Allele Database Tutorial

Requirements: Catalyst MVC framework, MySQL and MySQL Workbench must be installed.

The full source code for this tutorial can be found at:
https://github.com/Nerdylicious/AlleleDatabase

Create a new directory which will contain the entire project by running the following command in your home directory:

```
mkdir MyProject
```

Change directory into the MyProject directory:

```
cd MyProject
```

Create a new Catalyst application:

```
catalyst.pl AlleleDatabase
```

You should get output that looks like this:

```
created "AlleleDatabase"
created "AlleleDatabase/script"
created "AlleleDatabase/lib"
...
created "AlleleDatabase/script/alleledatabase_create.pl"
Change to application directory and Run "perl Makefile.PL" to make sure your
install is complete
```

Then change directory to AlleleDatabase:

```
cd AlleleDatabase
```

We should add the StackTrace plugin for debugging purposes. To do this, open the file lib/MyApp.pm:

`gedit lib/AlleleDatabase.pm` (you may use any text editor of your choice)

Add the StackTrace plugin so that the loaded plugins look like this:

```
# Load plugins
use Catalyst qw/
    -Debug
    ConfigLoader
    Static::Simple

    StackTrace
/;
```

Add the line `requires 'Catalyst::Plugin::StackTrace';` to Makefile.PL

Make sure that you are in the AlleleDatabase directory. We will now create a controller called 'Allele' using the following command:

```
script/alleledatabase_create.pl controller Allele
```

Then we will edit lib/AlleleDatabase/Controller/Allele.pm and add the following method to the controller below the 'index' method:

```
sub list :Local {
    my ($self, $c) = @_;
    $c->stash(template => 'allele/list.tt2');
}
```

Next we must create a Catalyst view. Run the following command to enable the TT style of view rendering:

```
script/alleledatabase_create.pl view HTML TT
```

This will create a file called HTML.pm that uses template toolkit as the rendering engine.

We want a wrapper page type of configuration for our views. Edit lib/AlleleDatabase/View/HTML.pm and update the package config so that it looks like the following:

```
__PACKAGE__->config(
    # Change default TT extension
    TEMPLATE_EXTENSION => '.tt2',
    render_die => 1,
);
```

This will change the default extension of Template Toolkit to '.tt2' instead of '.tt'

You must also change the package config in lib/AlleleDatabase.pm to the following:

```
__PACKAGE__->config(
    name => 'AlleleDatabase',
    # Disable deprecated behavior needed by old applications
    disable_component_resolution_regex_fallback => 1,
    enable_catalyst_header => 1, # Send X-Catalyst header
);
__PACKAGE__->config(
    # Configure the view
    'View::HTML' => {
        # Set the location for TT files
        INCLUDE_PATH => [
            __PACKAGE__->path_to('root', 'src'),
        ],
    },
);
```

This changes the base directory of your template files from 'root' to 'root/src'.

Next we will create a TT template page.

Change directory to MyProject/AlleleDatabase/root.

Then create a directory for TT templates related to alleles:

```
mkdir -p src/allele
```

Change directory to src/allele and create a file called list.tt2.

Add the following to list.tt2:

```
[% # This is a TT comment. %]

[%- # Provide a title %]
[% META title = 'Allele List' %]

<h1>Allele List</h1>
```

We can now test run our application by running the development server:

```
script/alleledatabase_server.pl -r
```

Try out the Catalyst main page by going to this location on your web browser:

localhost:3000

We can redirect to the 'index' method in the Allele.pm controller by going to the following location:

localhost:3000/allele

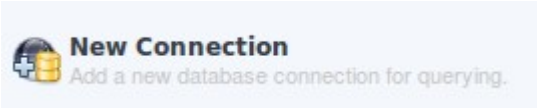
Next, we can redirect to the 'list' method in the Allele.pm controller by going to the following location:

localhost:3000/allele/list

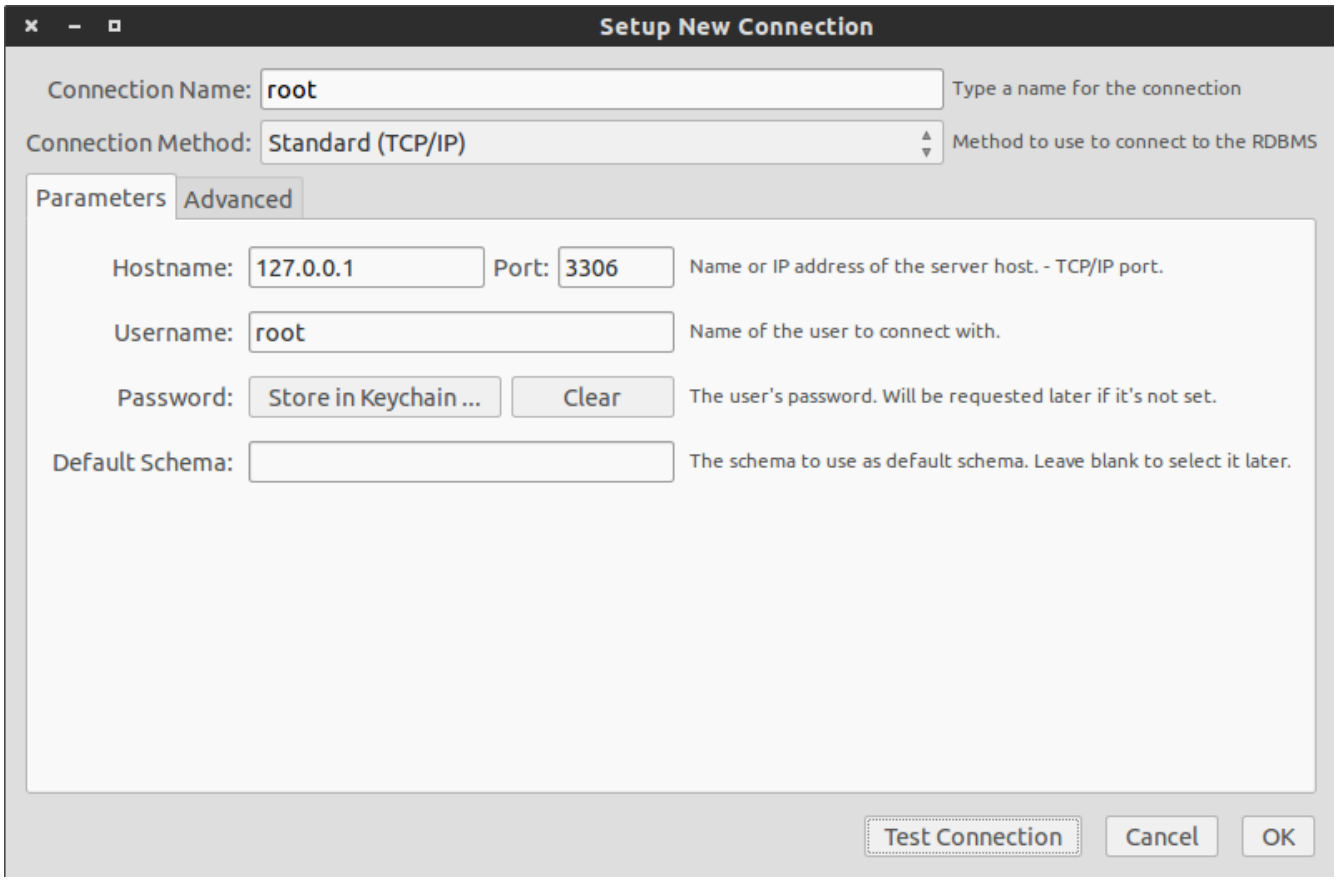
Our list.tt2 view is rendered since list.tt2 is included in the stash in the 'list' method.

Next we will create the database using the EER Model tools provided by MySQL Workbench.

Open MySQL Workbench. Create a new connection by clicking on 'New Connection'.



Next, create a root connection account with the following settings:



Please ensure that you have set up a password for the MySQL root account. If you have not set up a password for the MySQL root account, then run the following commands:

```
mysql -u root
```

In the mysql client:

```
UPDATE mysql.user SET Password = PASSWORD('newpwd') WHERE User = 'root';
```

Then flush privileges:

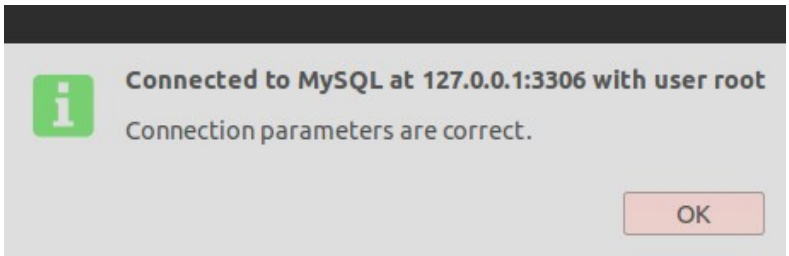
```
FLUSH PRIVILEGES;
```

You can now exit out of the mysql client by running the following command:

```
exit
```

In the Setup New Connection window, click on “Clear” beside Password. Next, click on “Test Connection” which will prompt you to enter your MySQL root account password.

Upon successfully testing your connection, you should get a message such as this one:



Next, create a new EER Model by clicking on “Create New EER Model”.

Double click on the mydb tab:

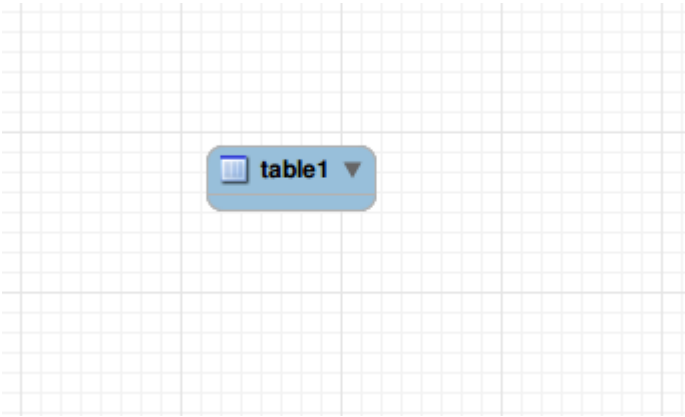
This should bring up a window that looks like this:

We will rename our database to “AlleleDatabase”:

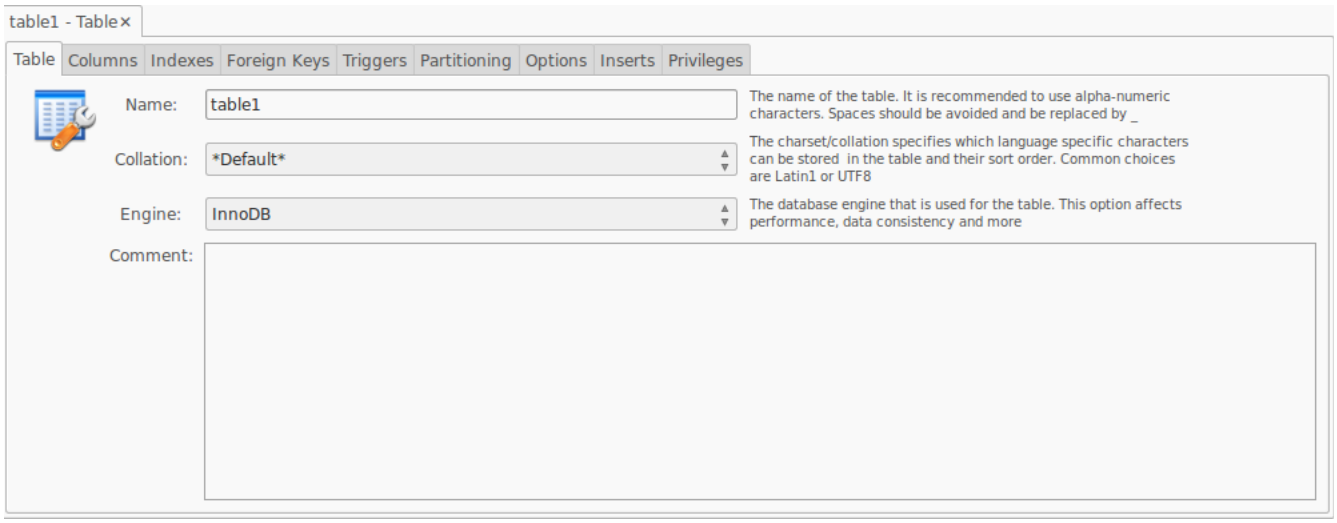
Next, Click on “Add Diagram”, this will open an EER Model workspace.

Click on the “Place a New Table” icon.

Next, click anywhere on the workspace in order to place this new table. This should create a new table called 'table1'.



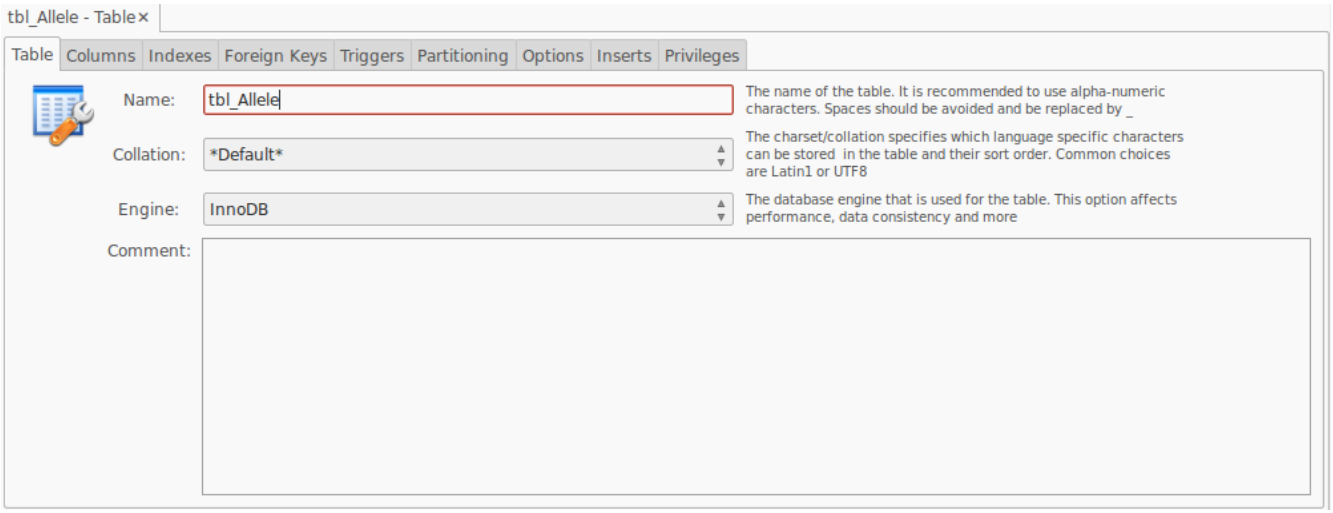
Double click on the 'table1' on your workspace. This should bring up a window that looks like this:

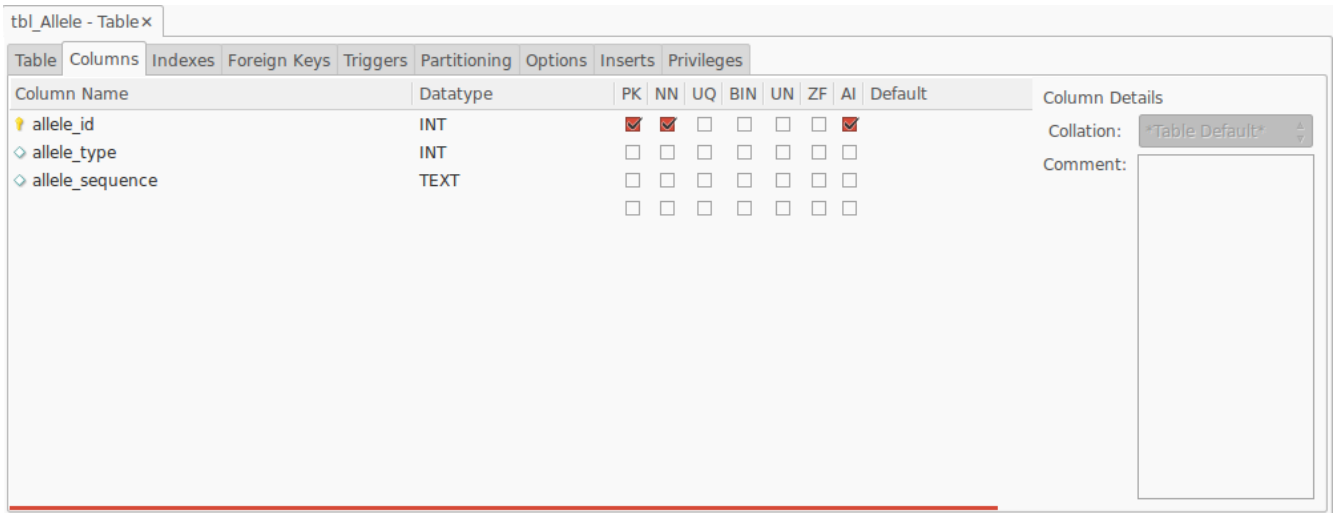


We want a table that adheres to this specification:

```
CREATE TABLE tbl_Allele(
    allele_id INT NOT NULL AUTO_INCREMENT,
    allele_type INT,
    allele_sequence TEXT,
    PRIMARY KEY (allele_id)
);
```

To do this, change the table schema as follows:





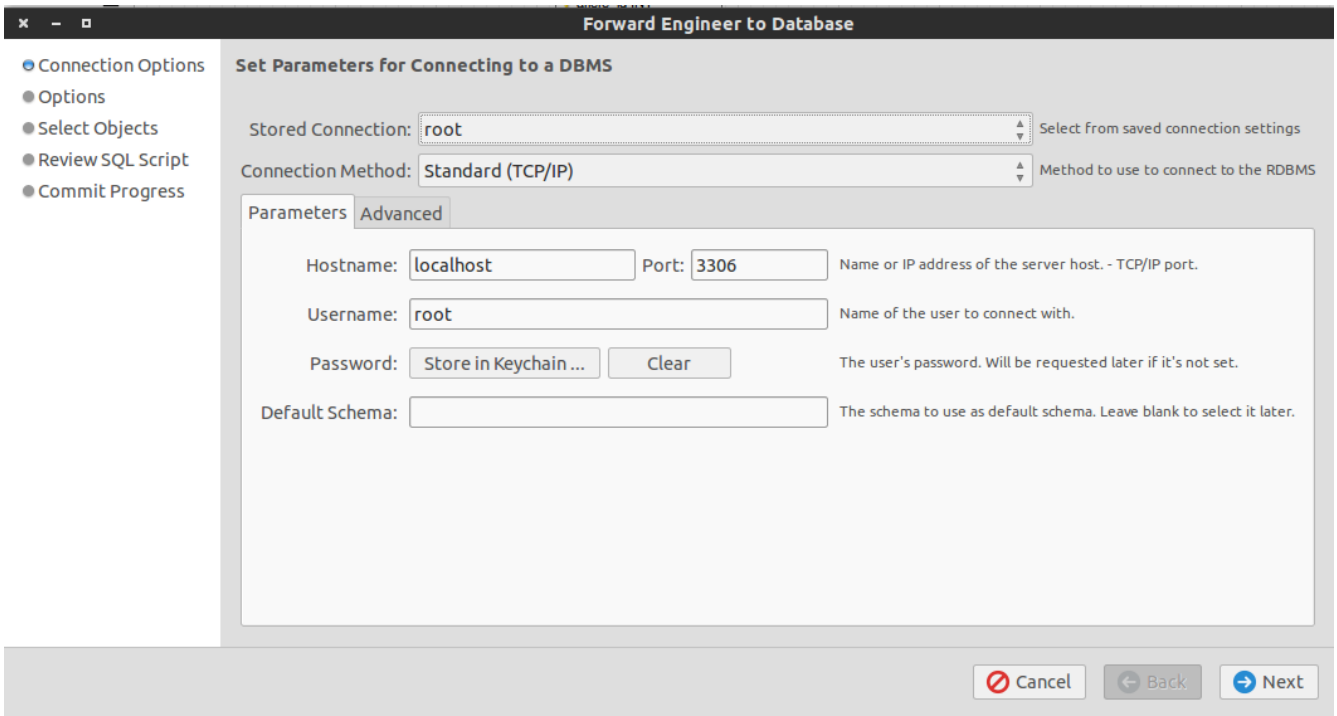
Make sure to check off the PK (Primary Key), NN (Not Null) and AI (Auto Increment) fields for the “allele_id” attribute.

You should end up with an EER Model that looks like this:

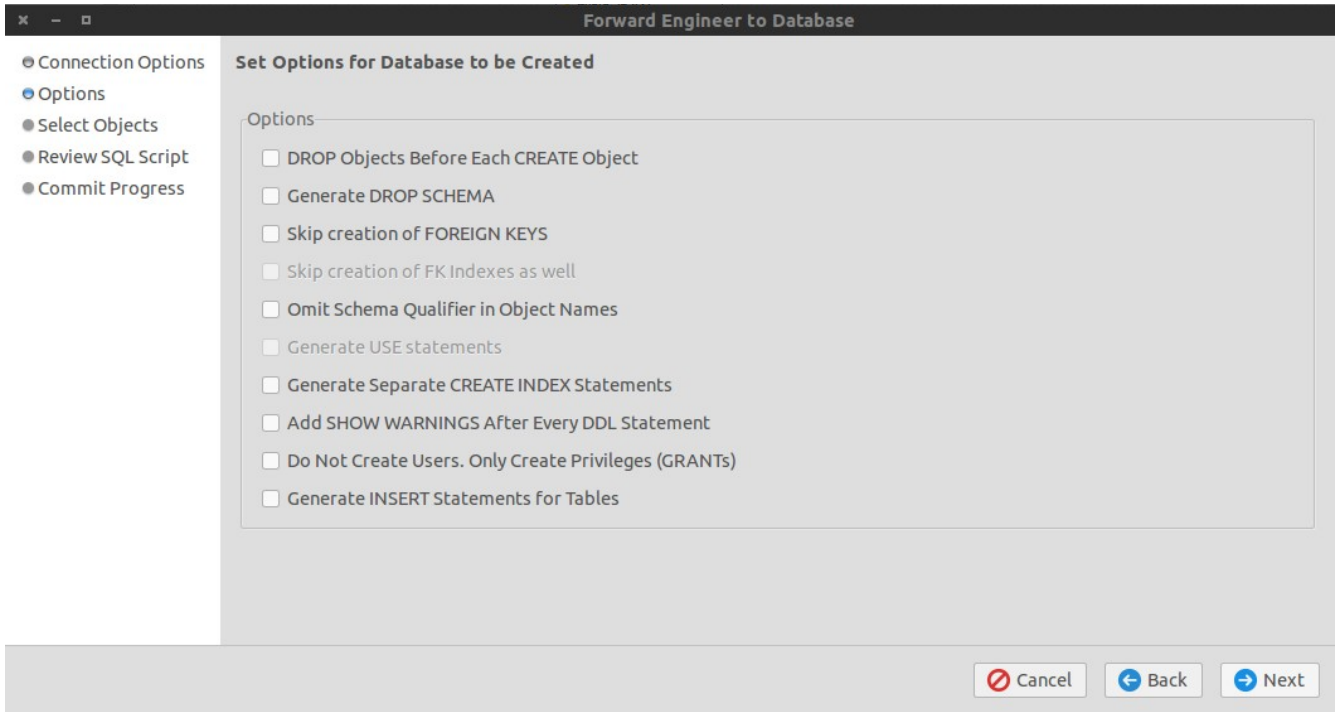
We will now “Forward Engineer” the model. Forward engineering automatically generates and executes the appropriate SQL statements for creating the tables and relationships specified in the corresponding EER Model. Forward engineering creates a MySQL database based on the schema specified in the EER Model.

To forward engineer, go to the toolbar and click on “Forward Engineer”.

For “Stored Connection”, select the root connection that you created earlier and click on “Next”:

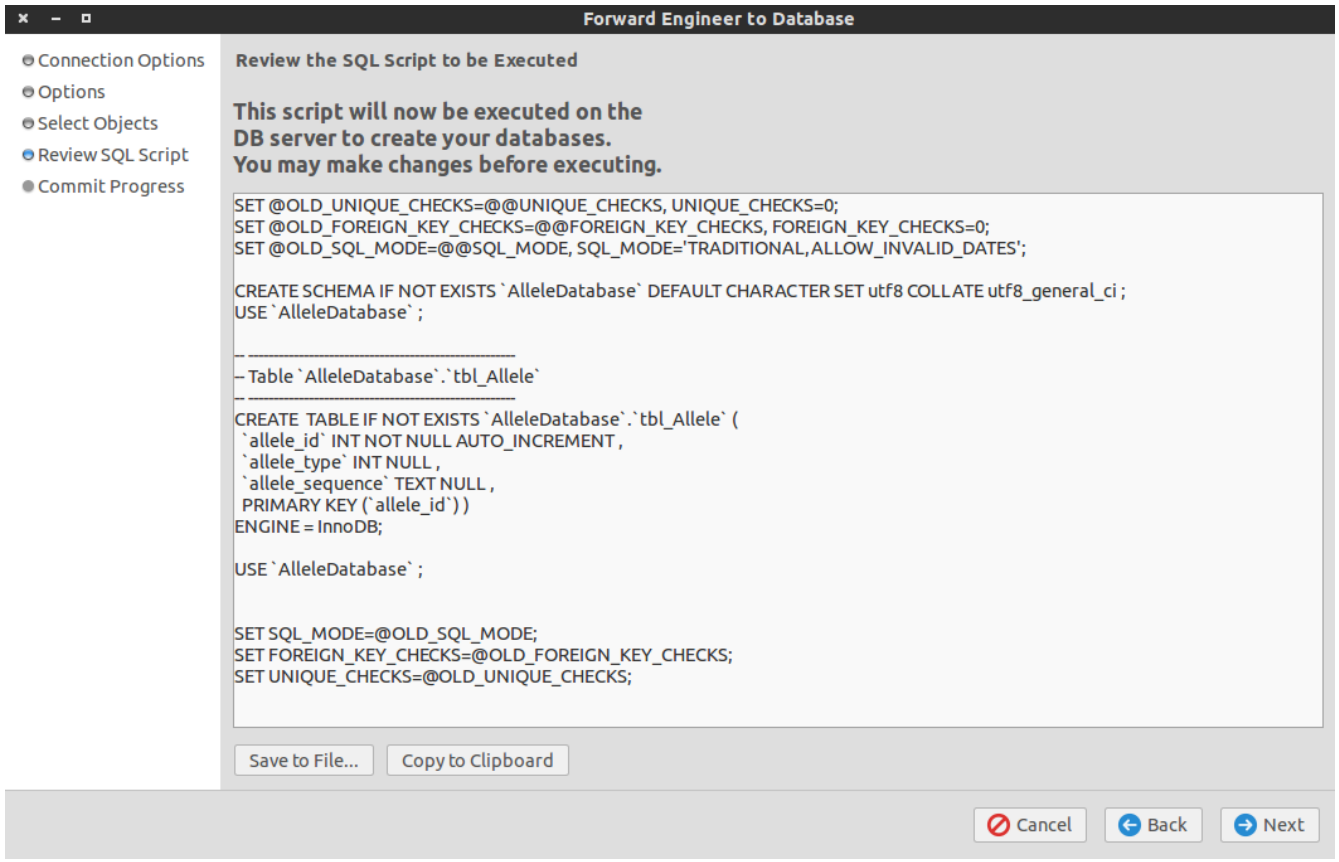


You will be presented with additional options, leave everything as is and click on “Next”:



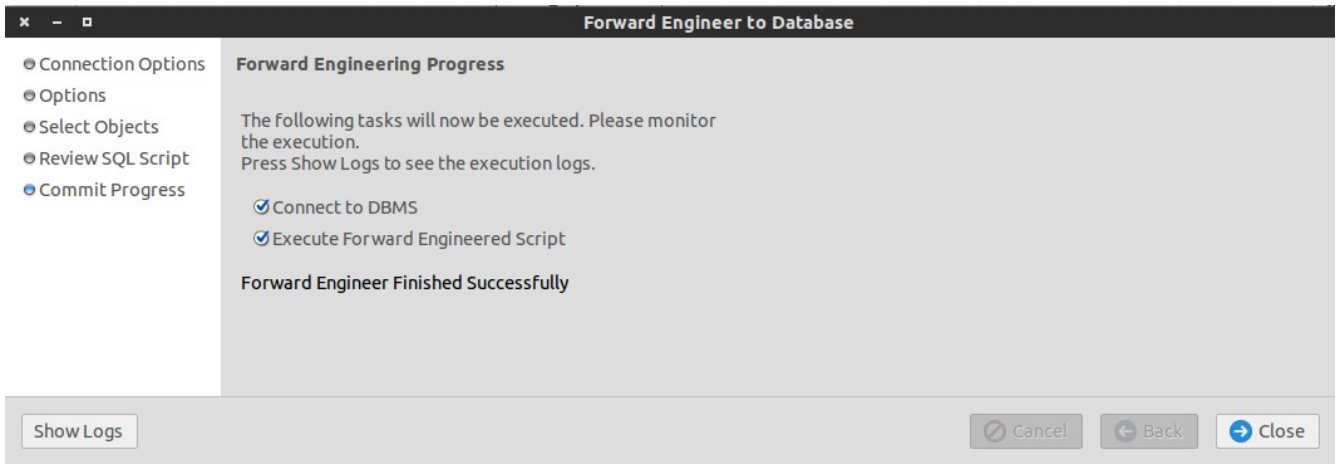
On “Select Objects to Forward Engineer” make sure to check off “Export MySQL Table Objects”:

You will now be presented with an automatically generated script that will be executed on the MySQL server to create your databases:



Click on “Next” to run this script.

You should get a message that says “Forward Engineer Finished Successfully”:



Next, login to the mysql client as root (you will be prompted for your root password):

```
mysql -u root -p
```

Once you have logged into the mysql client, you can view all mysql databases by running the following command:

```
show databases;
```

You should see “AlleleDatabase” in this list.

Next, run the following commands:

```
use AlleleDatabase;

show tables;
```

You should see the tables of AlleleDatabase:

```
+--------------------------+
| Tables_in_AlleleDatabase |
+--------------------------+
| tbl_Allele               |
+--------------------------+
```

We will be using DBIx::Class which is an ORM (Object Relational Mapper) to write code to access the database.

Create a directory called DatabaseObjects in the MyProject directory:

```
mkdir DatabaseObjects
```

Next, create a shell script called 'generate_db_models' with the following contents:

```
#!/bin/bash

#Purpose:   Run this program each time the database schema changes to re-generate
#           the database classes based on the current schema.
#Usage:     ./generate_db_models -d my_database_name -u my_dbms_username -p
my_dbms_password

NO_ARGS=0
E_OPTERROR=85

dbname=""
username=""
password=""

if [ $# -eq "$NO_ARGS" ]
then
    echo "Usage: `basename $0` -d database_name -u dbms_username -p dbms_password"
    exit $E_OPTERROR
fi

while getopts ":d:a:u:b:p:c" Option
do
    case $Option in
        d   ) dbname="$OPTARG";;
        u   ) username="$OPTARG";;
        p   ) password="$OPTARG";;
        *   ) echo "Unimplemented option chosen.";;
    esac
done

echo "dbname=$dbname"
echo "username=$username"

cmd='make_schema_at("NGSTAR::Schema",{debug=>1},["dbi:mysql:dbname='
cmd+=$dbname
cmd+='","'
cmd+=$username
cmd+='","'
cmd+=$password
cmd+='"])'

perl -MDBIx::Class::Schema::Loader=make_schema_at,dump_to_dir:./TestObjects -e $cmd

rm -r DatabaseObjects/
mv TestObjects DatabaseObjects

echo "Database classes successfully generated and is in the DatabaseObjects
directory"
```

The 'generate_db_models' script is used to automatically generate the DBIx::Class database models for us.

You will need to give execute permissions to this script. To do this, run the following command:

```
sudo chmod +x generate_db_models
```

You can check the usage of the generate_db_models script by running the script without providing any arguments:

```
./generate_db_models
Usage: generate_db_models -d database_name -u dbms_username -p dbms_password
```

Next, try running the script by providing the appropriate arguments:

```
./generate_db_models -d AlleleDatabase -u root -p your_root_password_here
```

After running the script you should get output that looks like this:

```
dbname=AlleleDatabase
username=root
AlleleDatabase::Schema::Result::TblAllele->table("tbl_Allele");
AlleleDatabase::Schema::Result::TblAllele->add_columns(
  "allele_id",
  { data_type => "integer", is_auto_increment => 1, is_nullable => 0 },
  "allele_type",
  { data_type => "integer", is_nullable => 1 },
  "allele_sequence",
  { data_type => "text", is_nullable => 1 },
);
AlleleDatabase::Schema::Result::TblAllele->set_primary_key("allele_id");
Dumping manual schema for AlleleDatabase::Schema to directory ./TestObjects ...
Schema dump completed.
Database classes successfully generated and is in the DatabaseObjects directory
```

The script creates DBIx::Class::Schema files based on the AlleleDatabase and are required in order to use the DBIx::Class ORM. In this case, the script generates a file called TblAllele.pm in MyProject/DatabaseObjects/AlleleDatabase/Schema/Result

We will build our forms using a Perl module called HTML::FormHandler. HTML forms can be constructed using HTML::FormHandler and HTML::FormHandler forms can automatically be rendered and validated. Validation for form fields can be specified upon defining a HTML::FormHandler form. One of the primary reasons as to why we use HTML::FormHandler is because it does client-side validation for us. We do not need to write JavaScript/jQuery code to perform basic client-side validation, all of this is already provided by the module as long as the developer specifies validation rules.

Make sure that you have the HTML::FormHandler module installed. If you don't have HTML::FormHandler then you can install it by running the following command:

```
sudo cpanm HTML::FormHandler
```

Add the line `requires 'HTML::FormHandler';` to Makefile.PL

Create a directory in lib/AlleleDatabase called Form:

```
mkdir Form
```

Change directory to the Form directory.

Next, create a file called AddAlleleForm.pm in lib/AlleleDatabase/Form with the following contents:

```
package AlleleDatabase::Form::AddAlleleForm;

use HTML::FormHandler::Moose;
use namespace::autoclean;
extends 'HTML::FormHandler';

has_field 'allele_type' => (
    do_label => 0,
    type => 'Text',
    size => 2,
    required => 1,
    messages => {required => 'Please enter an allele type'},
    minlength => 1,
    maxlength => 3,
    validate_method => \&check_type
);

has_field 'allele_sequence' => (
    do_label => 0,
    type => 'TextArea',
    rows => 15,
    cols => 60,
    required => 1,
    messages => {required => 'Please enter a sequence'},
    minlength => 20,
    maxlength => 5000,
    validate_method => \&check_sequence
);

sub check_sequence{
    my ($self) = @_;

    unless($self->value =~ /\A[ATCG]+\z/i){
        $self->add_error('Please enter a valid sequence');
    }
}

sub check_type{
    my ($self) = @_;

    unless($self->value =~ /^[0-9]+$/){
        $self->add_error('Please enter a valid type');
    }
    else{
        if(($self->value < 0) or ($self->value > 999)){
            $self->add_error('Please enter a type that is in range');
        }
    }
}

1;
```

At the top of lib/AlleleDatabase/Controller/Allele.pm, add the following line:

```
use AlleleDatabase::Form::AddAlleleForm;
```

In lib/AlleleDatabase/Controller/Allele.pm we will implement an 'add' method:

```
sub add :Local{
    my ($self, $c) = @_;

    my $form = AlleleDatabase::Form::AddAlleleForm->new;
    $c->stash(template => 'allele/AddAlleleForm.tt2', form => $form);
    $form->process(params => $c->request->params);
    return unless $form->validated;
    $c->response->body('Allele successfully added!');
}
```

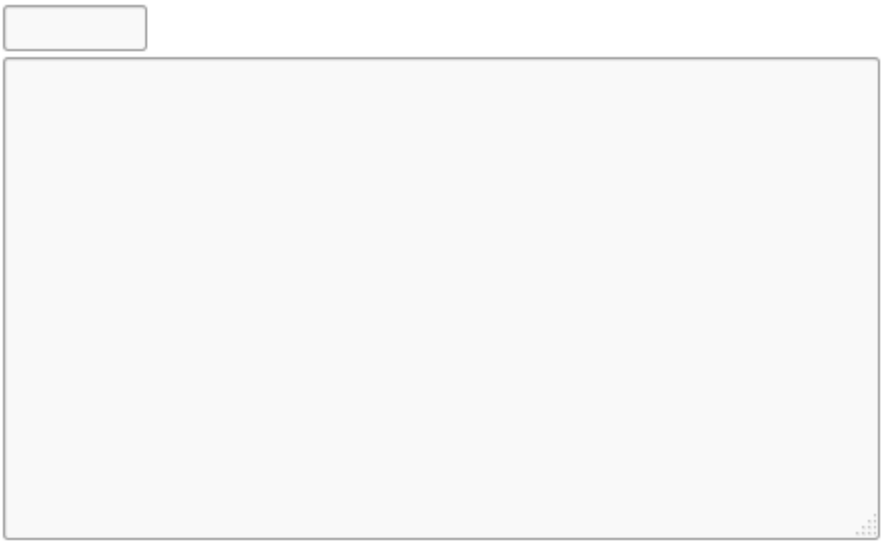
Next, create a new template called AddAlleleForm.tt2 in MyProject/AlleleDatabase/root/src/allele with the following contents:

```
[% META title = 'Add Allele' %]

[%# Render the HTML::FormHandler Form %]
[% form.render %]
```

Make sure that the development server is running. To run the development server, change directory to MyProject/AlleleDatabase and run the following command:

```
script/alleledatabase_server.pl -r
```

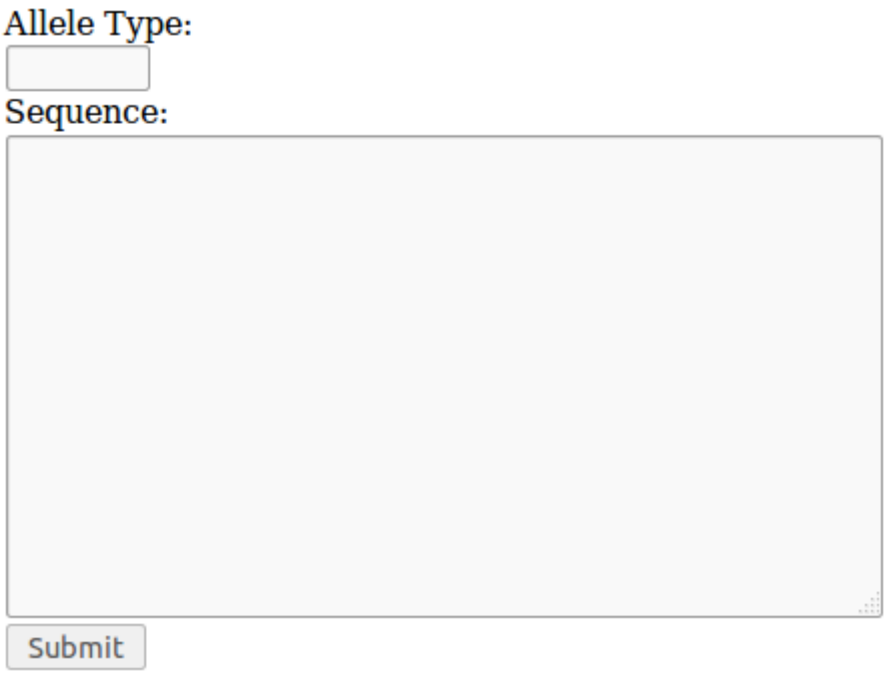
On your browser, go to localhost:3000/allele/add
You should see a form that looks like this:



We will want to add labels to these forms and a submit button as well. To do this, change the contents of AddAlleleForm.tt2 to:

```
[% META title = 'Add Allele' %]

[%# Render the HTML::FormHandler Form %]

<form>
    <label>Allele Type:</label>
    [% form.field('allele_type').render %]

    <label>Sequence:</label>
    [% form.field('allele_sequence').render %]

    <button type="submit">Submit</button>
</form>
```

Try refreshing the page. Your form should now look like this:



If you would like to see more examples on how to use HTML::FormHandler, clone the following project on GitHub and try running the example:
https://github.com/gshank/formhandler-example

Now we need to create a database access layer. Change directory to MyProject and use a tool called h2xs to create a new Perl module. This Perl module will contain our data access layer (DAL) code.

Run the following command to create a new Perl module:

```
h2xs -AX -n MyProjectDAL::Dao
```

For simplicity, rename the MyProjectDAL-Dao directory created to just MyProjectDAL:

```
mv MyProjectDAL-Dao MyProjectDAL
```

You will need to determine the location of Schema.pm. Change directory to MyProject/DatabaseObjects. Next run the following command:

```
pwd
```

You need to take note of the output produced by pwd as the output is needed in the next step.

Next, edit the file MyProjectDAL/lib/MyProjectDAL/Dao.pm to contain:

```
package MyProjectDAL::Dao;

use 5.014002;
use strict;
use warnings;

# Set the path to the output of pwd from the last step
use lib '/home/.../MyProject/DatabaseObjects';
use AlleleDatabase::Schema;

use Readonly;

require Exporter;

our @ISA = qw(Exporter);

# Items to export into callers namespace by default. Note: do not export
# names by default without a very good reason. Use EXPORT_OK instead.
# Do not simply export all your public functions/methods/constants.

# This allows declaration use MyProjectDAL::Dao ':all';
# If you do not need this, moving things directly into @EXPORT or @EXPORT_OK
# will save memory.
our %EXPORT_TAGS = ( 'all' => [ qw(

) ] );

our @EXPORT_OK = ( @{ $EXPORT_TAGS{'all'} } );

our @EXPORT = qw(
    insert_allele
);

our $VERSION = '0.01';

sub new{
    my ($class) = @_;
    my $self = {
        #you must specify the the mysql user and password here
        _schema => AlleleDatabase::Schema->connect('dbi:mysql:dbname=AlleleDatabase',
'root', 'my_root_user_password_here'),
    };
    bless $self, $class;
    return $self;
}

sub insert_allele{
    my ($self) = @_;
}

# Preloaded methods go here.

1;
__END__
```

Next, you need to run the makefile in MyProjectDAL:

```
perl Makefile.PL
```

Next run make and make install:

```
make
sudo make install
```

This installs the MyProjectDAL module to /usr/local/share/perl/x.y.z/ directory, making it easily accessible to the AlleleDatabase Catalyst project.

Each time you make a change to Dao.pm or any files in MyProjectDAL, you must run make and make install and restart the Catalyst development server in order for changes to take effect.

If you look in /usr/local/share/perl/x.y.z/ you should see the MyProjectDAL module (but do not edit the module from here).

Next, we will add logic to the 'insert_allele' method in Dao.pm:

```
sub insert_allele{
    my ($self, $allele_type, $allele_sequence) = @_;

    $self->{_schema}->resultset('TblAllele')->create({
        allele_type => $allele_type,
        allele_sequnece => $allele_sequence,
    });
}
```

We must run make and make install each time we make any updates to the MyProjectDAL module:

```
make
sudo make install
```

Now we need to create a business logic layer. Change directory to MyProject and use a tool called h2xs to create a new Perl module. This Perl module will contain our business logic code.

Run the following command to create a new Perl module:

```
h2xs -AX -n MyProjectBusinessLogic::MyProjectBusinessLogic
```

For simplicity, rename the MyProjectBusinessLogic-MyProjectBusinessLogic directory created to just MyProjectBusinessLogic:

```
mv MyProjectBusinessLogic-MyProjectBusinessLogic MyProjectBusinessLogic
```

Next, create a file called AddAllele.pm in MyProjectBusinessLogic/lib/MyProjectBusinessLogic with the following contents:

```
package MyProjectBusinessLogic::AddAllele;

use 5.014002;

use strict;
use warnings;

use Readonly;

use MyProjectDAL::Dao;

require Exporter;

our @ISA = qw(Exporter);

# Items to export into callers namespace by default. Note: do not export
# names by default without a very good reason. Use EXPORT_OK instead.
# Do not simply export all your public functions/methods/constants.

# This allows declaration use BusinessLogic::AddAllele':all';
# If you do not need this, moving things directly into @EXPORT or @EXPORT_OK
# will save memory.
our %EXPORT_TAGS = ( 'all' => [ qw(

) ] );

our @EXPORT_OK = ( @{ $EXPORT_TAGS{'all'} } );

our @EXPORT = qw(
    _add_allele
);

our $VERSION = '0.01';

sub new{
    my $class = shift;
    my $self = {
        _dao => MyProjectDAL::Dao->new(),
    };
    bless $self, $class;
    return $self;
}

sub _add_allele{
    my ($self, $allele_type, $allele_sequence) = @_;

    $self->{_dao}->insert_allele($allele_type, $allele_sequence);
}

1;
__END__
```

Each time a new file is added to MyProjectBusinessLogic/lib/MyProjectBusinessLogic, it must be specified in the MANIFEST.

In MyProject/MyProjectBusinessLogic, update the MANIFEST so that it includes AddAllele.pm:

```
Changes
Makefile.PL
MANIFEST
README
t/MyProjectBusinessLogic-MyProjectBusinessLogic.t
lib/MyProjectBusinessLogic/MyProjectBusinessLogic.pm
lib/MyProjectBusinessLogic/AddAllele.pm
```

Next, we must make and make install:

```
perl Makefile.PL
make
sudo make install
```

This installs the MyProjectBusinessLogic module to /usr/local/share/perl/x.y.z/

Now we need a way to call business logic code from our AlleleDatabase Catalyst project. It is important to understand that the business logic and data access layers are independent of the AlleleDatabase Catalyst project, so that there is proper separation of concerns and more modularity.

To do this, we use an Adaptor (design pattern). A Catalyst module has been written to provide this Adaptor pattern which can be found here: http://search.cpan.org/~bobtfish/Catalyst-Model-Adaptor-0.10/lib/Catalyst/Model/Adaptor.pm

If you haven't already installed the module, then you can do so by running the following command:

```
sudo cpanm Catalyst::Model::Adaptor
```

Next, change directory to MyProject/AlleleDatabase and run the following command to create an Adaptor:

```
script/alleledatabase_create.pl model AddAllele Adaptor
MyProjectBusinessLogic::AddAllele new
```

Restart the development server in order to make sure there aren't any errors at this point.

We will now update the Allele.pm controller so that we retrieve the information included in the form from the request parameters and call the business logic code for adding alleles to the database:

```
sub add :Local{
    my ($self, $c) = @_;

    my $form = AlleleDatabase::Form::AddAlleleForm->new;
    $c->stash(template => 'allele/AddAlleleForm.tt2', form => $form);

    $form->process(params => $c->request->params);
    return unless $form->validated;

    my $allele_type = $c->request->params->{allele_type};
    my $allele_sequence = $c->request->params->{allele_sequence};

    my $obj = $c->model('AddAllele');
    $obj->_add_allele($allele_type, $allele_sequence);

    $c->response->body('Allele successfully added!');
}
```

Next, go to localhost:3000/allele/add in your brower, fill out the form and submit it:

Allele Type:

1

Sequence:

ATGACAGATGAAGACGATGACGAGATGACGATATAGACGATA

Submit

Clicking on submit should return this message:

Allele successfully added!

We can verify that the allele was added to the database by logging into the mysql client and specifying the AlleleDatabase as the database to use:

```
mysql -u root -p
use database AlleleDatabase;
```

Next, run the following command to display the contents of the allele table:

```
select * from tbl_Allele;
```

You should get a result such as this:

```
+-----------+-------------+-------------------------------------------+
| allele_id | allele_type | allele_sequence                           |
+-----------+-------------+-------------------------------------------+
|         1 |           1 | ATGACAGATGAAGACGATGACGAGATGACGATATAGACGATA |
+-----------+-------------+-------------------------------------------+
1 row in set (0.01 sec)
```

We now want to implement functionality for listing all the alleles in the database.

First, we update MyProjectDAL/lib/MyProjectDAL/Dao.pm by adding the following lines of code:

```
#update the methods to export to include gt_allele_list
our @EXPORT = qw(
    get_allele_list
    insert_allele
);

#add this method to Dao.pm
sub get_allele_list{
    my ($self) = @_;

    my @allele_list;
    my $result_set = $self->{_schema}->resultset('TblAllele');

    while(my $allele = $result_set->next){
        push @allele_list, {allele_type => $allele->allele_type,
                            allele_sequence => $allele->allele_sequence};
    }

    return \@allele_list;
}
```

Next we need to make and make install again:

```
make
sudo make install
```

We need to add business logic code to make a call to get_allele_list. To do this, create a file called GetAlleleInfo.pm in MyProjectBusinessLogic/lib/MyProjectBusinessLogic

```
package MyProjectBusinessLogic::GetAlleleInfo;

use 5.014002;

use strict;
use warnings;

use Readonly;

use MyProjectDAL::Dao;

require Exporter;

our @ISA = qw(Exporter);

# Items to export into callers namespace by default. Note: do not export
# names by default without a very good reason. Use EXPORT_OK instead.
# Do not simply export all your public functions/methods/constants.

# This allows declaration    use BusinessLogic::AddAllele':all';
# If you do not need this, moving things directly into @EXPORT or @EXPORT_OK
# will save memory.
our %EXPORT_TAGS = ( 'all' => [ qw(

) ] );

our @EXPORT_OK = ( @{ $EXPORT_TAGS{'all'} } );

our @EXPORT = qw(
    _get_allele_list
);

our $VERSION = '0.01';

sub new{
    my $class = shift;
    my $self = {
        _dao => MyProjectDAL::Dao->new(),
    };
    bless $self, $class;
    return $self;
}

sub _get_allele_list{
    my ($self, $allele_type, $allele_sequence) = @_;

    my $allele_list = $self->{_dao}->get_allele_list();

    return $allele_list;
}

1;
__END__
```

Next, update the MANIFEST file:

```
Changes
Makefile.PL
MANIFEST
README
t/MyProjectBusinessLogic-MyProjectBusinessLogic.t
lib/MyProjectBusinessLogic/MyProjectBusinessLogic.pm
lib/MyProjectBusinessLogic/AddAllele.pm
lib/MyProjectBusinessLogic/GetAlleleInfo.pm
```

Then, make and make install again:

```
perl Makefile.PL
make
sudo make install
```

Next, change directory to MyProject/AlleleDatabase and run the following command to create an Adaptor:

```
script/alleledatabase_create.pl model GetAlleleInfo Adaptor
MyProjectBusinessLogic::GetAlleleInfo new
```

Restart the development server in order to make sure there aren't any errors at this point.

Update the allele controller 'list' method:

```
sub list :Local{
    my ($self, $c) = @_;

    my $obj = $c->model('GetAlleleInfo');
    my $allele_list = $obj->_get_allele_list();

    $c->stash(template => 'allele/list.tt2', allele_list => $allele_list);
}
```

Update the list.tt2 template:

```
[% # This is a TT comment. %]

[%- # Provide a title %]
[% META title = 'Allele List' %]

<h1>Allele List</h1>

<table>
    [% FOREACH allele IN allele_list %]
        <tr>
            <td>Allele Type</td>
            <td>Sequence</td>
        </tr>
        <tr>
            <td>[% allele.allele_type | html %]</td>
            <td>[% allele.allele_sequence | html %]</td>
        </tr>
    [% END %]
</table>
```

Go to locahost:3000/allele/list on your browser. You should see a table with the allele that you added in the previous steps.

Allele List

Allele Type	Sequence
1	ATGACAGATGAAGACGATGACGAGATGACGATATAGACGATA

We can update the 'add' method in the allele controller so that we automatically redirect to the Allele List view after successfully adding a new allele.

Update the line in the 'add' method of Allele.pm

```
$c->response->body('Allele successfully added!');
```

to be:

```
$c->response->redirect($c->uri_for($self->action_for('list')));
```

We may also want to make use of a CSS framework such as Bootstrap to style our application.

Download Bootstrap from http://www.getbootstrap.com/css/

Change directory to the Bootstrap compressed file that was downloaded and extract it.

Extracting the file should give a folder called 'dist' or 'bootstrap' with the contents 'css', 'fonts' and 'js'.

If this directory is named anything other than 'bootstrap' then rename it to 'bootstrap'.

Move the 'bootstrap' folder along with its contents to MyProject/AlleleDatabase/root/static

You also need to download jQuery from jquery.com

Move the 'jquery' directory to MyProject/AlleleDatabase/root/static

Next, create a base template file called bootstrap_wrapper.tt2 in MyProject/AlleleDatabase/root/src with the following contents:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="description" content="">
        <meta name="author" content="">

        <!-- Bootstrap core CSS -->
        <link href="/static/bootstrap/css/bootstrap.css" rel="stylesheet">
    </head>
    <body>
        <div class="container-fluid">
            <div id="content" class="col-sm-10 col-sm-offset-2 main">
                <!-- Render the content -->
                [% content %]
            </div>
        </div>
        <!-- These javascript files are loaded here (and not at the head) so
that pages load faster -->
        <!-- JQuery -->
        <script src="/static/jquery/jquery-1.11.0.min.js"></script>

        <!-- Bootstrap javascript -->
        <script src="/static/bootstrap/js/bootstrap.min.js"></script>
    </body>
</html>
```

If you have a different version of Bootstrap or jQuery, then the script import lines may differ slightly.

You will have to update the View::HTML configuration in MyProject/AlleleDatabase/lib/AlleleDatabase.pm to specify the base template file:

```
__PACKAGE__->config(
    # Configure the view
    'View::HTML' => {
        # Set the location for TT files
        INCLUDE_PATH => [
            __PACKAGE__->path_to('root', 'src'),
        ],
        WRAPPER => 'bootstrap_wrapper.tt2',
    },
);
```

Update MyProject/AlleleDatabase/lib/AlleleDatabase/Form/AddAlleleForm.pm so that it includes the line:

```
with 'HTML::FormHandler::Widget::Theme::Bootstrap3';
```

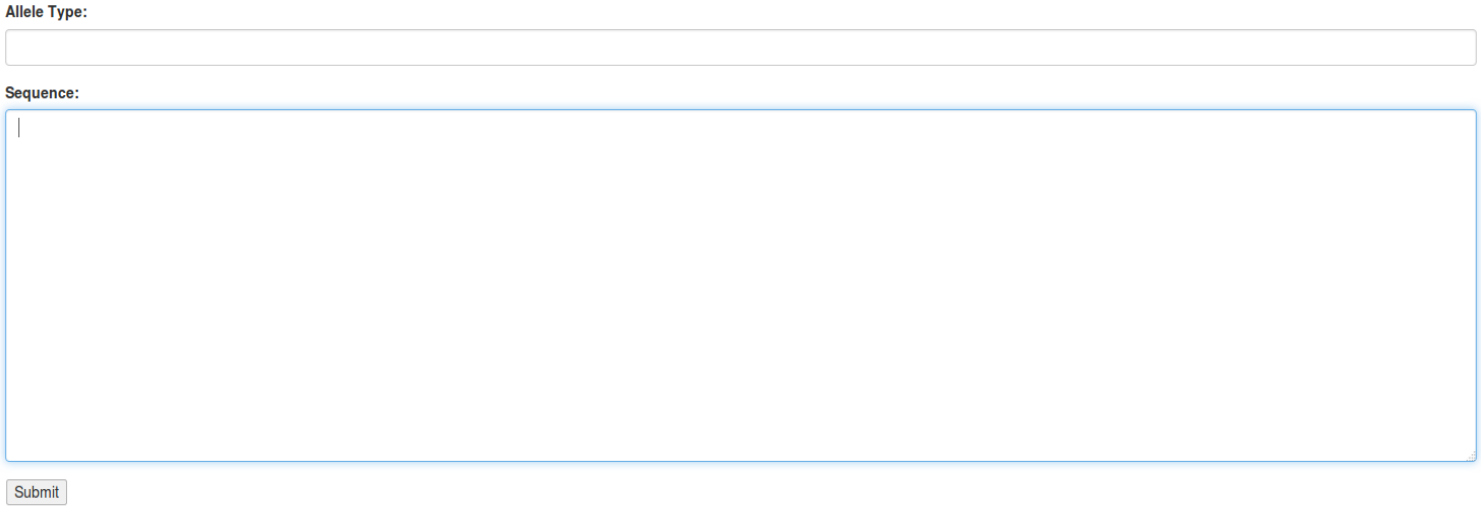
after the line:

```
extends 'HTML::FormHandler';
```

so that you have something that looks like this:

```
package AlleleDatabase::Form::AddAlleleForm;

use HTML::FormHandler::Moose;
use namespace::autoclean;
extends 'HTML::FormHandler';
with 'HTML::FormHandler::Widget::Theme::Bootstrap3';
```

Next, try visiting localhost:3000/allele/add on your browser. You should see some Bootstrap styling now:



Add some more styling to the form by updating the AddAlleleForm.tt2 template:

```
[% META title = 'Add Allele' %]

[%# Render the HTML::FormHandler Form %]

<div class="well">
    <form class="form-horizontal" role="form">
        <br>
        <div class="form-group">
            <div class="col-sm-offset-2">
                <label>Please enter information for the allele to add:</label>
            </div>
        </div>
        <div class="form-group">
            <label class="col-sm-2 control-label">Allele Type:</label>
            <div class="col-sm-10">
                [% form.field('allele_type').render %]
            </div>
        </div>
        <div class="form-group">
            <label class="col-sm-2 control-label">Sequence:</label>
            <div class="col-sm-10">
                [% form.field('allele_sequence').render %]
            </div>
        </div>
        <div class="form-group">
            <div class="col-sm-offset-2">
                <button type="submit" class="btn btn-primary">Submit</button>
            </div>
        </div>
    </form>
</div>
```

You should now get a form that looks like this:

Please enter information for the allele to add:

Allele Type:

Sequence:

Submit

Go to localhost:3000/allele/list. The styling for this page should look something like this:

Allele List

Allele TypeSequence
1 ATGACAGATGAAGACGATGACGAGATGACGATATAGACGATA
Allele TypeSequence
2 TAGACGATGACAGAGATTATAGACGATAGACGCGACGAGACGATGACGGACCCGAGTA
Allele TypeSequence
2 TAGACGATGACAGAGATTATAGACGATAGACGCGACGAGACGATGACGGACCCGAGTA
Allele TypeSequence
4 ATACGAGATGACGATGACGATGA
Allele TypeSequence
4 ATACGAGATGACGATGACGATGA

To add more styling to this view, update the list.tt2 template like so:

```
[% # This is a TT comment. %]

[%- # Provide a title %]
[% META title = 'Allele List' %]

<div class="panel panel-default">
    <div class="panel-heading">Allele List</div>
    <div class="panel-body">
    </div>
    <table class="table table-bordered table-hover table-striped">
        <tr>
            <td class="col-sm-1">Allele Type</td>
            <td>Sequence</td>
        </tr>
        [% FOREACH allele IN allele_list %]
            <tr>
                <td class="col-sm-1">[% allele.allele_type | html %]</td>
                <td>[% allele.allele_sequence | html %]</td>
            </tr>
        [% END %]
    </table>
</div>
```

Go to localhost:3000/allele/list. The styling for this page should look something like this:

Allele List

Allele Type	Sequence
1	ATGACAGATGAAGACGATGACGAGATGACGATATAGACGATA
2	TAGACGATGACAGAGATTATAGACGATAGACGCGACGAGACGATGACGGACCCGAGTA
2	TAGACGATGACAGAGATTATAGACGATAGACGCGACGAGACGATGACGGACCCGAGTA
4	ATACGAGATGACGATGACGATGA
4	ATACGAGATGACGATGACGATGA

Next, you may want to add a navbar to the base template. To do this, edit bootstrap_wrapper.tt2:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="description" content="">
        <meta name="author" content="">

        <!-- Bootstrap core CSS -->
        <link href="/static/bootstrap/css/bootstrap.css" rel="stylesheet">
    </head>
    <body>
        <div class="navbar navbar-default" role="navigation">
            <div class="container-fluid">
                <div class="navbar-header">
                    <a class="navbar-brand" href="[%
c.uri_for(c.controller('Allele').action_for('list')) %]">Allele Database</a>
                </div>
                <div class="collapse navbar-collapse">
                    <ul class="nav navbar-nav">
                        <li>
                            <a href="[% c.uri_for(c.controller('Allele').action_for('list'))
%]">
                                <span class="glyphicon glyphicon-home"></span> Home
                            </a>
                        </li>
                    </ul>
                    <ul class="nav navbar-nav navbar-right">
                        <li>
                            <a href="#">
                                About
                            </a>
                        </li>
                        <li>
                            <a href="#">
                                Sign In <span class="glyphicon glyphicon-log-in"></span>
                            </a>
                        </li>
                    </ul>
                </div>
            </div>
        </div>
        <div class="container-fluid">
            <div id="content">
                <!-- Render the content -->
                [% content %]
            </div>
        </div>
        <!-- These javascript files are loaded here (and not at the head) so that pages load
faster -->
        <!-- JQuery -->
        <script src="/static/jquery/jquery-1.11.0.min.js"></script>

        <!-- Bootstrap javascript -->
        <script src="/static/bootstrap/js/bootstrap.min.js"></script>
    </body>
</html>
```

After refreshing your browser, you should be able to see a navbar:

Allele Database Home About Sign In

Allele List

Allele Type	Sequence
1	ATGACAGATGAAGACGATGACGAGATGACGATATAGACGATA
2	TAGACGATGACAGAGATTATAGACGATAGACGCGACGAGACGATGACGGACCCGAGTA
2	TAGACGATGACAGAGATTATAGACGATAGACGCGACGAGACGATGACGGACCCGAGTA
4	ATACGAGATGACGATGACGATGA
4	ATACGAGATGACGATGACGATGA

Next, you may want to add a sidebar. First, create a directory called bootstrap-custom-css in AlleleDatabase/root/static:

```
mkdir bootstrap-custom-css
```

Next, create a file called bootstrap-dashboard.css in the boostrap-custom-css directory with the following contents (this css file can be found by viewing the page source of this site: http://getbootstrap.com/examples/dashboard/):

```
/*
 * Base structure
 */

/* Move down content because we have a fixed navbar that is 50px tall */
body {
  padding-top: 55px;
}


/*
 * Global add-ons
 */

.sub-header {
  padding-bottom: 10px;
  border-bottom: 1px solid #eee;
}


/*
 * Sidebar
 */

/* Hide for mobile, show later */
.sidebar {
  display: none;
}
@media (min-width: 768px) {
  .sidebar {
    position: fixed;
    top: 51px;
    bottom: 0;
    left: 0;
    z-index: 1000;
    display: block;
    padding: 20px;
    overflow-x: hidden;
    overflow-y: auto; /* Scrollable contents if viewport is shorter than content.
*/
    background-color: #f5f5f5;
    border-right: 1px solid #eee;
  }
}
```

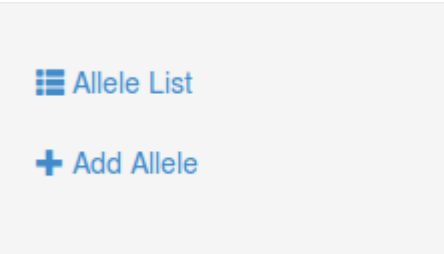
```
/* Sidebar navigation */
.nav-sidebar {
  margin-right: -21px; /* 20px padding + 1px border */
  margin-bottom: 20px;
  margin-left: -20px;
}
.nav-sidebar > li > a {
  padding-right: 20px;
  padding-left: 20px;
}
.nav-sidebar > .active > a {
  color: #fff;
  background-color: #428bca;
}


/*
 * Main content
 */

.main {
  padding: 20px;
}
@media (min-width: 768px) {
  .main {
    padding-right: 40px;
    padding-left: 40px;
  }
}
.main .page-header {
  margin-top: 0;
}


/*
 * Placeholder dashboard ideas
 */

.placeholders {
  margin-bottom: 30px;
  text-align: center;
}
.placeholders h4 {
  margin-bottom: 0;
}
.placeholder {
  margin-bottom: 20px;
}
.placeholder img {
  display: inline-block;
  border-radius: 50%;
}
```

Next you need to import this file in bootstrap_wrapper.tt2 and add code to implement the sidebar:

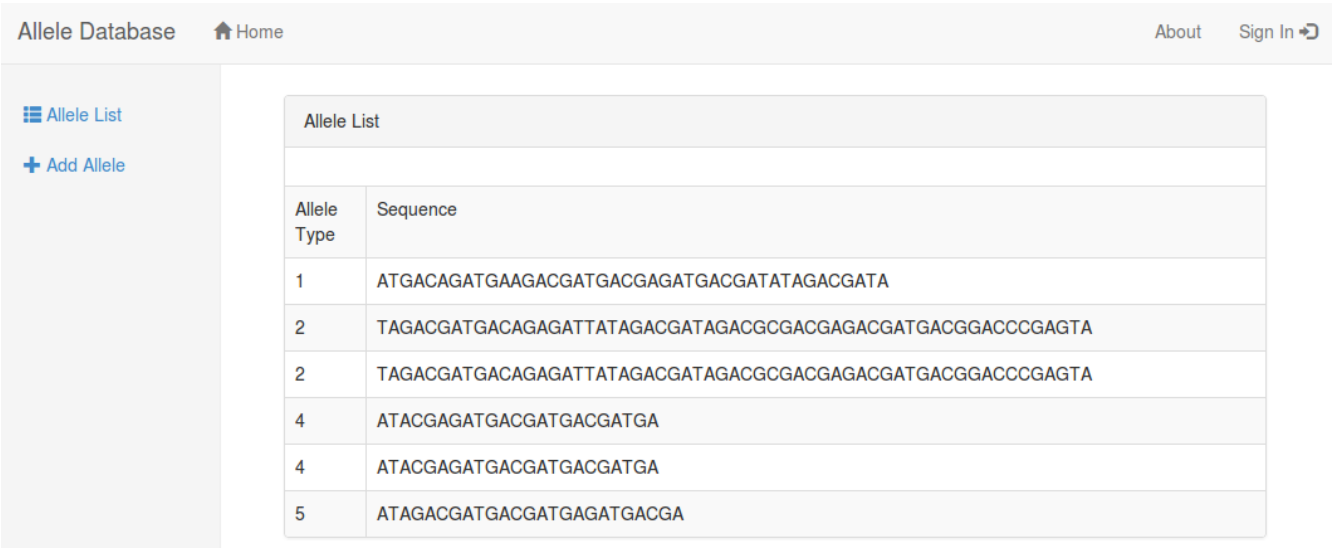
```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="description" content="">
        <meta name="author" content="">

        <!-- Bootstrap core CSS -->
        <link href="/static/bootstrap/css/bootstrap.css" rel="stylesheet">

        <!-- Bootstrap component CSS -->
        <link rel="stylesheet" href="/static/bootstrap-custom-css/bootstrap-dashboard.css">
    </head>
    <body>
        <div class="navbar navbar-default navbar-fixed-top" role="navigation">
            <div class="container-fluid">
                <div class="navbar-header">
                    <a class="navbar-brand" href="[%
c.uri_for(c.controller('Allele').action_for('list')) %]">Allele Database</a>
                </div>
                <div class="collapse navbar-collapse">
                    <ul class="nav navbar-nav">
                        <li>
                            <a href="[% c.uri_for(c.controller('Allele').action_for('list')) %]">
                                <span class="glyphicon glyphicon-home"></span> Home
                            </a>
                        </li>
                    </ul>
                    <ul class="nav navbar-nav navbar-right">
                        <li>
                            <a href="#">
                                About
                            </a>
                        </li>
                        <li>
                            <a href="#">
                                Sign In <span class="glyphicon glyphicon-log-in"></span>
                            </a>
                        </li>
                    </ul>
                </div>
            </div>
        </div>
        <div class="container-fluid">
            <div class="row">
                <div class="col-sm-2 sidebar">
                    <ul class="nav nav-sidebar">
                        <li>
                            <a href="[% c.uri_for(c.controller('Allele').action_for('list')) %]">
                                <span class="glyphicon glyphicon-th-list"></span> Allele List
                            </a>
                        </li>
                        <li>
                            <a href="[% c.uri_for(c.controller('Allele').action_for('add')) %]">
                                <span class="glyphicon glyphicon-plus"></span> Add Allele
                            </a>
                        </li>
                    </ul>
                </div>
            </div>
```

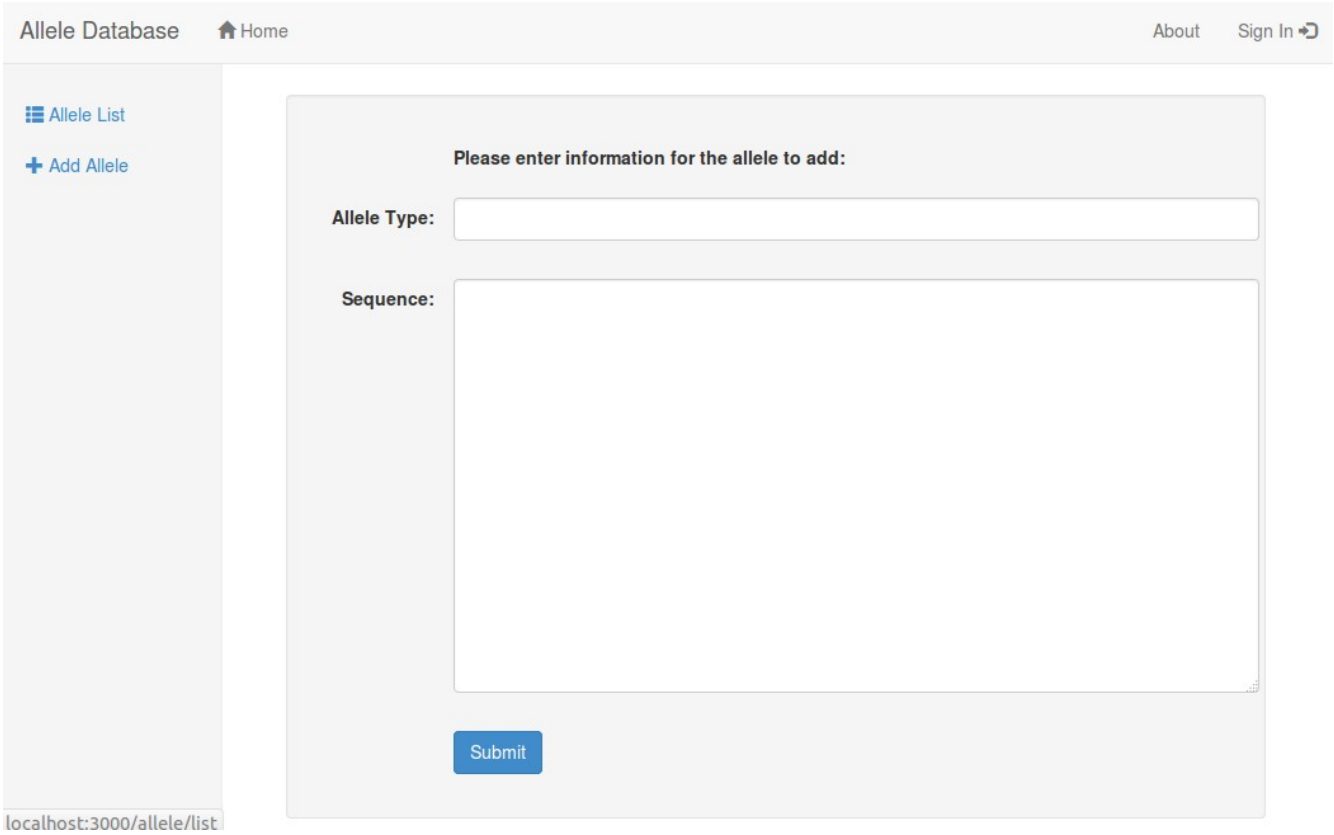
```
            <div id="content" class="col-sm-10 col-sm-offset-2 main">
                <!-- Render the content -->
                [% content %]
            </div>
        </div>
        <!-- These javascript files are loaded here (and not at the head) so that pages load faster -->
        <!-- JQuery -->
        <script src="/static/jquery/jquery-1.11.0.min.js"></script>

        <!-- Bootstrap javascript -->
        <script src="/static/bootstrap/js/bootstrap.min.js"></script>
    </body>
</html>
```

After refreshing your browser you should be able to see a sidebar:



It is now easy to navigate between listing alleles and adding alleles:





This concludes the Allele Database tutorial. After completing the exercises described below, you should be able to get started on implementing new functionality for NG-STAR.

Exercises

1. Implement functionality to Edit Alleles.
2. Implement functionality to Delete Alleles.
3. Develop server side validation for adding, editing and deleting alleles. For example, users should not be able to add an allele that is already in the database.