**3.1 Test Coverage**

**3.1.1 Tested Items**

**3.1.1.1 Unit Testing:**

Unit testing focuses on testing small units of code at a time. It breaks the system down into its individual classes or methods and tests them each separately. Unit testing is important for making sure each piece of an application works as expected, so that these pieces can then be combined with other parts of the application in order to ultimately have a full working system.

Here, Unit testing was done using PHPUnit 5.3. We performed unit tests on two of the model classes in our system, LoginForm and UserSchedule. For each class, a PHPUnit test class was written that included a test for each method in the model class, to ensure that every method works as expected.

**3.1.1.2 Requirements Testing:**

Requirements testing breaks a system down based on its requirements, and tests each one of them to ensure they function as expected. It is important for checking which requirements are met and which are not met. Here, each requirements test case includes a test ID, description, expected output, actual output and result. If the actual output matches the expected output, the result is a *pass*. If not, the result is a *fail*. For any failed tests, we can return to the source code used to implement this requirement and identify the bugs that caused it to fail.

**3.1.1.3 Stress Testing:**

Stress testing is used to test an application’s performance under extreme conditions of usage, such as a high volume of users or a large data load. It is done in order to determine the system’s robustness and reliability, and helps to identify issues that may only arise under stressful conditions.

There are a number of tools available for stress testing a web application that simulate extreme conditions. Here, we used Apache JMeter, a Java application that works by creating multiple threads that simultaneously access a given web server via HTTP requests. We tested stress conditions for both the login and schedule generation functions of the system.

**3.1.1.4 Security Testing:**

Security testing is crucial for any software system to ensure that the system can protect itself against malicious attacks. We used two tools to test security. The first was SQL Inject Me, a Firefox plugin that sends SQL injection strings through the forms of a webpage and checks for any database errors that may occur as a result. The second was Nikto, a web server scanner that uses a Perl script to perform multiple tests against a given server, including checking for dangerous files and checking for outdated versions.

**3.1.2 Untested Items of Interest**

For the unit tests, we were only able to test 2 of the 11 models in our system. This was due to tight time constraints, as it took a while to set up PHPUnit and write the test code. We chose to test the LoginForm and UserSchedule models because these classes are very important to the system’s functionality; every user needs the LoginForm to access the system, and UserSchedule is needed to generate a schedule. With more time, we would have ideally performed unit tests on the other 9 models in our system (CompletedCourses, ContactForm, Course, PreferenceForm, Prerequisite, Section, Subsection, User and UserSchedules), as well as the controller classes (CourseController, ProfileController, SchedulerController, SiteController and UserController).

For the requirements tests, we tested all requirements except the ones that were scoped out. Scoped out requirements included *create a course sequence*, *weekly schedule view* and *add a course*. With more time, we would have liked to implement these requirements and perform tests on them.

**3.2 Test Cases**

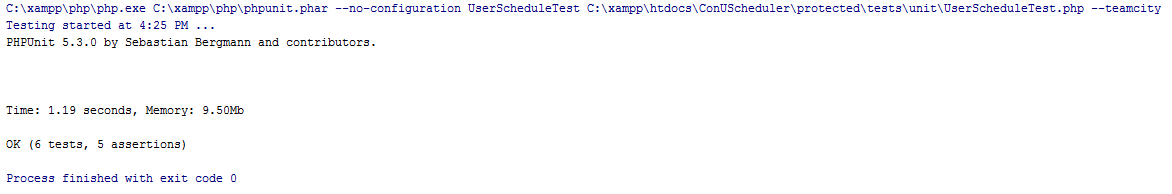
**3.2.1 Unit Testing**

**3.2.1.1 Code for UserScheduleTest:**

**<?php  
require\_once**(**'../../../protected/models/UserSchedule.php'**);  
$yiit=**'../../../framework/yiit.php'**;  
**require\_once**($yiit);  
  
**class** UserScheduleTest **extends** CDbTestCase {  
 **protected $fixture**;  
 **protected $db**;  
  
 **protected function** setUp() {  
 $this->**db** = **new** CDbConnection(**'mysql:host=conuscheduler.ddns.net;dbname=soen341'**,**'soen341'**,**'soen341'**);  
 $this->**db**->**active**=**true**;  
 CActiveRecord::*$db*=$this->**db**;  
 $this->**fixture** = **new** UserSchedule;  
 }  
  
 **protected function** tearDown() {  
 **if**($this->**db**)  
 $this->**db**->**active**=**false**;  
 }  
  
 **public function** testTableName() {  
 $expected = **'user\_schedule'**;  
 $results = $this->**fixture**->tableName();  
 $this->assertEquals($expected,$results);  
 }  
  
 **public function** testRules() {  
 $expected = **array**(  
 **array**(**'scheduleID, courseID, sectionID, subsectionID, year'**, **'required'**),  
 **array**(**'scheduleID, courseID, sectionID, subsectionID, year'**, **'numerical'**, **'integerOnly'**=>**true**),  
 **array**(**'ID, scheduleID, courseID, sectionID, subsectionID, year'**, **'safe'**, **'on'**=>**'search'**),  
 );  
 $results = $this->**fixture**->rules();  
 $this->assertEquals($expected,$results);  
 }  
  
 **public function** testRelations() {  
  
 $expected = **array**(  
 **'scheduleID'** => **array**(UserSchedule::***BELONGS\_TO***, **'UserSchedules'**, **'ID'**)  
 );  
 $results = $this->**fixture**->relations();  
 $this->assertEquals($expected,$results);  
 }  
  
 **public function** testAttributeLabels() {  
 $expected = **array**(  
 **'ID'** => **'ID'**,  
 **'scheduleID'** => **'Schedule'**,  
 **'courseID'** => **'Course'**,  
 **'sectionID'** => **'Section'**,  
 **'subsectionID'** => **'Subsection'**,  
 **'year'** => **'Year'**,  
 );  
 $results = $this->**fixture**->attributeLabels();  
 $this->assertEquals($expected, $results);  
 }  
  
 **public function** testSearch() {  
 $stub = $this->getMockBuilder(**'CDbCriteria'**)  
 ->disableOriginalConstructor()  
 ->getMock();  
 $stub->expects($this->any())  
 ->method(**'compare'**)  
 ->will($this->returnValue(**'true'**));  
 }  
  
 **public function** testModel() {  
 $stub = $this->getMockBuilder(**'UserSchedule'**)  
 ->disableOriginalConstructor()  
 ->getMock();  
 $stub->expects($this->any())  
 ->method(**'model'**)  
 ->will($this->returnValue($this->**fixture**));  
 $this->assertNotNull($stub);  
 }  
}

**Results:**

UserScheduleTest passed 6/6 tests, shown in the screenshot below:

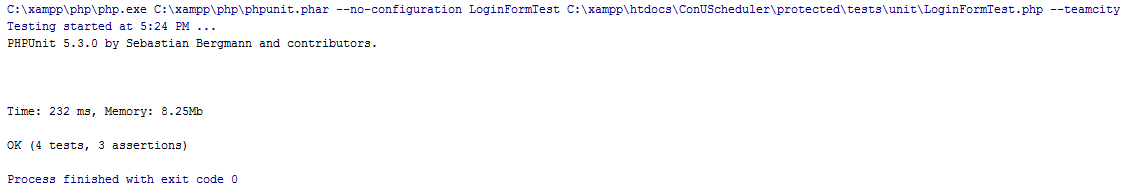


**3.2.1.2 Code for LoginForm test:**

**<?php  
  
require\_once**(**'../../../protected/models/LoginForm.php'**);  
**require\_once**(**'../../../protected/components/UserIdentity.php'**);  
$yiit=**'../../../framework/yiit.php'**;  
**require\_once**($yiit);  
  
**class** LoginFormTest **extends** CTestCase  
{  
 **protected $fixture**;  
 **protected function** setUp() {  
 $this->**fixture** = **new** LoginForm;  
 $this->**fixture**->**username** = **"erin"**;  
 $this->**fixture**->**password** = **"soen341"**;  
 $this->**fixture**->**rememberMe** = **"0"**;  
 }  
  
 **protected function** tearDown() {  
 }  
  
 **public function** testRules() {  
 $stub = **array**(**array**(**'username, password'**, **'required'**),  
 **array**(**'rememberMe'**, **'boolean'**),  
 **array**(**'password'**, **'authenticate'**));  
 $results = $this->**fixture**->rules();  
 $this->assertEquals($stub,$results);  
 }  
  
 **public function** testAttributeLabels() {  
 $stub = **array**(**'rememberMe'**=>**'Remember me next time'**,);  
 $results = $this->**fixture**->attributeLabels();  
  
 $this->assertEquals($stub, $results);  
 }  
  
 **public function** testAuthenticate() {  
 $stub = $this->getMockBuilder(**'UserIdentity'**)  
 ->disableOriginalConstructor()  
 ->getMock();  
 $stub->expects($this->any())  
 ->method(**'authenticate'**)  
 ->will($this->returnValue(**'true'**));  
 }  
  
 **public function** testLogin\_2() {  
 $stub = $this->getMockBuilder(**'LoginForm'**)  
 ->disableOriginalConstructor()  
 ->getMock();  
 $stub->expects($this->any())  
 ->method(**'login'**)  
 ->will($this->returnValue(**'true'**));  
 $this->assertEquals(**'true'**, $stub->login());  
 }  
}

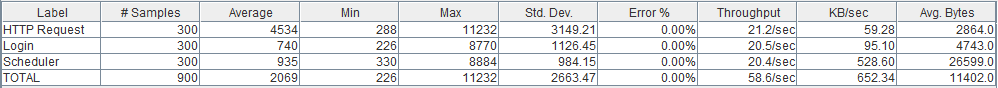
**Results:**

LoginFormTest passed 4/4 tests, shown in the screenshot below:



**3.2.3 Stress Testing**

We ran the JMeter tool with 300 simultaneous users and a 5-second ramp-up period (time to get all threads started). There were zero errors, and we obtained the following summary report:

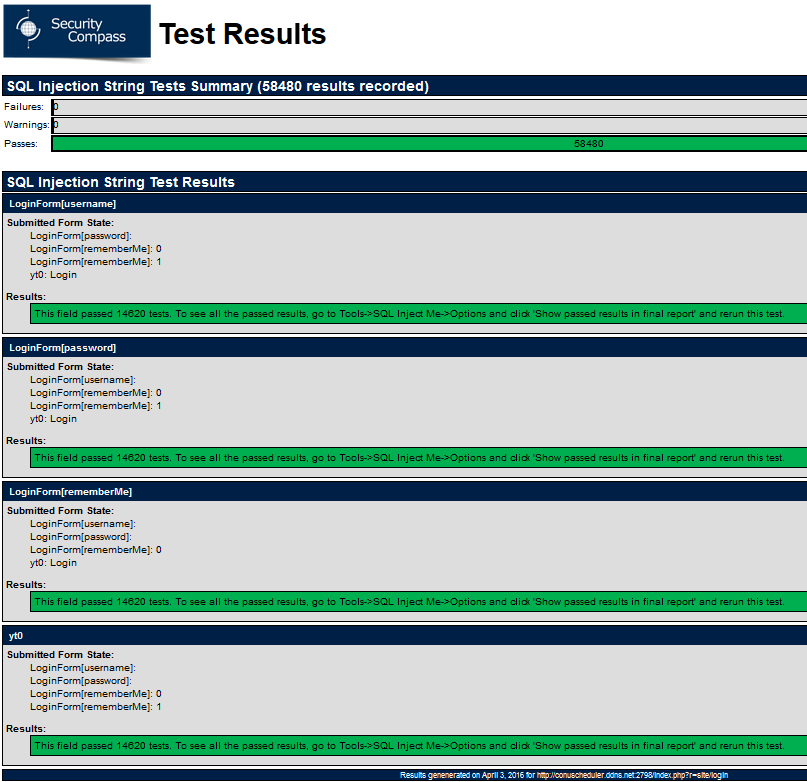


**3.2.4 Security Testing**

**3.2.4.1 SQL Inject Me results:**

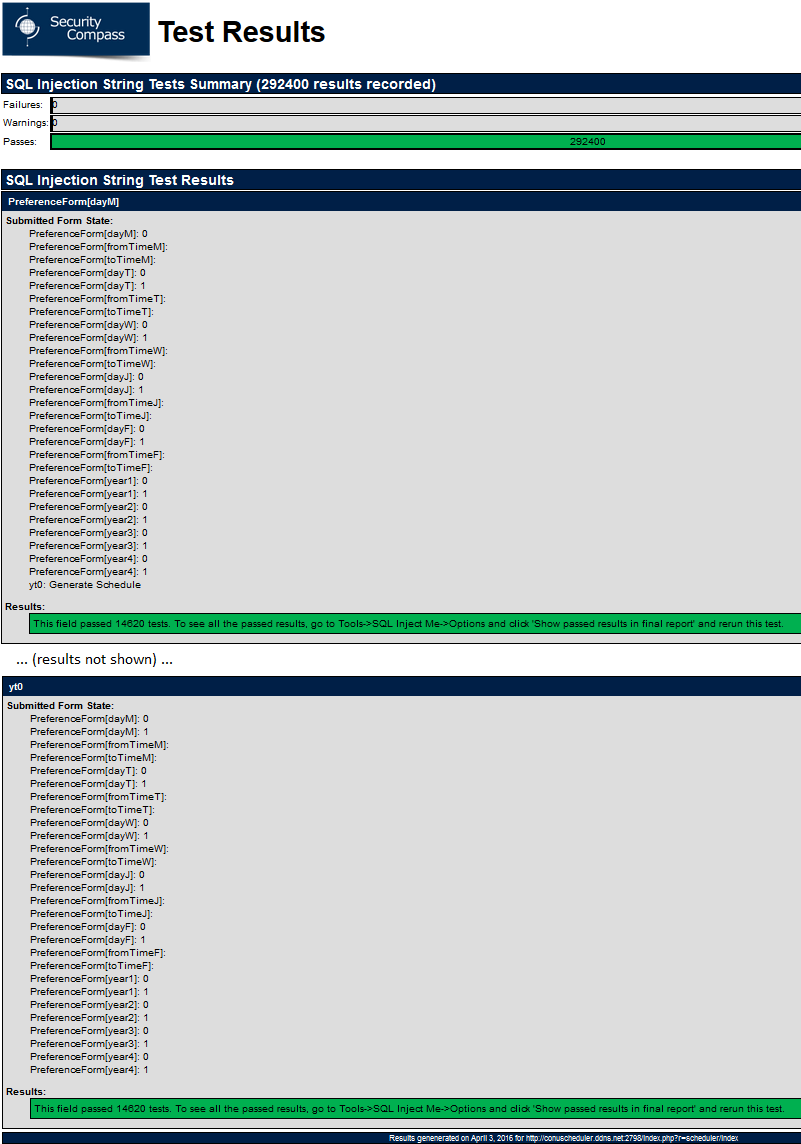
**LoginForm:**

The LoginForm passed all 58480 tests, shown in the screenshot below:



**PreferenceForm:**

The PreferenceForm passed all 292400 tests, shown in the screenshot below:



**3.2.4.2 Nikto results:**

