle/Tests/Controller/
created ./src/AppBundle/Tests/Controller/JobControllerTest.php
Generating the CRUD code: OK
created ./src/AppBundle/Form/
created ./src/AppBundle/Form/JobType.php
Generating the Form code: OK
Updating the routing: OK

Everything is OK! Now get to work :).

phoenix9@nlsrc3 [~/arcade.phoenixdragon.co.uk]#
```

Figure 3: Generation complete in PuTTY

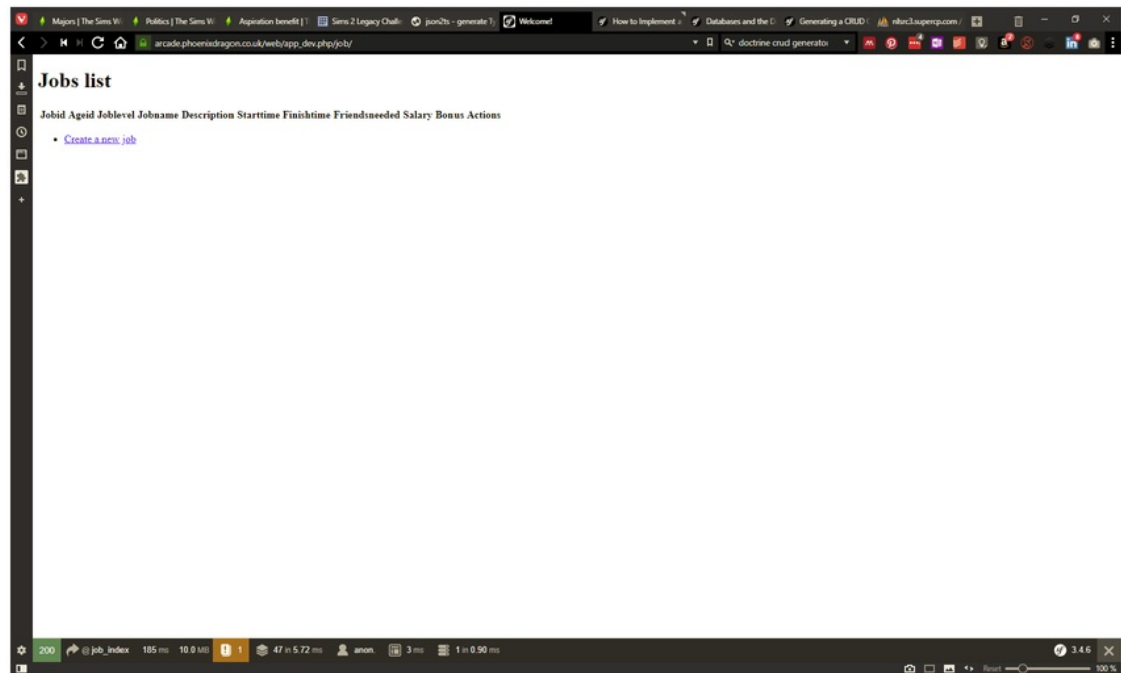


Figure 4: The page generated by the CRUD generator

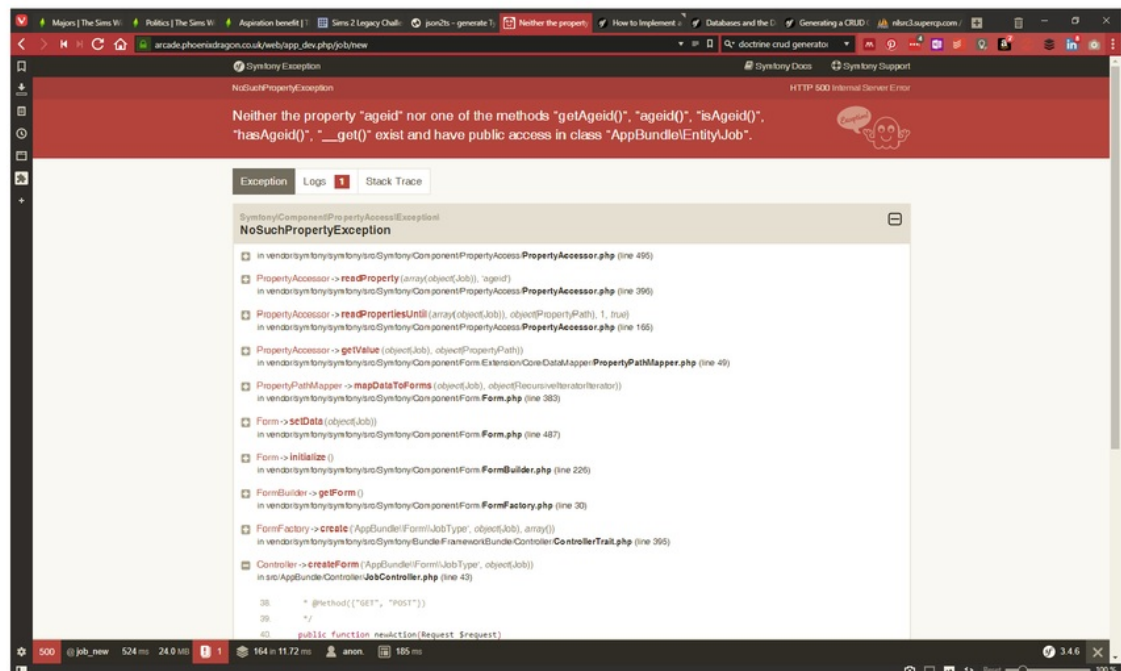
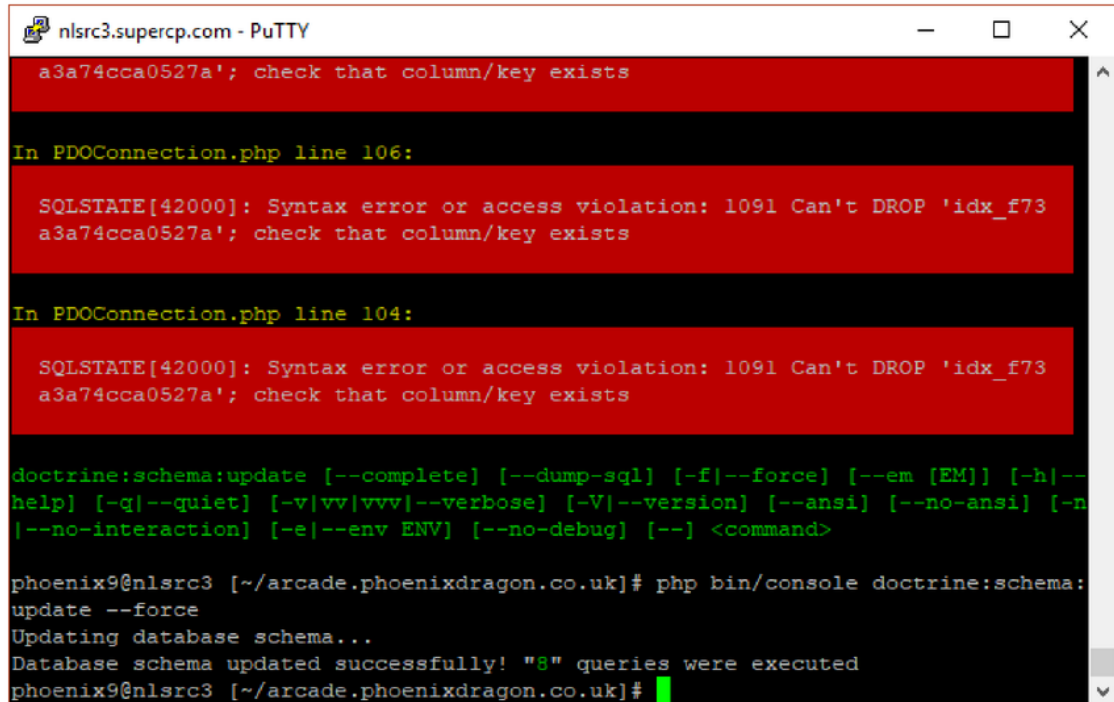


Figure 5: Creating a new item doesn't work

Doctrine also has the extremely useful feature of generating class files from an existing database[7]. This is helpful for those developers who build the database in SQL before beginning development on the web app; it even picks up the relationships between different tables.

While working with mapping classes, it is sometimes necessary to force an update between the ORM and the database. This proved to be very helpful for finding problems with the database structure, as shown below:



```
nlsr33.supercp.com - PuTTY

a3a74cca0527a'; check that column/key exists

In PDOConnection.php line 106:

SQLSTATE[42000]: Syntax error or access violation: 1091 Can't DROP 'idx_f73
a3a74cca0527a'; check that column/key exists

In PDOConnection.php line 104:

SQLSTATE[42000]: Syntax error or access violation: 1091 Can't DROP 'idx_f73
a3a74cca0527a'; check that column/key exists

doctrine:schema:update [--complete] [--dump-sql] [-f|--force] [--em [EM]] [-h|--
help] [-q|--quiet] [-v|vv|vvv|--verbose] [-V|--version] [--ansi] [--no-ansi] [-n
|--no-interaction] [-e|--env ENV] [--no-debug] [--] <command>

phoenix9@nlsr33 [~/arcade.phoenixdragon.co.uk]# php bin/console doctrine:schema:
update --force
Updating database schema...
Database schema updated successfully! "8" queries were executed
phoenix9@nlsr33 [~/arcade.phoenixdragon.co.uk]#
```

Figure 6: Syncing Doctrine with the database is a good way to debug the database schema

Once the database was synced correctly, the next challenge was to retrieve the stored data and display it in a view. It turned out that Twig can be quite tricky to work with; Figure 7 shows what happened when I tried to print a simple array:

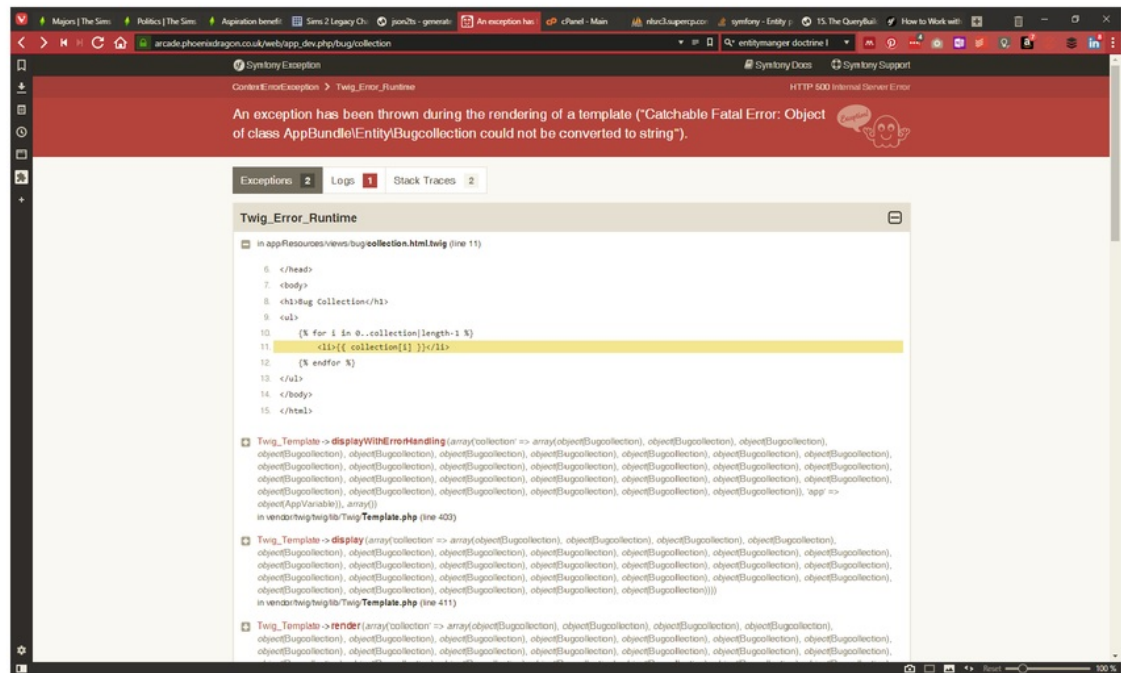


Figure 7: Initial problems getting Twig to render the view for BugCollection

During the initial debugging, I made use of a handy feature to dump the variable contents in Twig using the `dump(array)` syntax, as shown below:

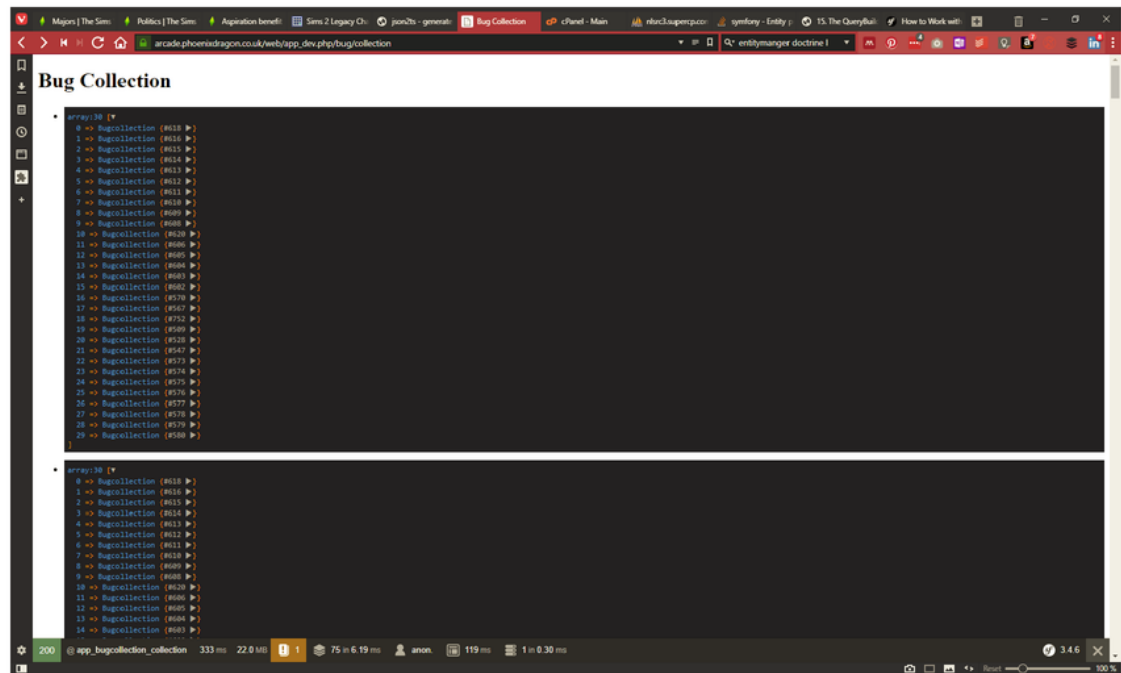


Figure 8: Using Twig's dump function to check the array is populated

Eventually, after reading more on how to use Twig[8], I managed to print the array contents successfully (shown below).

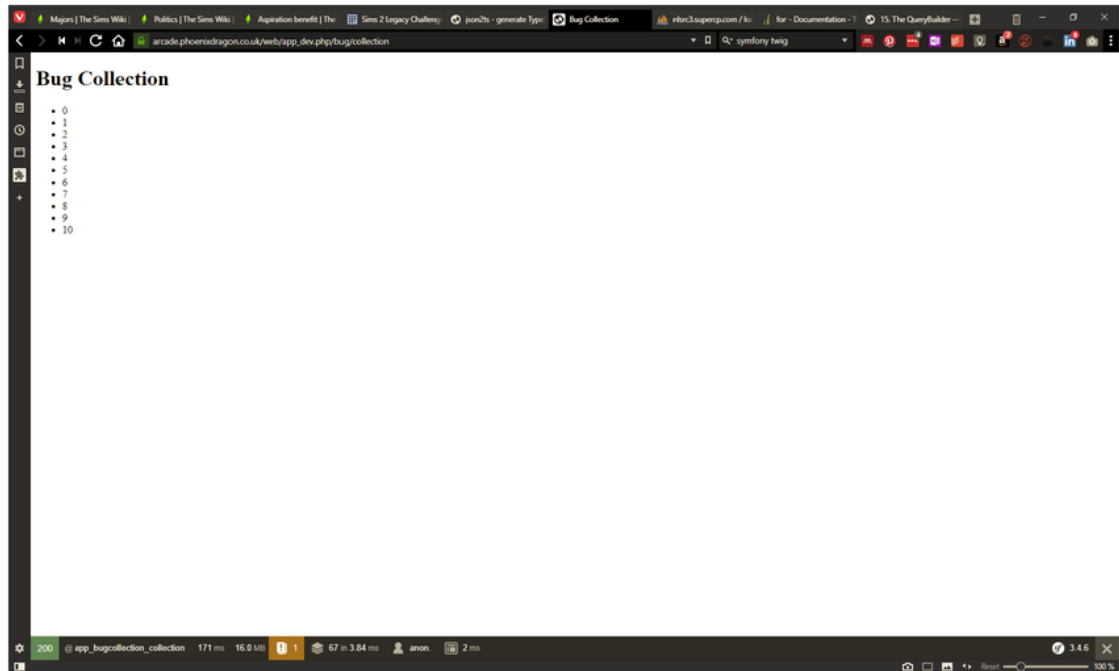


Figure 9: Successful printing of array

Changing the printing of an array to printing database results proved more difficult. Using suggested error-catching syntax (if [name]!=null), I was at least able to (eventually) restore the printing of an array:

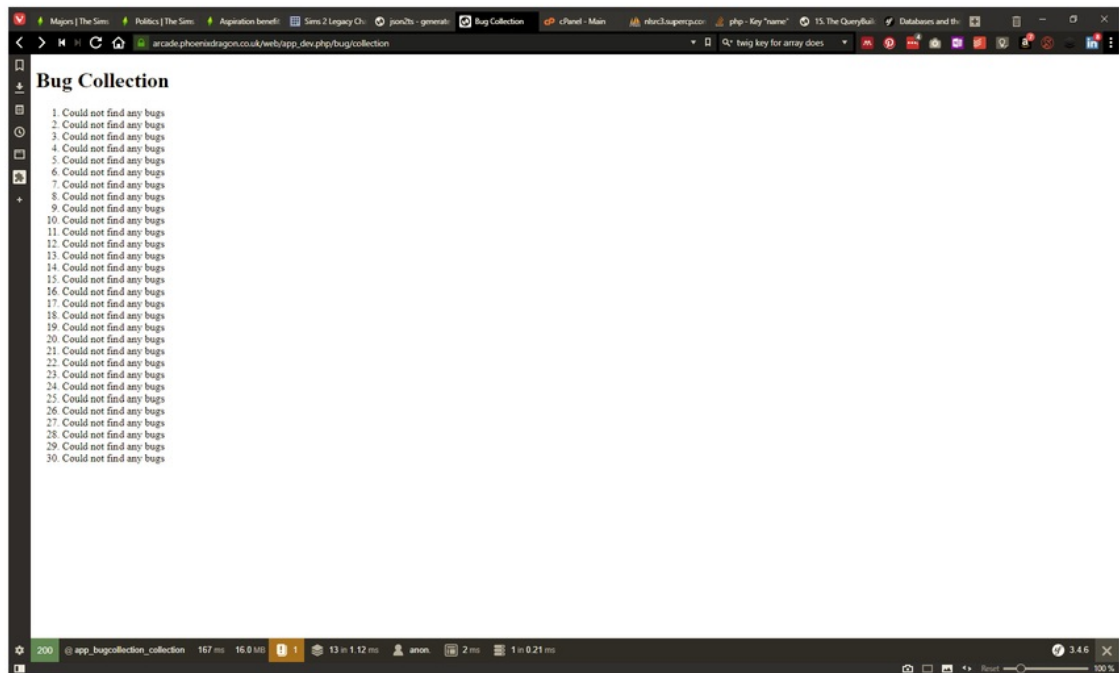


Figure 10: Trying to switch from array to database threw some errors

After some more searching for answers, I finally got it working:

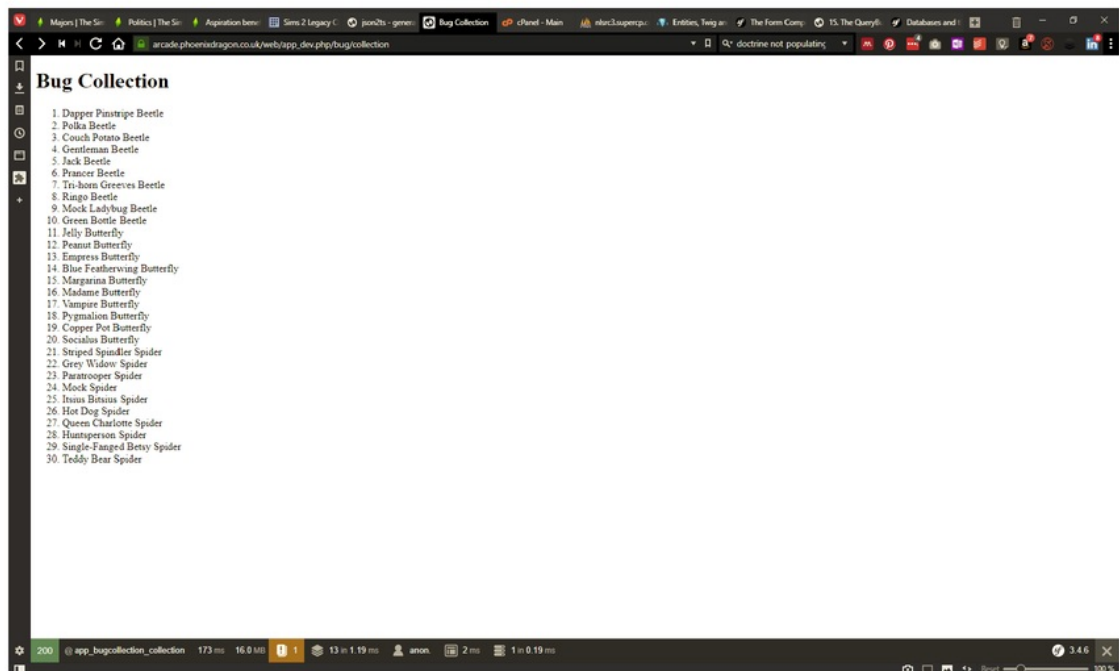


Figure 11: Successfully printing database contents

I decided to extend this by adding images to the database table, and changing the layout to tidy up the presentation:

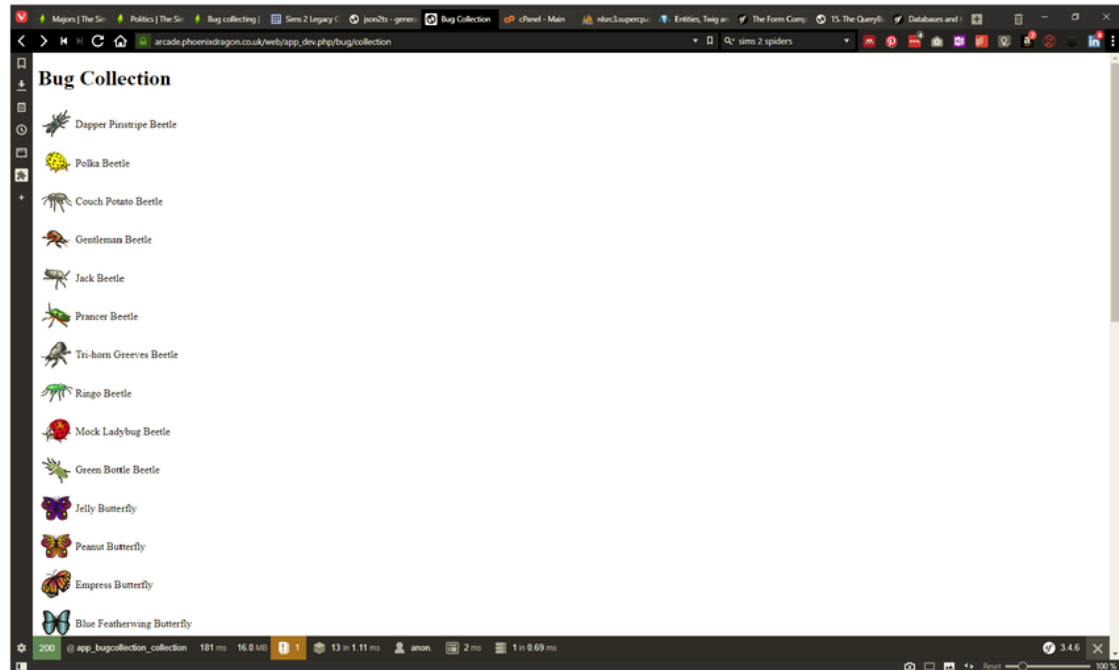


Figure 12: Successful test to also render images

Experimenting with other bundles (such as the FOSUserBundle mentioned above) came after I had successfully got the basics working – which was just as well, because trying to get this handy tool working broke the website.

Regarding testing, Symfony provides a built-in debugging system when running through the app.dev controller, as shown in the above images. Exception pages give detailed error messages, and there is a toolbar along the bottom of the page that enables the developer to check for more information, such as the number of database queries (and how long they take), whether the user is logged in, the status code (i.e. 200), etc.

Since Symfony supports MongoDB, it is possible to consider using NoSQL for database management – which may be necessary for large-scale applications. It also offers support for session handling via database rather than files, which goes back to the configuration concept, as much of this setup is done via the configuration files.

As well as spending a lot of time in the config directory, there is also much time spent in the SSH terminal, particularly if installing other modules. Even running Doctrine is done mostly in PuTTY rather than the PHP interface. While this is fine for developers who are comfortable working in SSH, it is not an ideal way to build a web app if you just want to get something up and running quickly.

III. References

- [1] Symfony, 'Symfony, High Performance PHP Framework for Web Development', *Symfony*, 2014. [Online]. Available: <https://symfony.com/>. [Accessed: 30-Mar-2018].
- [2] SensioCloud, 'Home - Twig - The flexible, fast, and secure PHP template engine', *SensioLabs*, 2017. [Online]. Available: <https://twig.symfony.com/>. [Accessed: 30-Mar-2018].
- [3] 'PHP MVC Frameworks preview of 2018 (Phalcon 3, Symfony 4, Laravel 5.x and others) - zen of coding'. [Online]. Available: <https://www.zenofcoding.com/2017/12/31/php-mvc-frameworks-preview-of-2018-phalcon-3-symfony-4-laravel-5-x-and-others/>. [Accessed: 30-Mar-2018].
- [4] Unknown, 'CakePHP Conventions', 2014. [Online]. Available: <https://book.cakephp.org/3.0/en/intro/conventions.html#model-and-database-conventions>. [Accessed: 26-Mar-2018].
- [5] G. Started, 'How to Implement a Simple Registration Form', pp. 1–7, 2016.
- [6] 'Authentication - Laravel - The PHP Framework For Web Artisans'. [Online]. Available: <https://laravel.com/docs/5.6/authentication>. [Accessed: 30-Mar-2018].
- [7] G. Started, 'How to Generate Entities from an Existing Database', pp. 1–4, 2016.
- [8] 'Entities, Twig and the Magic dot Syntax > Symfony 3: Doctrine & the Database | KnpUniversity'. [Online]. Available: <https://knpuniversity.com/screencast/symfony-doctrine/twig-dot-syntax>. [Accessed: 23-Mar-2018].

IV. Appendix I: Code

Below is the relevant code for the functioning test app, which was built using Symfony.

A. Model

1. SQL data for BugCollection table

```
-- phpMyAdmin SQL Dump
-- version 4.7.7
-- https://www.phpmyadmin.net/
--
-- Host: localhost:3306
-- Generation Time: Mar 25, 2018 at 09:46 PM
-- Server version: 5.6.23-cll-lve
-- PHP Version: 5.6.30

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Database: `phoenix9_simstracker`
--

--
-- -----
--
-- Table structure for table `BugCollection`
--

CREATE TABLE `BugCollection` (
  `BugID` int(11) NOT NULL,
  `BugIcon` varchar(80) DEFAULT NULL,
  `BugName` varchar(60) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

--
-- Dumping data for table `BugCollection`
--

INSERT INTO `BugCollection` (`BugID`, `BugIcon`, `BugName`) VALUES
(1, 'Dapper Pinstripe Beetle.png', 'Dapper Pinstripe Beetle'),
(2, 'Polka Beetle.png', 'Polka Beetle'),
(3, 'Couch Potato Beetle.png', 'Couch Potato Beetle'),
(4, 'Gentleman Beetle.png', 'Gentleman Beetle'),
(5, 'Jack Beetle.png', 'Jack Beetle'),
(6, 'Prancer Beetle.png', 'Prancer Beetle'),
(7, 'Tri-Horn Greeves Beetle.png', 'Tri-horn Greeves Beetle'),
(8, 'Ringo Beetle.png', 'Ringo Beetle'),
(9, 'Mock Ladybug Beetle.png', 'Mock Ladybug Beetle'),
(10, 'Green Bottle Beetle.png', 'Green Bottle Beetle'),
(11, 'Jelly Butterfly.png', 'Jelly Butterfly');
```

```

(12, 'Peanut_Butterfly.png', 'Peanut Butterfly'),
(13, 'Empress_Butterfly.png', 'Empress Butterfly'),
(14, 'Blue_Featherwing_Butterfly.png', 'Blue Featherwing Butterfly'),
(15, 'Margarina_Butterfly.png', 'Margarina Butterfly'),
(16, 'Madame_Butterfly.png', 'Madame Butterfly'),
(17, 'Vampire_Butterfly.png', 'Vampire Butterfly'),
(18, 'Pygmalion_Butterfly.png', 'Pygmalion Butterfly'),
(19, 'Copper_Pot_Butterfly.png', 'Copper Pot Butterfly'),
(20, 'Socialus_Butterfly.png', 'Socialus Butterfly'),
(21, NULL, 'Striped Spindler Spider'),
(22, NULL, 'Grey Widow Spider'),
(23, NULL, 'Paratrooper Spider'),
(24, NULL, 'Mock Spider'),
(25, NULL, 'Itsius Bitsius Spider'),
(26, NULL, 'Hot Dog Spider'),
(27, NULL, 'Queen Charlotte Spider'),
(28, NULL, 'Huntsman Spider'),
(29, NULL, 'Single-Fanged Betsy Spider'),
(30, NULL, 'Teddy Bear Spider');

--
-- Indexes for dumped tables
--

--
-- Indexes for table `BugCollection`
--
ALTER TABLE `BugCollection`
  ADD PRIMARY KEY (`BugID`),
  ADD KEY `BugID` (`BugID`);

--
-- AUTO_INCREMENT for dumped tables
--

--
-- AUTO_INCREMENT for table `BugCollection`
--
ALTER TABLE `BugCollection`
  MODIFY `BugID` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=31;
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;

```

2. BugCollection.php

```

<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Bugcollection
 *
 * @ORM\Table(name="BugCollection", indexes={@ORM\Index(name="BugID",
columns={"BugID"})})
 * @ORM\Entity
 */
class Bugcollection

```

```

{
    /**
     * @var integer
     *
     * @ORM\Column(name="BugID", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $bugid;

    /**
     * @var string
     *
     * @ORM\Column(name="BugIcon", type="string", length=80, nullable=true)
     */
    private $bugicon;

    /**
     * @var string
     *
     * @ORM\Column(name="BugName", type="string", length=60, nullable=false)
     */
    private $bugname;

    /**
     * @return int
     */
    public function getBugid()
    {
        return $this->bugid;
    }

    /**
     * @return string
     */
    public function getBugicon()
    {
        return $this->bugicon;
    }

    /**
     * @return string
     */
    public function getBugname()
    {
        return $this->bugname;
    }

    /**
     * @param int $bugid
     */
    public function setBugid($bugid)
    {
        $this->bugid = $bugid;
    }

    /**
     * @param string $bugicon
     */
    public function setBugicon($bugicon)
    {
        $this->bugicon = $bugicon;
    }
}

```

```

/**
 * @param string $bugname
 */
public function setBugname($bugname)
{
    $this->bugname = $bugname;
}
}

```

B. View

1. Collection.html.twig

```

<!DOCTYPE html>
<html lang="en-gb">
<head>
    <meta charset="UTF-8">
    <title>Bug Collection</title>
</head>
<body>
<h1>Bug Collection</h1>
<table>
    <thead>
        <tr>
            <th>
                Image
            </th>
            <th>
                Name
            </th>
        </tr>
    </thead>
    <tbody>
        {% for bug in collection %}
            <tr>
                {% if bug.bugicon is defined and bug.bugicon is not null %} <!-- Some bugs don't
have images -->
                    <td></td>
                {% else %}
                    <td></td>
                {% endif %}
                <td>{{ bug.bugname }}</td>
            </tr>
        {% endfor %}
    </tbody>
</table>

<!-- First test code -->
<ol>
    {% for bug in collection %}
        {% if bug.bugname is defined and bug.bugname is not null %}
            <li>{{ bug.bugname|e }}</li>
        {% else %}
            <li>Could not find any bug data</li>
        {% endif %}
    {% endfor %}
</ol>
</body>
</html>

```

C. Controller

1. BugCollectionController.php

```
<?php
/**
 * Created by PhpStorm.
 * User: Sam
 * Date: 22/03/2018
 * Time: 22:18
 */

namespace AppBundle\Controller;

use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use AppBundle\Entity\Bugcollection;

class BugCollectionController extends Controller
{
    /**
     * @Route("/bug/collection")
     */
    public function collectionAction()
    {
        // Retrieves a repository
        $collection = $this->getDoctrine()
            ->getRepository(Bugcollection::class)
            ->findAll();

        if (!$collection) {
            throw $this->createNotFoundException(
                'No items found '
            );
        }

        // ... do something, like pass the $product object into a template
        return $this->render('bug/collection.html.twig', array(
            'collection' => $collection,
        ));
    }
}
```

D. Config

1. Doctrine configuration from config.yml

```
# Doctrine Configuration
doctrine:
    dbal:
        driver: pdo_mysql
        host: '%database_host%'
        port: '%database_port%'
        dbname: '%database_name%'
        user: '%database_user%'
```

```
password: '%database_password%'
charset: utf8mb4
default_table_options:
  charset: utf8mb4
  collate: utf8mb4_general_ci

orm:
  auto_generate_proxy_classes: '%kernel.debug%'
  naming_strategy: doctrine.orm.naming_strategy.underscore
  auto_mapping: true
```

2. Database setup in parameters.yml

```
# This file is auto-generated during the composer install
parameters:
  database_host: localhost
  database_port: 3306
  database_name: phoenix9_simstracker
  database_user: phoenix9_tracker
  database_password: [redacted]
  mailer_transport: smtp
  mailer_host: localhost
  mailer_user: null
  mailer_password: null
  secret: ThisTokenIsNotSoSecretChangeIt
```

NB: This is linked to my web host account, so this all functions online.

Advance Web 2

GRADEMARK REPORT

FINAL GRADE

60/100

GENERAL COMMENTS

Instructor

Investigation of Fowler (30) 18

Investigation of MVC frameworks (20) 17

Quality of website and techniques used (25) 14

View handling (10) 5

Database handling (15) 6

The Fowler patterns and their benefits could have been discussed in a little more detail.

Implementation experimented with Symfony and a number of problems/solutions described. The resultant implementation is a little small but demonstrates a number of features.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22
