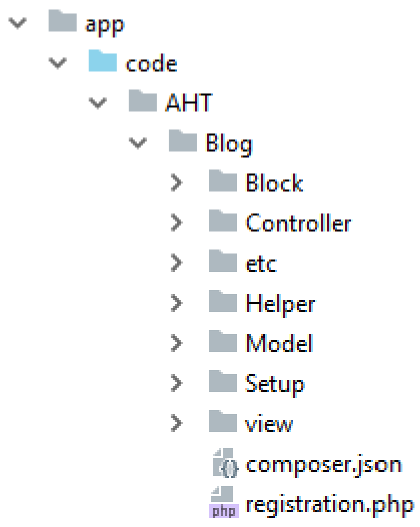
Structure of module Magento 2



```
app
  code
    AHT
      Blog
        Block
        Controller
        etc
        Helper
        Model
        Setup
        view
        composer.json
        registration.php
```

- AHT - it is a vendor's name (as a rule it coincides with the company name and should be unique).
- Blog: module (extension) name.
- Block: contains PHP view classes as part of Model View Controller(MVC) vertical implementation of module logic.
- Controller: contains PHP controller classes as part of MVC vertical implementation of module logic.
- etc: contains configuration files; in particular, module.xml, which is required.
- Helper: contains the files that are responsible for performing general tasks for extension objects and variables.
- Model: contains PHP model classes as part of MVC vertical implementation of module logic.
- Setup: includes the files that are necessary to make changes in the database – i.e. creating tables, fields, or other records required for the module performance (optional directory).
- View: contains view files, including static view files, design templates, email templates, and layout files that are necessary for information output on the frontend and backend.
- registration.php: Among other things, this file specifies the directory in which the component is installed by vendors in production environments. By default, composer automatically installs components in the <Magento root dir>/vendor directory.
- etc/module.xml: This file specifies basic information about the component such as the components dependencies and its version number. This version number is used to determine schema and data updates when bin/magento setup:upgrade is run.
- composer.json: Specifies component dependencies and other metadata.

Create AHT Blog module for Magento 2

To create AHT Blog module, you need to complete the following high-level steps:

- Step 1: Create the folder of AHT Blog module
- Step 2: Create etc/module.xml file
- Step 3: Create etc/registration.php file
- Step 4: Enable the module
- Step 5: Create routes.xml file.

- Step 6: Create controller file.
- Step 7: Create View: Block, Layouts, Templates.
- Step 8: CRUD Models

Step 1: Create the folder of AHT Blog module

Name of the module is defined as "VendorName_ModuleName". First part is name of the vendor and last part is name of the module: For example: Magento_Blog, AHT_OnePageCheckout. Focus on following guide to create the folders:

```
  app/code/AHT/Blog
```

Step 2: Create etc/module.xml file.

Then, it is necessary to create etc folder and add the module.xml file

```
  app/code/AHT/Blog/etc/module.xml
```

Contents would be:

```
<?xml version="1.0"?>

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">

    <module name="AHT_Blog" setup_version="1.0.0">

    </module>

</config>
```

Step 3: Create etc/registration.php file

In this step, we will add registration.php as following guide:

```
  app/code/AHT/Blog/registration.php
```

Contents would be:

```
<?php

\Magento\Framework\Component\ComponentRegistrar::register(

        \Magento\Framework\Component\ComponentRegistrar::MODULE,

        'AHT_Blog',
        __DIR__
);
```

Step 4: Enable the module

Finish the step 3, we have already created the `Blog` module. And we will enable this module in this step
After create the module if you run the command as:

```
  php bin/magento module:status
```

You should see the module is disable now:
List of disabled modules: `AHT_Blog`
Follow exact guide to enable the module right now, let run the command as:

```
  php bin/magento module:enable AHT_Blog
```

Or other way, you can access the file:

```
  app/etc/config.php
```

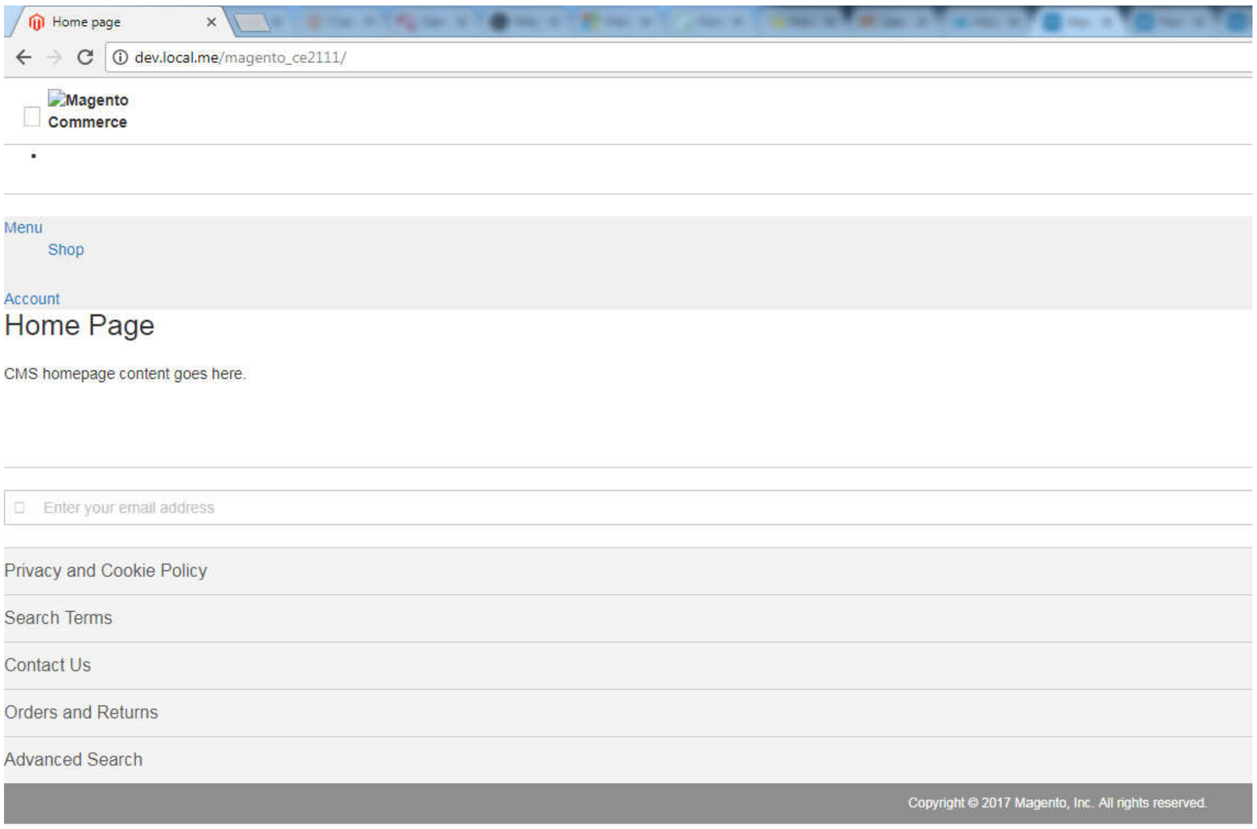
You will see a long list of modules there, just add your module as well

```
  ...

  'AHT_Blog' => 1,

  ....
```

Your module should be available now.
After this step, when you open your website in browser you will get an error saying
Please upgrade your database: Run “bin/magento setup:upgrade” from the Magento root directory.
Let run the command:

```
  php bin/magento setup:upgrade
```

After complete,when you open your website in browser you will see the layout of the website is broken.



Home Page

CMS homepage content goes here.

Enter your email address

Privacy and Cookie Policy

Search Terms

Contact Us

Orders and Returns

Advanced Search

Copyright © 2017 Magento, Inc. All rights reserved.

Please run

```
php bin/magento setup:static-content:deploy
```

to fix this.
After deploy completed,you can also see your module from backend at System Configuration -> Advanced -> Disable Modules Output.

Step 5: Create routes.xml file.

Now, we will create a controller to test module.
Before create a controller, we will create a route for Blog module.
Route's in magento are divided into 3 parts: Route frontname, controller and action as following example:

```
http://<yourhost.com>/index.php/route_name/controller/action
```

- `route_name` is a unique name which is set in routes.xml.
- `controller` is the folder inside Controller folder.
- `action` is a class with execute method to process request.

To add route, it is necessary to create routes.xml file
`app/code/AHT/Blog/etc/frontend/routes.xml`

since this is a frontend route, we added it in frontend/ folder else we need to add it to adminhtml/ folder
Content would be:

```
<?xml version="1.0" ?>

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLo
cation="urn:magento:framework:App/etc/routes.xsd">

    <router id="standard">

        <route frontName="blog" id="blog">

            <module name="AHT_Blog"/>

        </route>

    </router>

</config>
```

After define the first part of the route, the URL will be displayed as:

```
  http://<yourhost.com>/blog/
```

Step 6: Create test controller file

Then, we will continue the controller and action
The folder and file you need to create is:
`app/code/AHT/Blog/Controller/Index/Test.php`
Contents would be:

```
<?php

namespace AHT\Blog\Controller\Index;



class Test extends \Magento\Framework\App\Action\Action

{

        protected $_pageFactory;



        public function __construct(

                \Magento\Framework\App\Action\Context $context,

                \Magento\Framework\View\Result\PageFactory $pageFactory)
        {
                $this->_pageFactory = $pageFactory;
                return parent::__construct($context);
        }

        public function execute()
        {
                echo "AHT Blog";
                exit;
        }
}
```

After completed, please run `php bin/magento cache:clean` to check result.
Your URL now should be as:

```
http://<yourhost.com>/blog/index/test
```

After finish all steps, the output `AHT Blog` should be displayed in your browser when you open the URL.

Step 7: Create View: Block, Layouts, Templates

Firstly, we will create a controller to call the layout file .xml

File: `app/code/AHT/Blog/Controller/Index/Index.php`

```
<?php
namespace AHT\Blog\Controller\Index;

class Index extends \Magento\Framework\App\Action\Action
{
        protected $_pageFactory;
        public function __construct(
                \Magento\Framework\App\Action\Context $context,

                \Magento\Framework\View\Result\PageFactory $pageFactory)

        {

                $this->_pageFactory = $pageFactory;

                return parent::__construct($context);

        }



        public function execute()

        {

                return $this->_pageFactory->create();

        }

}
```

We have to declare the PageFactory and create it in execute method to render view.

Then, create layout file .xml:

The Layout is the major path of view layer in Magento 2 module. The layout file is a XML file which will define the page structure and will be locate in `{module_root}/view/{area}/layout/` folder. The Area path can be frontend or adminhtml which define where the layout will be applied.
There is a special layout file name `default.xml` which will be applied for all the page in it's area. Otherwhile, the layout file will have name as format: `{router_name}_{controller_name}_{action_name}.xml`.

When rendering page, Magento will check the layout file to find the handle for the page and then load Block and Template. We will create a layout handle file for this module:

File: `app/code/AHT/Blog/view/frontend/layout/blog_index_index.xml`

```
<?xml version="1.0"?>

<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" layout="1column" xsi:noNa
mespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configuration.xsd">

    <referenceContainer name="content">

        <block class="AHT\Blog\Block\Index" name="blog_index" template="AHT_Blog::ind
ex.phtml" />

    </referenceContainer>
```

```
</page>
```

In this file, we define the block and template for this page:
Block class: `AHT\Blog\Block\Index`
Template file: `AHT_Blog::index.phtml`
name: It is the required attribute and is used to identify a block as a reference

Create Block: The Block file should contain all the view logic required, it should not contain any kind of html or css. Block file are supposed to have all application view logic.

Create a file:

```
app/code/AHT/Blog/Block/Index.php
```

Contents would be:

```
<?php

namespace AHT\Blog\Block;

class Index extends \Magento\Framework\View\Element\Template

{

        public function __construct(\Magento\Framework\View\Element\Template\Context
 $context)

        {

                parent::__construct($context);

        }



        public function getBlogInfo()

        {

                return __('AHT Blog module');

        }

}
```

Every block in Magento 2 must extend from `Magento\Framework\View\Element\Template`. In this block we will define a method getBlogInfo() to show the word “AHT Blog module”. We will use it in template file.

Create a template file call `index.phtml`

```
app/code/AHT/Blog/view/frontend/templates/index.phtml
```

Insert the following code:

```
<?php



/**

 * @var \AHT\Blog\Block\Index $block

 */



echo $block->getBlogInfo();
```

In the layout file, we define the template by `AHT_Blog::index.phtml`. It mean that Magento will find the file name index.phtml in templates folder of module AHT_Blog. The template folder of the module is `app/code/{vendor_name}/{module_name}/view/frontend/templates/`.
In the template file, we can use the variable $block for the block object. As you see, we call the method `getBlogInfo()` in Block. It's done, please access to this page again (http://<yourhost>/blog/index/index) and see the result.

Step 8: CRUD Models

CRUD Models in Magento 2 can manage data in database easily, you don't need to write many line of code to create a CRUD. CRUD is stand for Create, Read, Update and Delete. We will learn about some main contents: How to setup Database, Model, Resource Model and Resource Magento 2 Get Collection and do database related operations.

Create a table aht_blog_post with the following columns:

- `post_id` - the post unique identifier
- `name` - the name of the post
- `url_key` - url of the post
- `image` - the image of the post
- `content` - the content of the post
- `status` - the status of the post
- `created_at` - the date created of the post
- `updated_at` - the date updated of the post

Firstly, we will create database table for our CRUD models. To do this we need to insert the setup file:

```
app/code/AHT/Blog/Setup/InstallSchema.php
```

This file will execute only one time when install the module. Let put this content for this file to create above table:

```
<?php

namespace AHT\Blog\Setup;



class InstallSchema implements \Magento\Framework\Setup\InstallSchemaInterface

{



        public function install(\Magento\Framework\Setup\SchemaSetupInterface $setup
, \Magento\Framework\Setup\ModuleContextInterface $context)

        {

                $installer = $setup;

                $installer->startSetup();

                if (!$installer->tableExists('aht_blog_post')) {

                        $table = $installer->getConnection()->newTable(

                                $installer->getTable('aht_blog_post')

                        )

                                ->addColumn(

                                        'post_id',

                                        \Magento\Framework\DB\Ddl\Table::TYPE_INTEG
ER,

                                        null,

                                        [

                                                'identity' => true,

                                                'nullable' => false,

                                                'primary'  => true,

                                                'unsigned' => true,

                                        ],

                                        'Post ID'

                                )
                                ->addColumn(

                                        'name',

                                        \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,

                                        255,

                                        ['nullable => false'],

                                        'Name'

                                )

                                ->addColumn(

                                        'url_key',

                                        \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,

                                        255,

                                        [],

                                        'URL Key'

                                )

->addColumn(

                                        'image',

                                        \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,

                                        null,

                                        ['nullable' => true, 'default' => null],

                                        'Image'

                                )


                                ->addColumn(

                                        'content',

                                        \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,

                                        '64k',

                                        ['nullable' => false],
                                        'Content'
                                )
                                ->addColumn(
                                        'status',
                                        \Magento\Framework\DB\Ddl\Table:: TYPE_SMALL
INT,
                                        null,
                                        ['nullable' => false, 'default' => '1'],
                                        'Status'
                                )
                                ->addColumn(
                                                'created_at',
                                                \Magento\Framework\DB\Ddl\Table::TY
PE_TIMESTAMP,
                                                null,
                                                ['nullable' => false, 'default' =>
\Magento\Framework\DB\Ddl\Table::TIMESTAMP_INIT],
                                                'Created At'
                                )->addColumn(
                                        'updated_at',
                                        \Magento\Framework\DB\Ddl\Table::TYPE_TIMES
TAMP,
                                        null,
                                        ['nullable' => false, 'default' => \Magento
\Framework\DB\Ddl\Table::TIMESTAMP_INIT_UPDATE],
                                        'Updated At')
                                ->setComment('Blog Post Table');
                        $installer->getConnection()->createTable($table);


                        $installer->getConnection()->addIndex(
                                $installer->getTable('aht_blog_post'),

                                $setup->getIdxName(

                                        $installer->getTable('aht_blog_post'),

                                        ['name','url_key','image','content'],

                                        \Magento\Framework\DB\Adapter\AdapterInterf
ace::INDEX_TYPE_FULLTEXT

                                ),

                                ['name','url_key','image','content'],

                                \Magento\Framework\DB\Adapter\AdapterInterface::IND
EX_TYPE_FULLTEXT

                        );

                }

                $installer->endSetup();

        }

}
```

This content is showing how the table created, you can edit it to make your own table. Please note that Magento will automatically run this file for the first time when installing the module. If you installed the module before, you will need to upgrade module and write the table create code to the UpgradeSchema.php in that folder and change attribute `setup_version` greater than current setup version in `module.xml` at `app/code/AHT/Blog/etc/module.xml`.
Contents would be:
File: `app/code/AHT/Blog/etc/module.xml`

```
<?xml version="1.0"?>

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLo
cation="urn:magento:framework:Module/etc/module.xsd">

    <module name="AHT_Blog" setup_version="1.0.1">

    </module>

</config>
```

In `module.xml` file, we changed the attribute to `1.0.1` greater than `setup_version` before
File: `app/code/AHT/Blog/Setup/UpgradeSchema.php`

```
<?php

namespace AHT\Blog\Setup;



use Magento\Framework\Setup\UpgradeSchemaInterface;

use Magento\Framework\Setup\SchemaSetupInterface;

use Magento\Framework\Setup\ModuleContextInterface;



class UpgradeSchema implements UpgradeSchemaInterface

{

        public function upgrade( SchemaSetupInterface $setup, ModuleContextInterface
 $context ) {

                $installer = $setup;



                $installer->startSetup();



                if(version_compare($context->getVersion(), '1.0.1', '<')) {

                        if (!$installer->tableExists('aht_blog_post')) {

                                $table = $installer->getConnection()->newTable(

                                        $installer->getTable('aht_blog_post')

                                )

                                        ->addColumn(

                                        'post_id',

                                        \Magento\Framework\DB\Ddl\Table::TYPE_INTEG
ER,

                                        null,

                                        [

                                                'identity' => true,

                                                'nullable' => false,
                                                'primary'  => true,

                                                'unsigned' => true,

                                        ],

                                        'Post ID'

                                )

                                ->addColumn(

                                        'name',

                                        \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,

                                        255,

                                        ['nullable => false'],

                                        'Name'

                                )

                                ->addColumn(

                                        'url_key',

                                        \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,

                                        255,

                                        [],

                                        'URL Key'

                                )

->addColumn(

                                        'image',

                                        \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,

                                        null,

                                        ['nullable' => true, 'default' => null],

                                        'Image'

                                )
```

```
                                 ->addColumn(
                                         'content',
                                         \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,
                                         '64k',
                                         ['nullable' => false],
                                         'Content'
                                 )
                                 ->addColumn(
                                         'status',
                                         \Magento\Framework\DB\Ddl\Table:: TYPE_SMALL
INT,
                                         null,
                                         ['nullable' => false, 'default' => '1'],
                                         'Status'
                                 )
                                 ->addColumn(
                                                 'created_at',
                                                 \Magento\Framework\DB\Ddl\Table::TY
PE_TIMESTAMP,
                                                 null,
                                                 ['nullable' => false, 'default' =>
\Magento\Framework\DB\Ddl\Table::TIMESTAMP_INIT],
                                                 'Created At'
                                 )->addColumn(
                                         'updated_at',
                                         \Magento\Framework\DB\Ddl\Table::TYPE_TIMES
TAMP,
                                         null,
                                         ['nullable' => false, 'default' => \Magento
\Framework\DB\Ddl\Table::TIMESTAMP_INIT_UPDATE],
                                         'Updated At')

                                 ->setComment('Blog Post Table');

                        $installer->getConnection()->createTable($table);


                        $installer->getConnection()->addIndex(

                                 $installer->getTable('aht_blog_post'),

                                 $setup->getIdxName(

                                         $installer->getTable('aht_blog_post'),

                                         ['name','url_key','image','content'],

                                         \Magento\Framework\DB\Adapter\AdapterInterf
ace::INDEX_TYPE_FULLTEXT

                                 ),

                                 ['name','url_key','image','content'],

                                 \Magento\Framework\DB\Adapter\AdapterInterface::IND
EX_TYPE_FULLTEXT

                        );

                        }

                }


                $installer->endSetup();

        }

}
```

After this please run this command line:

```
php bin/magento setup:upgrade
```

When you run upgrade completed, please continue run deploy like this

```
php bin/magento setup:static-content:deploy
```

Now checking your database, you will see a table with name `aht_blog_post` and above columns. If this table is not created, it may be because you ran the above command line before you add content to

InstallSchema.php. To fix this, you need remove the information that let Magento know your module has installed in the system. Please open the table 'setup_module', find and remove a row has module equals to `aht_blog_post`. After this, run the command again to install the table.
This `InstallSchema.php` is used to create database structure. If you want to install the data to the table which you was created, you need to use `InstallData.php` file:

```
app/code/AHT/Blog/Setup/InstallData.php
```

Please take a look in some InstallData file in Magento to know how to use it. This's some file you can see:

```
- vendor/magento/module-tax/Setup/InstallData.php

- vendor/magento/module-customer/Setup/InstallData.php

- vendor/magento/module-catalog/Setup/InstallData.php
```

As I said above, those install file will be used for first time install the module. If you want to change the database when upgrade module, please try to use `UpgradeSchema.php` and `UpgradeData.php`.
Model is a huge path of MVC architecture. In Magento 2 CRUD, models have many different functions such as manage data, install or upgrade module. We have to create Model, Resource Model, Resource Model Conllection to manage data in table: `aht_blog_post` as I mentioned above.
Now we will create the model file:

```
app/code/AHT/Blog/Model/Post.php
```

And this is the content of that file:

```
<?php

namespace AHT\Blog\Model;

class Post extends \Magento\Framework\Model\AbstractModel implements \Magento\Framewo
rk\DataObject\IdentityInterface

{

        const CACHE_TAG = 'aht_blog_post';



        protected $_cacheTag = 'aht_blog_post';



        protected $_eventPrefix = 'aht_blog_post';



        protected function _construct()

        {
                $this->_init('AHT\Blog\Model\ResourceModel\Post');

        }


        public function getIdentities()

        {

                return [self::CACHE_TAG . '_' . $this->getId()];

        }


        public function getDefaultValues()

        {

                $values = [];


                return $values;

        }

}
```

This model class will extends AbstractModel class `Magento\Framework\Model\AbstractModel` and implements `\Magento\Framework\DataObject\IdentityInterface`. The IdentityInterface will force Model class define the `getIdentities()` method which will return a unique id for the model. You must only use this interface if your model required cache refresh after database operation and render information to the frontend page.
The `_construct()` method will be called whenever a model is instantiated. Every CRUD model have to use the _construct() method to call _init() method. This _init() method will define the resource model which will actually fetch the information from the database. As above, we define the resource model Mageplaza\Post\Model\ResourceModel\Post The last thing about model is some variable which you should you in your model:

- `$_eventPrefix` - a prefix for events to be triggered
- `$_eventObject` - a object name when access in event
- `$_cacheTag` - a unique identifier for use within caching

As you know, the model file contain overall database logic, it do not execute sql queries. The resource model will do that. Now we will create the Resource Model for this table: `app/code/AHT/Blog/Model/ResourceModel/Post.php`
Content for this file:

```
<?php

namespace AHT\Blog\Model\ResourceModel;
```

```
class Post extends \Magento\Framework\Model\ResourceModel\Db\AbstractDb

{



        public function __construct(

                \Magento\Framework\Model\ResourceModel\Db\Context $context

        )

        {

                parent::__construct($context);

        }



        protected function _construct()

        {

                $this->_init('aht_blog_post', 'post_id');

        }



}
```

Every CRUD resource model in Magento must extends abstract class `\Magento\Framework\Model\ResourceModel\Db\AbstractDb` which contain the functions for fetching information from database.
Like model class, this resource model class will have required method `_construct()`. This method will call `_init()` function to define the table name and primary key for that table. In this example, we have table `aht_blog_post` and the primary key `post_id`.

The collection model is considered a resource model which allow us to filter and fetch a collection table data. The collection model will be placed in:

```
app/code/AHT/Blog/Model/ResourceModel/Post/Collection.php
```

The content for this file:

```
<?php

namespace AHT\Blog\Model\ResourceModel\Post;
```

```
class Collection extends \Magento\Framework\Model\ResourceModel\Db\Collection\Abstrac
tCollection

{

        protected $_idFieldName = 'post_id';

        protected $_eventPrefix = 'aht_blog_post_collection';

        protected $_eventObject = 'post_collection';



        /**

         * Define resource model

         *

         * @return void

         */

        protected function _construct()

        {

                $this->_init('AHT\Blog\Model\Post', 'AHT\Blog\Model\ResourceModel\Po
st');

        }



}
```

The CRUD collection class must extends from `\Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection` and call the `_init()` method to init the model, resource model in `_construct()` function.

We are done with creating the database table, CRUD model, resource model and collection. So how to use them?
In this part, we will talk about Factory Object for model. As you know in OOP, a factory method will be used to instantiate an object. In Magento, the Factory Object do the same thing.
The Factory class name is the name of Model class and append with the 'Factory' word. So for our example, we will have PostFactory class. You must not create this class. Magento will create it for you. Whenever Magento's object manager encounters a class name that ends in the word 'Factory', it will automatically generate the Factory class in the var/generation folder if the class does not already exist. You will see the factory class in

```
var/generation/<vendor_name>/<module_name>/Model/ClassFactory.php
```

In this case, it will be:

```
var/generation/AHT/Blog/Model/PostFactory.php
```

For example, we will call the model to get data in controller.

```
app/code/AHT/Blog/Controller/Index/Index.php
```

Content for this file:

```
<?php

namespace AHT\Blog\Controller\Index;


class Index extends \Magento\Framework\App\Action\Action

{

        protected $_pageFactory;


        protected $_postFactory;


        public function __construct(

                \Magento\Framework\App\Action\Context $context,

                \Magento\Framework\View\Result\PageFactory $pageFactory,

                \AHT\Blog\Model\PostFactory $postFactory

                )

        {

                $this->_pageFactory = $pageFactory;

                $this->_postFactory = $postFactory;

                return parent::__construct($context);

        }

        public function execute()

        {

                $post = $this->_postFactory->create();

                $collection = $post->getCollection();

                foreach($collection as $item){

                        echo "<pre>";

                        print_r($item->getData());

                        echo "</pre>";

                }

                exit();

                return $this->_pageFactory->create();

        }

}
```

As you see in this controller, the PostFactory object will be created in the `_construct()` function. In the `execute()` function, we use `$post = $this->_postFactory->create();` to create the model object.
Now, You need go to phpmyadmin and open `aht_blog_post` table to add some record to test post model work.

After completed, let's open browser and navigate to

```
http://<yourhost.com>/blog/index/index
```

and see the result.