Summer Fellowship Report

On

Fossee Web Development

Submitted by

Fahad Israr

Under the guidance of

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

July 11, 2019

Acknowledgment

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

I owe my deep gratitude to our **Project Mentors: Mr.Nitish Kumar , Mr.Tejas Vaidya and Miss Ruchi Kumari**, who took keen interest on our project work and guided us all along, till the completion of our project work by providing all the necessary information for developing a good system.

I respect and thank **Dr. P. Sunthar**, for providing me an opportunity to do the project work in **Drupal** under his esteemed guidance and giving us all support and guidance which made me complete the project duly. I am extremely thankful to him for providing such a nice support and guidance.

I heartily thankful to other Mentors and **entire Fossee Team** for their guidance and suggestions during this project work.

I would not forget to remember my co-fellows Mansimran,Bhavika,Esha and Kalpesh for their encouragement and more over for their timely support and friendly encouragement till the completion of our project work.

I am thankful to and fortunate enough to get constant support and cooperation from **IIT Bombay Administration** which helped us in successfully completing our project work.Also, I would like to extend our sincere esteems to all the working staff of IIT B for their sincere effort and support.

Contents

Chapter 1

Introduction

A content management system (CMS) is a software tool that lets users add, publish, edit, or remove content from a website, using a web browser on a smartphone, tablet, or desktop computer. Typically, the CMS software is written in a scripting language, and its scripts run on a computer where a database and a web server are installed. The content and settings for the website are usually stored in a database, and for each page request that comes to the web server, the scripts combine information from the database and assets (JavaScript files, CSS files, image files, etc. that are part of the CMS or have been uploaded) to build the pages of the website.The combination of the operating system that the CMS runs on, the scripting language it is written in, the database it stores its information in, and the web server that runs the scripts to retrieve information and return it to the site visitors web browser is known as the stack that the CMS runs on; the commonly used combination of the Linux operating system, Apache web server, MySQL database, and PHP scripting language is known as the LAMP stack.

Drupal is a flexible CMS based on the LAMP stack, with a modular design allowing features to be added and removed by installing and uninstalling modules, and allowing the entire look and feel of the website to be changed by installing and uninstalling themes. The base Drupal download, known as Drupal Core, contains the PHP scripts needed to run the basic CMS functionality, several optional modules and themes, and many JavaScript, CSS, and image assets. Many additional modules and themes can be downloaded from the Drupal.org website.

Our project focuses centrally on Distributions,Custom Modules and Migrations in Drupal CMS.We tend to employ drupal modules to generate the desired configurations and also contribute our work globally on Open Source via Github.

Chapter 2

Preface to Drupal

Drupal is a free, open-source content management system (CMS) with a large, supportive community.The Drupal slogan is "Come for the code, stay for the community." You can download the software for free and do what you like with it. There are also tens of thousands of people around the world who come together to improve the code, write documentation, run events, and support each other.

2.1 Reasons for using Drupal

When building a website, you have your choice of using one of the many existing CMS packages and hosted services, developing your own CMS, or building the site without using a CMS. Here are some of the reasons you might choose to use Drupal:

- Building a small, simple site with static HTML pages is not difficult, and you can get a simple site up very quickly. Setting up a site in a CMS generally requires more time initially, but brings you the benefits of on-line editing (easier for less experienced content maintainers), uniformity (harder to maintain using static HTML for larger sites), and the possibility of more complex features requiring a database.

- Some CMS software is special-purpose; for instance, there are packages and hosted services that you can use to build a blog or a club membership website. Drupal, in contrast, is a general-purpose CMS. If you are building a special-purpose site, you might choose to use a special-purpose CMS; however, if your site falls even slightly outside the intended purpose, you will probably be better off using a general-purpose CMS rather than trying to adapt a special-purpose CMS.

- Building your own CMS-type software can seem attractive. However, using a general-purpose CMS like Drupal as a starting point is usually a better idea, because the basic CMS functionality (such as user accounts and content management) has thousands of developer hours behind it, including many years of user testing, bug fixing, and security hardening.

- Some CMS software packages are expensive to purchase a license for. Some are free or have a free version, but have restrictive licenses that do not allow you to make modifications and extensions. You might prefer to use a package (like Drupal) that has a less restrictive software license, and is developed by a world-wide community.

- Flexibility:One of the major USP's of Drupal is its ability to create and manage a wide variety of content types, including but not limited to videos, polls, blogs, podcasts, and statistics. Because of this feature, Drupal enables a flexible design platform to create content-rich websites for a variety of different markets like media or commerce. The script also includes capabilities of design elements editing, which makes it easy to create both simple and complicated web page configurations.

- Customizability: In addition to being flexible, Drupal is also highly customizable. Boasting over 16,000 modules and plug-ins, Drupal allows you to modify, adjust and implement an endless wealth of additional custom features into your website like CRM, security, social media and SEO.

- Scalibility:Another major strength of Drupal is that its tremendously scalable. You can exponentially grow the number of your web pages without changing a thing. Because of this, Drupal is great at accommodating content growth. Its also great at alternating between periods of constant traffic and high traffic spikes, which is why it's used by weather.com and whitehouse.gov.

- Security:Drupal also offers unshakable security. As of 2015, the annual Drupal security report notes that there have been no widely exploited vulnerabilities in Drupal core for which there was no patch or upgrade available at the time of public disclosure." The closest known instance was an exploit in a common XML-RPC code library that was in use in early versions of Drupal. All vendors using the library were at risk, but no wide attack was known to be in play. In short, the CMS is highly secure and offers regular patches and safeguarding from exploits, making it great for enterprise clients.

- Community:Drupal is an open source CMS. Aside from being a licensing option, open source is really a culture and an approach to technology that revolves around the free exchange of ideas and innovation. The open source community offers extensive public documentation, well-developed discussion boards, chat and mailing lists, alongside an air of approachable online culture.

- Besides,Drupal can also run on other technology stacks:

  - The operating system can be Windows or Mac OS instead of Linux.
  - The web server can be Nginx or IIS instead of Apache.
  - The database can be PostgreSQL or SQLite instead of MySQL, or a MySQL-compatible replacement such as MariaDB or Percona.

2.2 The Drupal Architecture

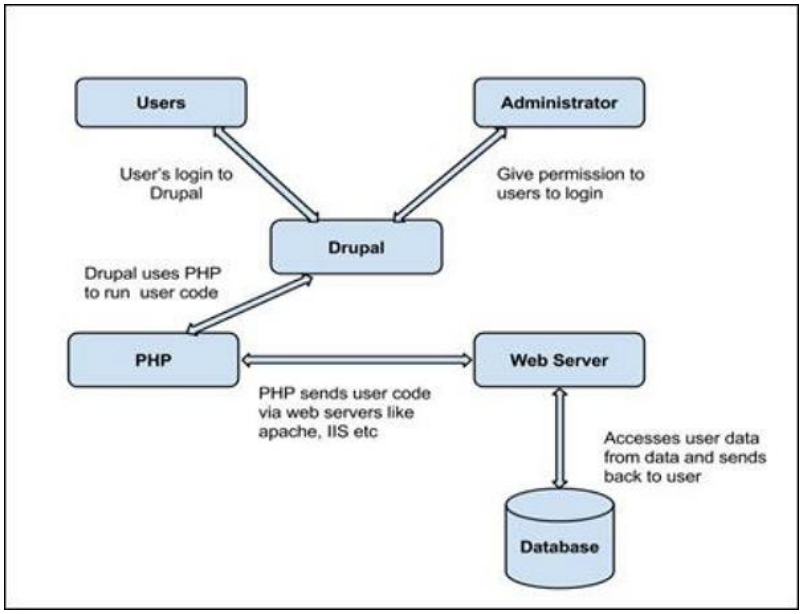

Figure 2.1: The Drupal Architecture

The architecture of Drupal contains the following layers and entities:

- **Users:** These are the users on the Drupal community. The user sends a request to a server using Drupal CMS and web browsers, search engines, etc. acts like clients.
- **Administrator:** Administrator can provide access permission to authorized users and will be able to block unauthorized access. Administrative account will be having all privileges for managing content and administering the site.
- **PHP:** Drupal uses PHP in order to work with an application which is created by a user. It takes the help of web server to fetch data from the database. PHP memory requirements depend on the modules which are used in your site. Drupal 6 requires at least 16MB, Drupal 7 requires 32MB and Drupal 8 requires 64MB.
- **Web Server:** Web server is a server where the user interacts and processes requests via HTTP (Hyper Text Transfer Protocol) and serves files that form web pages to web users. The communication between the user and the server takes place using HTTP. You can use different types of web servers such as Apache, IIS, Nginx, Lighttpd, etc.
- **Database:** Database stores the user information, content and other required data of the site. It is used to store the administrative information to manage the Drupal site. Drupal uses the database to extract the data and enables to store, modify and update the database.

2.3 Drupal Terminology

2.3.1 Modules

A module is a set of PHP, JavaScript, and/or CSS files that extends site features and adds functionality. You can turn the features and functionality on by installing the module, and you can turn it off by uninstalling the module; before uninstalling, you may need to remove data and configuration related to the feature or functionality. Each module that is installed adds to the time needed to generate pages on your site, so it is a good idea to uninstall modules that are not needed. The core download provides modules for functionality such as:

- Managing user accounts (the core User module)
- Managing basic content (the core Node module) and fields (the core Field and Field UI
- modules; there are also core modules providing field types)
- Managing navigation menus (the core Menu UI module)
- Making lists, grids, and blocks from existing content (the core Views and Views UI modules)

A Drupal site can have three kinds of modules:

- **Core modules** that ship with Drupal and are approved by the core developers and the community.
- **Contributed modules** written by the Drupal community and shared under the same GNU Public License (GPL) as Drupal.
- **Custom modules** created by the developer often for a particular use case specific to the site they're working on.

2.3.2 Distributions

Distributions are full copies of Drupal that include Drupal Core, along with additional software such as themes, modules, libraries, and installation profiles. There are two main types of Drupal distributions:

- Full-featured distributions: complete solutions for specialized use cases.
- Other distributions: quick-start tools, starting points for developers and site builders.

2.3.3 Content Structure Glossary

- **Content types:** content types are one of the main building blocks within a Drupal site; as the name suggests, content types hold content. However, different content types can hold different kinds of content; an event can hold information that is specific to a time, where a discussion can be used for people to talk. Most sites have multiple different content types, and the name of a content type will ideally give information about how it is used. Node: within Drupal, a node is a piece of content. All data stored via a content type is a node.

- **Taxonomy:** Drupal's taxonomy system is used to categorize information. It is a general term that is used to describe how things are categorized. Vocabulary: a vocabulary is a specific, high level subject area. Each vocabulary consists of multiple terms (see below). For example, an example of two vocabularies used in a news site could be "Subject" and "Region".

- **Term:** a term is an individual topic within a vocabulary. For example, the terms "Texas" and "The rest of the United States" could be in the "Region" vocabulary. Menus: collections of links; these links can be displayed as a list, as drop-down items, with graphics, etc, depending on how they are styled by the theme.

- **Blocks:** Blocks contain and display a variety of information on a Drupal site. They can be created in a variety of ways, and provide a range of options for displaying and theming content.

- **Entity:** an entity is a piece of data within a Drupal site. Nodes, users, comments, and taxonomy are all entities; additionally, with custom code you can create new entity types if/when needed. You can also add fields to entities, which allows for things like detailed user profiles, or more sophisticated comment forms.

- **Fields:** fields are used to store and display structured information. For example, on a user profile, you would want to create a "First" and "Last" name field to store normalized data; or, you would break an address down into individual fields to store the components of an address. There are also different types of fields; for example, things as varied as email addresses and pictures can be stored within fields, and this allows us to make some assumptions about the information stored in the field.

- **Bundle:** a bundle is an entity and all its fields.

- **Core:** Drupal core contains the central codebase of Drupal. Each component of core has a dedicated maintainer; in general, core is the base upon which everything else gets built. Within versions (6.x, 7.x, 8.x, etc) the structure of core will remain relatively unchanged.

Chapter 3

Migrations

Migration is the process of import/export of Drupal data such as **node, user or taxonomy term** The Migrate API provides services for migrating data from a source system to Drupal 8.

3.1 Migrate Modules

- Drupal 8 core Migrate module implements the general-purpose framework.
- Drupal 8 core Migrate Drupal module builds on that foundation to provide an upgrade path from Drupal 6 and Drupal 7 to Drupal 8.
- Drupal 8 core Migrate Drupal UI module provides a browser user interface for Migrate Drupal.
- Migrations can be executed with different tools :Executing migrations from non-Drupal sources require contributed modules that work together with the core Migrate API.

3.2 Migrations as Extract - Transform - Load (ETL) processes

In computing, extract, transform, load (ETL) is the general procedure of copying data from one or more sources into a destination system which represents the data differently from the source(s) or in a different context than the source(s). The ETL process became a popular concept in the 1970s and is often used in data warehousing[1].
Data extraction involves extracting data from homogeneous or heterogeneous sources; data transformation processes data by data cleansing and transforming them into a proper storage format/structure for the purposes of querying and analysis; finally, data loading describes the insertion of data into the final target database such as an operational data store, a data mart, data lake or a data warehouse.[2][3]

A properly designed ETL system extracts data from the source systems, enforces data quality and consistency standards, conforms data so that separate sources can be used together, and finally delivers data in a presentation-ready format so that application developers can build applications and end users can make decisions.[4] Since the data extraction takes time, it is common to execute the three phases in parallel. While the data is being extracted, another transformation process executes while processing the data already received and prepares it for loading while the data loading begins without waiting for the completion of the previous phases.
ETL systems commonly integrate data from multiple applications (systems), typically developed and supported by different vendors or hosted on separate computer hardware. The separate systems containing the original data are frequently managed and operated by different employees. For example, a cost accounting system may combine data from payroll, sales, and purchasing.
Migration in Drupal is an Extract, Transform, Load (ETL) process.

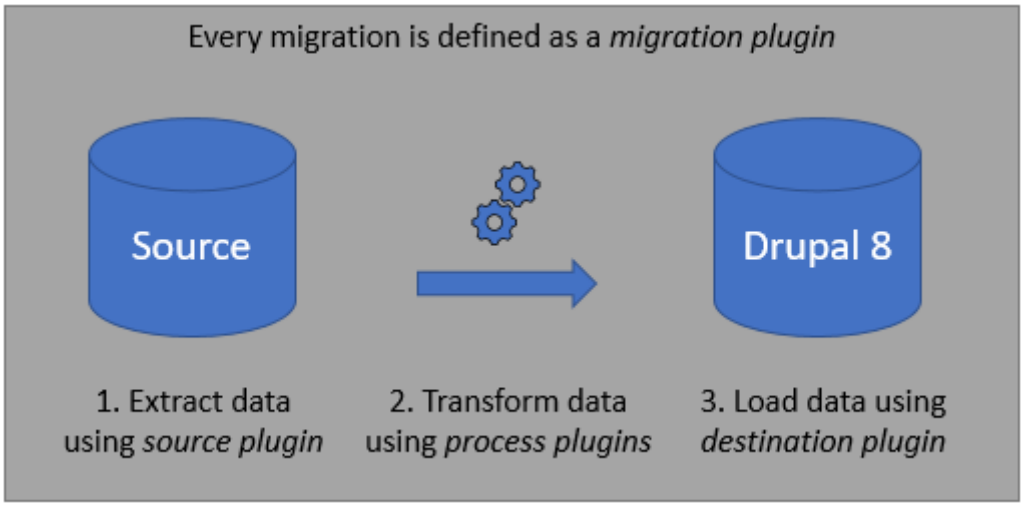

Figure 3.1: Migration in Drupal

- **Extract phase is called Source**
- **Transform phase is called Process**
- **Load phase is called Destination**

It is important to understand that the term load in ETL means to load data into the storage, while in a typical Drupal context the term load refers to loading data from storage.
In the source phase, a set of data, called the row, is retrieved from the data source. The data can be migrated from a database, loaded from a file (for example CSV, JSON or XML) or fetched from a web service (for example RSS or REST). The row is sent to the process phase where it is transformed as needed or marked to be skipped. After processing, the transformed row is passed to the destination phase where it is loaded (saved) into the target Drupal site.

3.3 Migrate API plugins

Migration plugins specify individual ETL migrations, such as node, user or taxonomy term migration.
Migration plugins are defined in YAML format.

- **Source plugins** extract the data from the source.
- **Process plugins** transform the data.
- **Destination plugins** save the data to Drupal 8.

3.4 Executing migrations

3.4.1 Importing migration YAML definitions

The contributed Migrate Plus module allows migration plugins to be implemented as configuration entities, allowing them to flexibly be loaded, modified, and saved.

- Make sure that core Migrate and contributed Migrate Plus modules are enabled.
- Navigate to admin/config/development/configuration/single/import of your Drupal 8 site.
- Select 'Migration' as the configuration type.
- Paste your migration definition YAML to the configuration import form and click 'Import'.

3.4.2 Checking migration status

The contributed Migrate Tools provides drush migrate-status Drush command that you can execute on the command line of our server. If you don't see the migration:

- verify that you have enabled your custom module (if you are using a custom module that provides Migrate API source / process / destination plugins)
- verify that your custom source / process / destination plugins are located in src/Plugin/migrate/source, src/Plugin/migrate/process, src/Plugin/migrate/destination directories
- if you are using an SQL source plugin, verify that you have defined the source database connection in your settings.php or settings.local.php and that the database connection parameters are correct.

3.4.3 Executing the migration

- **Using Drush and Migrate Tools:**
  The contributed Migrate Tools provides 'migrate-import' and 'migrate-rollback' Drush commands. The command below migrates 10 rows from a migration with id 'games' so that you can verify the results on your Drupal 8 site. If you wish to migrate all rows from the source, leave out the 'limit' argument.

```
$ drush migrate-import games --limit=10
```

- **Using Drush and Migrate Manifest:**
  The contributed Migrate Manifest module allows you to run a group of migrations in a reproducible manner and in the correct order based on the migration.

3.4.4 Rollback the migration

After you have executed the migration, verify the result on your Drupal 8 site. If you find out that you need to modify the migration a bit, you can roll back the migration with the 'migrate-rollback' Drush command. Also this Drush command is provided by the contributed Migrate Tools module.

- Using Drush

```
$ drush migrate-rollback games
```

3.4.5 Executing the migration on schedule:

The contributed module Migrate Scheduler provides a way to execute the migrations on a configurable schedule. It also provides an option to execute the migrations with an –update flag.

3.4.6 Modifying the migration definition:

There are different ways how the already imported YAML format migration definitions can be updated.

- Method 1:
  - First export the current version of the migration definition at admin/config/development/configuration/single/export
  - Copy the migration definition to a text editor and modify it as needed
  - Import the modified migration definition at admin/config/development/configuration/single/import
  - After configuration refresh; refresh the migration record via its originating configuration.

```
$ drush migrate-status
```

- Method 2:
  - It is possible to modify the configuration using the Drupal Console command line tool:

```
$ drupal config:edit migrate_plus.migration.games
```

  - You will need to run drush cr to rebuild the cache before the import is executed again.

```
$ drush cr
```

  - After configuration refresh; refresh the migration record via its originating configuration.

```
$ drush migrate-status
```

3.5 Stubs

Taxonomy terms are an example of a data structure where an entity can have a reference to a parent. When a term is being migrated, it is possible that its parent term has not yet been migrated. Migrate API addresses this 'chicken or the egg' dilemma by creating a stub term for the parent so that the child term can establish a reference to it. When the parent term is eventually being migrated, Migrate API updates the previously created stub with the actual content.

3.6 Our work with Drupal Migration:

Analysing our College Websites ,we tend to chalk out a methodology to realise the same using Drupal. For that we identified Content Types,Taxonomies and Basic Pages.Following that we created the identified content types and then we would add content to our Distribution using Migration.So we created a Configuration as a yml file for Content to be imported.Finally we created Modules that could be reused by one else.So we exported our Generated Modules,Tested them and Uploaded on GIthub.We made an open Source Contribution for Drupal.

We wrote configurations for importing various content entities like multiple image, entity reference,taxonomy term,files,paragraphs and many more.

3.6.1 Our Methodology:

- Download and enable following modules: Migrate Source CSV , Migrate Plus, Migrate Tools.
- Create csv file with the intended data to be imported.
- The contributed Migrate Plus module allows migration plugins to be implemented as configuration entities, allowing them to flexibly be loaded, modified, and saved. Navigate to Administration - Configuration - Development - Synchronize (admin/config/development/configuration/single/import)

- select Migration as the Configuration type.
- Insert the YAML format migration definition.
- The contributed Migrate Tools provides 'migrate-import' Drush command that you can execute on the command line of Drupal server:

```
$ drush migrate-import article_csv_import
```

Enlisting all our migration definitions wouldn't be feasible.So here are few of them:

3.6.2 Migrating Text,Entity Reference and File :

```
langcode: en
status: true
dependencies: {  }
id: 'FACULTY MIGRATIONS'
class: null
field_plugin_method: null
cck_plugin_method: null
migration_tags: null
migration_group: null
label: 'Custom Content migration from CSV faculty'
source:
  plugin: csv
  path: /var/www/html/project/docroot/faculty.csv
  header_row_count: 1
  keys:
    - id
  constants:
    file_source: /var/www/html/project/docroot/fac_images
    file_dest: 'public://fac_images/'
process:
  type:
    plugin: default_value
    default_value: fac_page
  source_path:
    -
      plugin: skip_on_empty
      method: process
      source: phuto
    -
      plugin: concat
      delimiter: /
      source:
        - constants/file_source
        - phuto
  title: title
  field_email_new: email
  field_full_name: name
  field_faculty_photo:
    plugin: file_import
    source: '@source_path'
    destination: constants/file_dest
  field_department:
    -
      plugin: entity_lookup
      entity_type: node
      bundle: department
      source: entity_ref
destination:
  plugin: 'entity:node'
migration_dependencies:
  required: {  }
  optional: {  }
```

3.6.3 Migrating Taxonomy Term and Image:

```
langcode: en
status: true
dependencies: {  }
id: hostel_mess
class: null
field_plugin_method: null
cck_plugin_method: null
migration_tags: null
migration_group: null
label: 'Custom Content migration from CSV for Hostel and Mess'
source:
  plugin: csv
  path: /var/www/html/project/docroot/article.csv
  header_row_count: 1
  keys:
    - id
  constants:
    file_source: /var/www/html/project/docroot/fac_images
    file_dest: 'public://fac_images/'
process:
  type:
    plugin: default_value
    default_value: article
  source_path:
    -
      plugin: skip_on_empty
      method: process
      source: phuto
    -
      plugin: concat
      delimiter: /
      source:
        - constants/file_source
        - phuto
  title: title
  body: body
  field_article_link: link
  field_image:
    plugin: file_import
    source: '@source_path'
    destination: constants/file_dest
  field_tags:
    -
      plugin: entity_lookup
      entity_type: taxonomy_term
      bundle: tags
      source: tags
destination:
  plugin: 'entity:node'
migration_dependencies:
  required: {  }
  optional: {  }
```

3.6.4 Migrating Basic Page

```
langcode: en
status: true
dependencies: {  }
id: basic_pages
class: null
field_plugin_method: null
cck_plugin_method: null
migration_tags: null
migration_group: null
label: 'Custom Content migration from CSV Trail'
source:
  plugin: csv
  path: /var/www/html/project/docroot/basic_page.csv
  header_row_count: 1
  keys:
    - id
process:
  type:
    plugin: default_value
    default_value: page
  title: title
  body: body
destination:
  plugin: 'entity:node'
migration_dependencies:
  required: {  }
  optional: {  }
```

Also the generated YAML Configuration files have been included with our modules to help easy migration of files.

Chapter 4

Form alter,View Creation and Exporting Modules

4.1 Hook Form alter

Perform alterations before a form is rendered.

One popular use of this hook is to add form elements to the node form. When altering a node form, the node entity can be retrieved by invoking

```
$form_state->getFormObject()->getEntity().
```

Implementations are responsible for adding cache contexts-tags- max-age as needed. The call order is as follows: all existing form alter functions are called for module A, then all for module B, etc., followed by all for any base theme(s), and finally for the theme itself. The module order is determined by system weight, then by module name.

Within each module, form alter hooks are called in the following order: first, hook_ form_ alter second, hook_ form_ BASE_ FORM_ ID_ alter third, hook_ form_ FORM_ ID_ alter. So, for each module, the more general hooks are called first followed by the more specific.

Parameters:

- **form:** Nested array of form elements that comprise the form.
- **form_ state:** The current state of the form.
- **form_ id:** String representing the name of the form itself. Typically this is the name of the function that generated the form.

File:

```
core/lib/Drupal/Core/Form/form.api.php, line 201
```

Code:

```
function hook_form_alter(&$form, \Drupal\Core\Form\FormStateInterface  $form_state,

  if (isset($form['type']) && $form['type']['#value'] . '_node_settings' == $form_id
  {
    $upload_enabled_types = \Drupal::config('mymodule.settings')
      ->get('upload_enabled_types');
    $form['workflow']['upload_' . $form['type']['#value']] = array(
      '#type' => 'radios',
      '#title' => t('Attachments'),
      '#default_value' => in_array($form['type']['#value'], $upload_enabled_types) ?
      '#options' => array(
        t('Disabled'),
        t('Enabled'),
      ),
    );

    // Add a custom submit handler to save the array of types back to the config fil
    $form['actions']['submit']['#submit'][] = 'mymodule_upload_enabled_types_submit'
  }
}
```

We used hook form later to modify the content addition form and Modify basic site settings viz. Site name,Email etc.

4.2 View Creation

After Creation of Content Types we created Views for each content type.

A view is a listing of content on a website. The core Views module handles the display of views, and the core Views UI module allows you to create and edit them in the administrative interface. When you define views, you are interested in taking data from your website and displaying it to the user.

The ways data can be output using views:

- Table with sortable fields
- Grid layouts
- Teasers or pictures that link to articles
- Blocks
- JSON output
- RSS feeds
- Calendars
- On-screen slideshows

We created Views for all the identified content types.For each of the views a link was included in the Main Navigation.The Navigation had several collapsible items.They would expand on hover.Menu were organized as menu and sub menu as present in the college website. The next Page displays a glimpse of the Home Page with all the Menu.

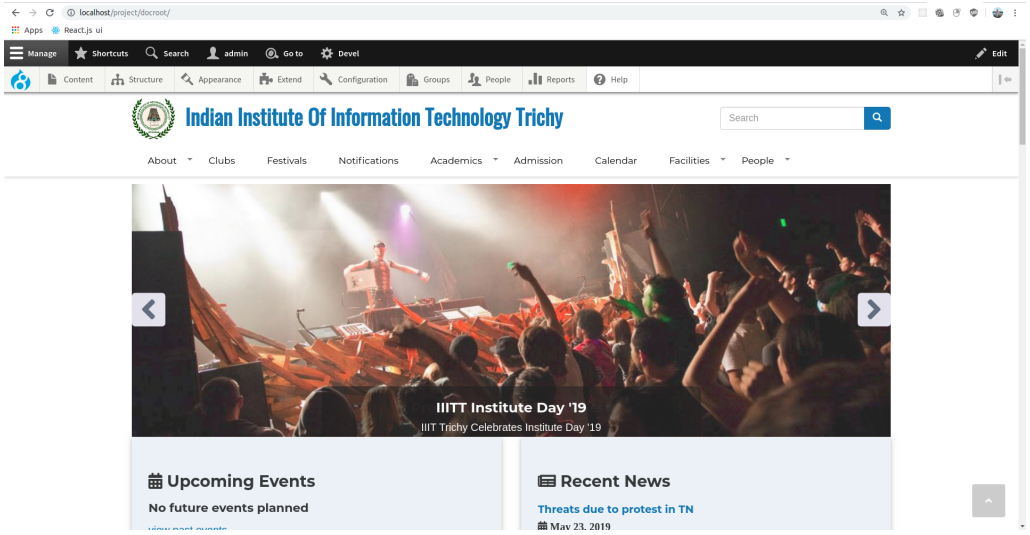

Figure 4.1: Home Page with Navigation Menu

4.3 Exporting Custom Modules

- To **generate** custom modules we need to first **install Drupal console:**

```
$ cd ~

$ curl https://drupalconsole.com/installer -L -o drupal.phar

$ mv drupal.phar /usr/local/bin/drupal

$ chmod +x /usr/local/bin/drupal

$ drupal self-update
```

- **To generate module:**

```
$ drupal generate:module
```

- **To export module we use:**

```
$drupal config:export:content:type content_type_name --module="module_name"
```

  *All names are **machine_ names**

- Then add required dependencies to the obtained modules in their config folder.
- Also add dependecy in info.yml files.
- **Test modules** on another system by installing them.

```
$ drush en [module_name]
```

4.4 Deploying on Github

After Creating Modules Push them on Github.

- Firstly,Create a new Repository on Github.
- Next,Click on clone and copy the address.
- Navigate to local directory where you have Custom Modules.
- Initiatize Empty git here.

```
$ git init
```

  Add Origin

```
$ git remote add origin https://github.com/user_name/repository_name
```

- Push our Modules:

```
$ git add .

$ git commit -m"commit_message_optional"

$ git push origin master
```

- Add Documentation to our Repository about:.
  - Installing our Modules
  - Testing our Modules
  - Contributing to the Modules

  Find my git repo at:
  https://github.com/fahad-israr/contributed-drupal-module

  The next figure shows my Modules Readme file after deployment.

https://github.com/fahad-israr/contributed-drupal-module

README.md

Custom Modules Developed for Drupal 8

--Pre Requisites:

--**Dupal Console Or Drush**

--To Install the Module(s):

--Place the required Content Type Module(s) in The Custom Folder of Modules of your Drupal Directory**(./modules/custom)**

--Intsalling with Drupal Console :

-->In the Terminal navigate to root directory of your Project.

-->Run the Command:**$ drupal modules:install module_name.**

-->Example if module is 'artcle' then run: $ drupal module:install article.

--Installing with Drush :

-->In the Terminal navigate to root directory of your Project.

-->Run the Command:**$ drush en module_name**

-->Example if module is 'artcle' then run: $ drush en article

--I'd love it if you could Contribute to my Modules:

-->**Fork** my Repository: https://github.com/fahad-israr/contributed-drupal-module

-->Work with your Repo and commit to your Repository.

-->Create a Pull Request.

-->We'll test your changes and Merge it.

Figure 4.2: Readme.md file of My Modules after Deployment on Github

Chapter 5

Headless Drupal: Progressive Web App with Drupal

Headless Drupal is basically a Drupal without Front End.You can use your own frontend framework with the Backend api from Drupal.The systems are entirely separate and communicate via HTTP example REST.

5.1 Setting Up our Back-End with REST

- Firstly,go to Modules of your Drupal UI.Then go to Extend.
- Now Enable All Web Services
  - RESTful Web Service
  - Serialization
- Install and Enable REST UI Module
- Create a Content type and Add content to them and then create a View for it.
- Go to /admin/config/services/rest.
- Select content type and set format json, auth cookie in the GET method. In this case, we are only going to download data, so the GET method will be perfectly fine.
- Go to view you created in above steps and just add '?_format=json' to the address.For Example my final address was http://localhost/drupalmod2/festivals?_format=json
- Go to address and you will see JSON object.If that appears then we are almost done with Backend.
- Enable CORS:You wont be able to access until you set up CORS.To do that your project root directory then site/default/default.services.yml
- Modify Cors.config as :

```
cors.config:

    enabled: true

    # Specify allowed headers, like 'x-allowed-header'.
    allowedHeaders: ['x-csrf-token','authorization','content-type','accept','o

    # Specify allowed request methods, specify ['*'] to allow all possible one
    allowedMethods: ['*']

    # Configure requests allowed from specific origins.
    allowedOrigins: ['http://localhost/','http://localhost:3000','http://local

    # Sets the Access-Control-Expose-Headers header.
    exposedHeaders: true

    # Sets the Access-Control-Max-Age header.
    maxAge: false

    # Sets the Access-Control-Allow-Credentials header.
    supportsCredentials: true
```

- Save the file as services.yml
- We have created the end point for Drupal. Lets Create our front End.

5.2 Developing Front-End React JS

React Js is A Popular Open source JavaScript library for building user interfaces developed by Facebook. It is declarative, efficient, and flexible.

- To get Started we first install NODE JS:

```
$ sudo apt-get install -y nodejs
```

- Now install create-react-app:

```
$ npm install -g create-react-app
```

- Now create a react app :

```
$ create-react-app Headless_Drupal
```

- Now go into the folder by: cd Headless_Drupal
- Run the following Command to start dev server:

```
$ npm start
```

- Now Edit App.js file:

```
import React from 'react';
import logo from './logo.svg';
import './App.css';

export default class App extends React.Component {



  constructor(props) {
    super(props);
    this.state = {
      festivals: null ,
      fetched:false,
    };
  }

  fetchFestivals=()=>{

    fetch('http://localhost/drupalmod2/festivals?_format=json', {
          method: 'get',
          headers: {'Content-Type': 'application/json'}
      }).then(response=>response.json()).then(data=>{if(data)this.setState({fe
    }

    componentDidMount=()=>{
      this.fetchFestivals();

    }
  render(){
    console.log(this.state.festivals)
  return (
    <div className="App">
      <h1 style={{fontSize:'50px'}}>A PWA with Headless Drupal</h1>
       <h1>Festivals</h1>
      {this.state.fetched&&this.state.festivals&&this.state.festivals.map((ite
        return(<div key={item.nid[0].value} className='festival'>
        <h2>{item.title[0].value}</h2>
        <img src={item.field_festival_image[0].url} />
        {item.body[0].value.substring(3,(item.body[0].value.length-12))}
        </div>)
      })}

    </div>
  )
}
}
```

- Note that you need to change the address according to your own address.
- Here we've simply developed a Single Page PWA with contents from BackEnd.
- The upcoming Image shows the app in action with console printing all the content Received from Drupal End Point.

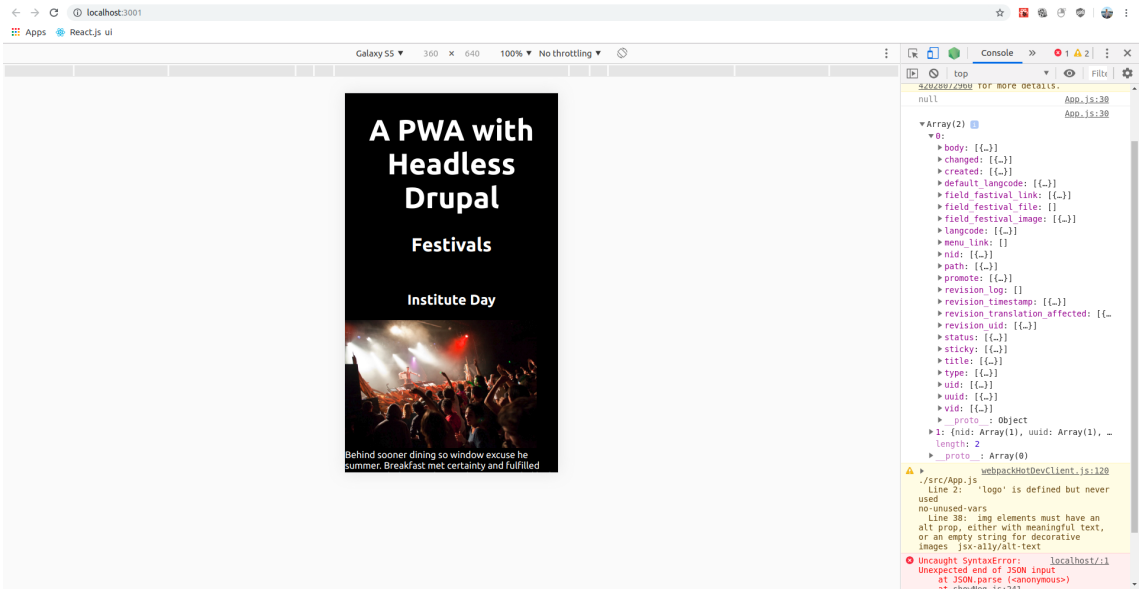

Figure 5.1: PWA in Action

View the Sorce on Github at: https://github.com/fahad-israr/PWA-with-Drupal-and-React-Js

The front End and Back End are in Separate Directories.Follow Appropriate instructions from Readme.md file to set up the App on your Local System.

Reference

- Drupal.org https://www.drupal.org/
- Drupalize.me https://drupalize.me/
- Acquia Docs https://docs.acquia.com
- BeFused https://befused.com/drupal