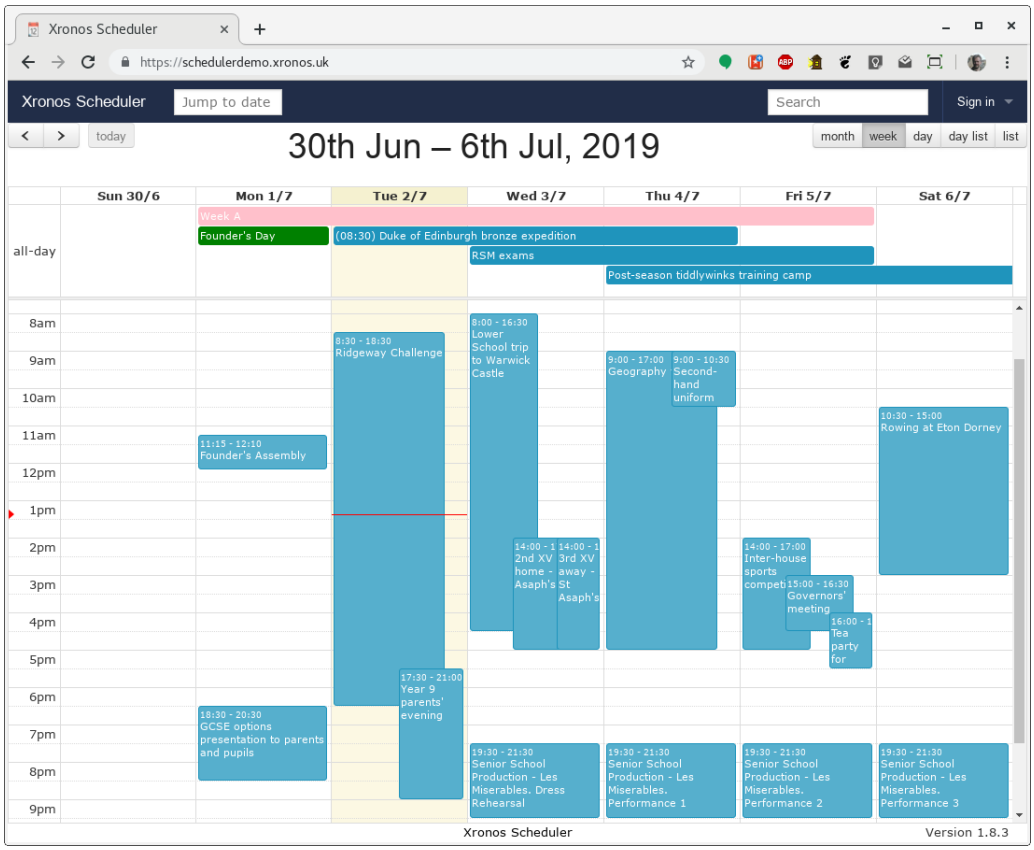
Xronos Scheduler Installation Guide

Release 1.17.2

John Winters

Oct 15, 2024

CONTENTS



Xronos Scheduler - installation instructions.

Xronos Scheduler is a web application, written using the Ruby on Rails web application framework. As such, it can be deployed to any of the many hosting services which provide support for such applications.

These instructions are intended for those who want instead to set up a suitable hosting platform for themselves - either a dedicated physical machine, or some kind of virtual server.

In these circumstances, the recommended operating system is Debian GNU/Linux. It should be relatively straightforward to use any other version of Linux, or MacOS.

You may already have your own web application infrastructure, in which case you should be able to slot Scheduler into it.

These instructions assume you are going to use a fresh, blank installation of Debian GNU/Linux, with just the bare operating system installed.

Download this guide as a PDF

CHAPTER

ONE

HARDWARE REQUIREMENTS

Xronos Scheduler has pretty modest hardware requirements by modern standards. As always, more RAM is better, and the use of an SSD instead of a mechanical hard disk drive will significantly improve database response times.

1.1 Minimum

- 2G RAM
- 10G Hard disk
- Reasonably modern CPU (e.g. 1.8GHz) with at least 2 cores

1.2 Recommended

- 4G RAM
- 20G SSD
- CPU with multiple cores

More RAM is always nice, but beyond 4G there is little more that Scheduler can do with it. On a long running system with 8G of RAM in a large school the actual usage barely rises above 4G until the system is backed up. Once the whole database, application and files have been cached in RAM, the usage gets to about 4G, and then no more can be used.

Multiple CPU cores can do a lot to improve response times when many people are using Scheduler at once.

Scheduler will actually run very acceptably on a Raspberry Pi 4 (4G version) provided you add a USB3 SSD as the system drive. See https://scheduler.rpi4.uk/

CHAPTER

TWO

OPERATING SYSTEM

The recommended operating system for Xronos Scheduler is Debian GNU/Linux version 10 (Buster).

2.1 Download

Download the "Small CDs or USB sticks" image for amd64 (or i386 if you have only a 32-bit processor) for Debian GNU/Linux version 10.

Note: If you are doing a fresh installation of Debian GNU/Linux onto a normal PC or virtual machine, then you almost certainly want to choose the "amd64" version of the installer. This is the 64 bit version for both Intel and AMD processors. Use the "i386" version only for old processors which are restricted to 32 bit.

2.2 Install

If you are using a physical machine for your installation, you will need to write the image to a CD or USB stick. If you installing to a Virtual Machine, you should be able to use the image file directly as an emulated device.

Boot your machine (or VM) with the installation image and follow the instructions. At each choice, take the default offered except at the stage of choosing the packages to install. At that point, un-tick "Debian Desktop Environment" and tick "SSH Server". There's no point in having a graphical desktop environment on a headless server - it's a waste of resources - but you will need to have an SSH server running in order to be able to connect to it.

Many VM providers will bootstrap your VM for you with the operating system of your choice. In this case, just ask for the current Debian GNU/Linux.

2.3 Initial packages

Once you have a running system, you will need some basic extra packages and a user account under which to run the application.

It is assumed you have root access to the system and are going to create a user called "scheduler" to run the software.

Note: You can call the user anything you want, but don't try to run the application as root. These instructions assume that the user will be called "scheduler".

As root, do the following:

```
# apt install sudo vim curl git libicu-dev memcached dirmngr libmagick++-dev
# apt install nodejs npm
# apt install mariadb-server mariadb-client libmariadb-dev
# adduser scheduler
# adduser scheduler sudo
# adduser scheduler staff
```

These lines install the necessary packages, add the scheduler user, and then put that user in a couple of necessary groups.

It's a good idea to run the command:

```
# mysql_secure_installation
```

After installing mariadb above. Accept all the default options to tighten up your mariadb installation.

Log out as root and log back in as your chosen user.

Note: The following instructions frequently ask you to edit a file. To do this you need some sort of text editor, and as this is a headless server, it's going to need to be a text-mode one, not a graphical one.

You may have noticed that vim was installed as one of the basic packages, so if you're happy with vim then do use that. It's a very powerful editor, but perhaps has a bit of a steep initial learning curve. For small edits, and if you don't already have a preference for a particular editor, "nano" is an editor which is very easy to get started with.

If you'd like to learn vim, you'll find an interactive tutorial on your system which can be invoked with the command "vimtutor".

CHAPTER

THREE

RUBY

Warning: All the following should be done as your chosen user - scheduler - not as root.

If you are still logged in as root (your command line prompt is a # symbol) then log out now and log back in as scheduler. You'll note that your command line prompt changes to a $ symbol.

3.1 Ruby Version Manager (RVM)

In order to make sure we are using a compatible set of packages, we use the Ruby Version Manager (RVM) to install and manage them.

By using RVM, one can have multiple copies of Ruby, Rails and supporting packages installed on the same machine, and be sure of always using exactly the one intended.

You will find instructions for installing RVM on the page referenced above, but in essence it comes down to just two commands.

```
$ gpg --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3␣
↪7D2BAF1CF37B13E2069D6956105BD0E739499BDB
$ \curl -sSL https://get.rvm.io | bash -s stable
```

This incidentally is why you installed curl in the previous step.

Note: It is just possible that the gpg command above will fail, apparently due to a compatibility problem between dirmngr and some DNS servers. It's not a common occurence, but it has been known to happen. If it happens to you, there is a workaround.

```
$ echo "standard-resolver" >>~/.gnupg/dirmngr.conf
$ killall dirmngr
```

and then try again.

Once the installation has completed, log out and then back in again in order to pick up your modified environment.

Note: If you are doing all this not on a headless server but on a workstation with the default Debian Gnome desktop, there's one little extra wrinkle.

To get rvm to work properly you need to edit your terminal preferences and under the "Command" tab, tick "Run command as a login shell". If you don't do that then rvm doesn't get to initialize itself properly when you start up an interactive terminal.

3.2 Ruby

The application currently uses Ruby version 2.6.6. Ruby 2.7 has been released but there are some deprecations within it which clash with Rails 5. Rails 5 applications will run with Ruby 2.7, but produce warnings as a result of these deprecations.

To install the necessary Ruby, use the following command. It can take a few minutes to complete, depending on the speed of your processor.

```
$ rvm install ruby-2.6.6
```

You will be prompted for your user's password (not the root password) in order to install extra required operating system packages.

CHAPTER

FOUR

SCHEDULER

Now you've set up all the necessary pre-requisites, you can install the Scheduler application itself.

Note: There is nothing to stop you fetching the Scheduler code first if you just want to have a look at the source etc. You won't however be able to run the application (or do the "bundle install" step below) until you have all the pre-requisites in place.

4.1 Download

The Scheduler source code resides on Github.

Before you download it, think about where you're going to put the files in your user's home directory. All the development work is done using a directory name of:

~/Work/Coding/scheduler

so unless you have a good reason not to, that might be a good place for it.

```
$ cd
$ mkdir -p Work/Coding
$ cd Work/Coding
$ git clone https://github.com/XronosSchedulingLtd/scheduler.git
$ cd scheduler
$ bundle install
```

These commands create the necessary directory structure, fetch the application files from Github, and then finally fetch and install the necessary support software.

4.2 Create database

Before you can run up the application, you need some databases to be created within MariaDB. A script is provided to assist in setting these up. Under the scheduler directory it will be found as:

support/setupmysql

Edit the script and you will see an environment variable to set.

```
DATABASE_PASSWORD="not set"
```

The password is up to you but make it something secure. Then run the script with

```
$ sudo support/setupmysql
```

The sudo part of this command line will prompt you for your user's password.

Edit the file:

config/database.yml

and set the password there to the same as you chose to put in the DATABASE_PASSWORD variable.

You can then create the database tables and give them some initial sample data with the following commands.

```
$ rake db:schema:load
$ rake db:seed
```

4.3 Run it

And now you should be in a position to run up the application in demonstration mode with the following command:

```
$ rails s -b 0.0.0.0
```

Point a web browser to http://<your host>:3000 and you should see the Scheduler demonstration site. The application is running in development mode with a copy of the data used on the demonstration site.

Note: The "-b 0.0.0.0" bit is needed because without it the rails server listens only on address 127.0.0.1. This is fine if you're running your web browser on the same machine as the rails application, and can thus access it at http://localhost:3000 but fails when you're setting up a headless server and need to access it from a web browser running on a different machine.

You can log in as one of the three demonstration users using the menu at the top right.

4.4 A little extra

You now have the application running in development mode but before you go, there are a few more things to set up.

4.4.1 Credentials

For security, Rails applications use some secret random strings to make sure that no-one can break in on the sessions between server and web browser. These aren't provided as part of the Scheduler distribution because they need to be unique to your system. Create them with the following command:

```
$ EDITOR=vi rails credentials:edit
```

(Set EDITOR to your preferred editor if it isn't vi.)

As there are currently no credentials set up on your system a new set will be created for you and displayed in the editor. Simply save the file, exit the editor and all will be set up.

The credentials will be stored encrypted in a file config/credentials.yml.enc and the encryption key will be saved in config/master.key.

4.4.2 Ruby environment

All the utilities and particularly the cron jobs within the system need to know which version of Ruby and which gemset to use. Rather than editing them all to specify "ruby-2.6.6@scheduler" they expect an alias to be in place. Do the following:

```
$ rvm alias create scheduler ruby-2.6.6@scheduler
```

4.4.3 Environment variables

There are also a couple of files which will contain environment variables to aid in running the application and associated jobs. These live in an etc directory under your Scheduler user's home directory. Copy them as follows:

```
$ mkdir ~/etc
$ cp support/whichsystem ~/etc
$ cp support/authcredentials ~/etc
```

You will configure the contents of these files a little more later.

Warning: Don't miss these steps. Without them, the application will fail to start in production mode and your cron jobs will fail.

CHAPTER

FIVE

NEXT STEPS

At this point you have all the Scheduler software installed and running, but:

- It's running in development mode, from the command line.
- It's running with demonstration data.

Obviously you want it running in production mode using a proper web server, and you want it to have data specific to your school, rather than made up stuff.

It is necessary to perform the next three steps in order because of the constraints of authentication.

- You need a working web server before you can set up https.
- You need https before you can enable authentication.

CHAPTER

SIX

PRODUCTION MODE

So far, you've set up Scheduler running in development mode. This means it is started from the command line, and runs on a non-privileged port (3000 by default) on your computer. It is also producing far more logging information than you would want in a live installation.

To make it work as a proper server, you need a web server and the means to get Scheduler to start up silently in the background each time the server is re-booted.

If you still have Scheduler running in development mode, then shut it down by typing Ctrl-C in the relevant terminal window.

6.1 Overview

The preferred configuration for Scheduler in a production environment is to use Puma as the application server, running behind Nginx which functions as the web server.

Requests arriving from clients go first to Nginx which handles the ones which it can (requests for static resources - JavaScript files, stylesheets and images) and passes on the rest to Puma. Puma runs the Scheduler application.

6.2 IP address

In order to be accessible, your server will need to be allocated a stable IP address. How this is done will depend on your organisation's existing infrastructure. If you are using a commercial Virtual Private Server (VPS) then they should be able to tell you what your IP address is.

6.3 Domain name

Similarly, you will need a domain name for your server. Typically this will be something like "scheduler.myschool.org.uk". Again, it will need to be allocated by your organisation's IT administration, and pointed to the IP address mentioned above.

6.4 Scheduler preparation

A little more preparation is needed for Scheduler to run in production mode. When it's running in development mode, resources are compiled on the fly and sent to the web browser. This means that things can be changed and the change takes place immediately, without even restarting the Scheduler process.

In production mode, things aren't meant to change, so they're prepared in advance and then sent out directly by Nginx.

To switch your system to running production mode by default, edit the file ~/etc/whichsystem and change the RAILS_ENV environment from a value of "development" to a value of "production".

At the same time make sure the SCHEDULER_DIR variable in the same file is set to where you have installed Scheduler.

Then make those settings effective in your current session by typing:

```
. ~/etc/whichsystem
```

Note the full stop at the start of that line.

Next you need to pre-compile all the static resources of your application. Type the following:

```
$ cd $SCHEDULER_DIR
$ rake assets:precompile
```

If you haven't already set up your production database (depending on whether you have done the *Real data* step) then create a production database with:

```
$ rake db:schema:load
$ rake db:seed
```

Scheduler is now ready to run in production mode.

6.5 Puma

Puma is a Ruby Gem and it has already been installed as part of your installation of Scheduler. When you started Scheduler in development mode what you actually started was Puma, which in turn started up your application.

For production mode, what is needed is for Puma to start automatically as soon as your system boots.

Take a look at the file config/puma-live.rb This contains Puma's configuration for live running. In particular, you might want to adjust the first setting - "workers 4" - to match the number of CPU cores in your system.

If you have used a non-standard installation path for Scheduler then edit the file support/puma.service and change the WorkingDirectory, EnvironmentFile and ExecStart lines to match your installation. Then copy it to to /etc/systemd/system:

```
$ sudo cp support/puma.service /etc/systemd/system
```

This gives the system the information which it needs to start up Puma and thus Scheduler. Now it's just a case of telling systemd that we want Puma to start:

```
$ sudo systemctl enable puma.service
$ sudo systemctl start puma.service
```

The first line tells systemd that in general you want puma to start when the system boots, and the second one tells it to start puma *now*.

You can check that puma is running by looking in the directory shared/log where you should find two log files which it has created.

6.6 Nginx

The preferred web server for Scheduler is Nginx, although you can also run it with Apache if you prefer. Instructions for using Apache are not included here.

Nginx can be installed easily from the standard Debian repositories with the command:

```
$ sudo apt install nginx
```

Once that command completes, you will have Nginx running on your system.

Next you need to tell Nginx how to pass requests on to Puma. Again, a suitable configuration file is provided in the support directory.

Edit the file support/scheduler to set:

- Your server's domain name
- The full path of where you have installed Scheduler (two places)

The relevant bit of the file looks like this:

```
#
#  This tells Nginx where to find our application server.
#
upstream scheduler {
  server unix:///home/scheduler/Work/Coding/scheduler/shared/sockets/scheduler.sock;
}

#
#  Front-end our scheduler server processes
#
server {
  server_name scheduler.myschool.org.uk;
  listen 80;
  listen [::]:80;
  root /home/scheduler/Work/Coding/scheduler/public;

  ...
}
```

If you haven't changed Scheduler's installation directory, then you need change only the server_name in this file. Once you've edited it, install it with:

```
$ sudo cp support/scheduler /etc/nginx/sites-available
$ sudo ln -s /etc/nginx/sites-available/scheduler /etc/nginx/sites-enabled/scheduler
```

and then restart Nginx with:

```
$ sudo service nginx restart
```

You should then be able to access the application by pointing your web browser at http://<your domain name>/

6.7 Log rotation

Because Puma and thus Scheduler are now running permanently in the background their log files will grow monotonically. Debian GNU/Linux includes a program called logrotate which can rotate such log files periodically provided it is told about them.

If you have changed the Scheduler installation directory then edit the file support/logrotate.scheduler to specify your new directory, then install it with:

```
$ sudo cp support/logrotate.scheduler /etc/logrotate.d/scheduler
```

CHAPTER

SEVEN

CONFIGURE HTTPS

Having got your installation up and running using unencrypted connections, you now need to configure https. Running an unencrypted web server is frowned upon, and the authentication services generally won't let you use them for authentication if you are.

7.1 Let's Encrypt

In order to make use of encrypted connections, your web server will need a digital certificate. Your organisation may already have a supplier for such certificates, but if not then you can get them free from the Let's Encrypt project. The following instructions assume you're going to use Let's Encrypt.

7.2 Install Certbot

The process of getting a certificate from Let's Encrypt is enabled by a utility called certbot. This is included in Debian GNU/Linux.

To install it execute the following command:

```
$ sudo apt install certbot python-certbot-nginx
```

7.3 Request a certificate

You should already have Nginx installed and running on your system and accessible from the outside world, albeit only over http. Before Let's Encrypt will issue you with a certificate, they need to be satisfied that you are actually have control of the machine making the request, so their server will need to be able to contact yours to validate the request.

Happily the whole process is pretty thoroughly automated. You run a command on your server, it contacts LetsEncrypt's servers, they make a request back to your Nginx instance to make sure it really is you and then the certificate is issued. The utility even installs it for you.

The command to use is:

```
$ sudo certbot --nginx
```

You will be prompted for an e-mail address (so the certificate issuing authority can contact you if one of your certificates hasn't been renewed and is about to expire) and then to accept LetsEncrypt's terms of service.

The utility will work out your configured domain name from the Nginx configuration file which you set up earlier and then offer to re-direct all requests to https only. Take this option (option 2) and it will modify the Nginx configuration file for your website as needed.

Once the utility has finished you should find that any attempt to access http://scheduler.myschool.org.uk (or whatever your site is called) is automatically redirected to https://scheduler.myschool.org.uk.

The utility also sets up a background job which will automatically renew the certificate before it expires.

CHAPTER

EIGHT

REAL DATA

8.1 Authentication

As soon as you think about setting up real data in your Scheduler system, you have to start thinking about authentication - the process of your users identifying themselves to the system.

In demo mode, no authentication is necessary - you can log in as two different teachers, or as one student, just by selecting the user you want from a pop-down menu.

As soon as you set up a system which isn't in demo mode, this option disappears - you need to authenticate yourself properly.

Scheduler makes use of OmniAuth, a flexible authentication module with a lot of different possible methods of authentication.

Currently only two of these are enabled within Scheduler - Google or Microsoft's Azure. If you need something different, do get in touch - info@xronos.uk

Scheduler defaults to using Google but the following instructions will tell you how to switch this to Microsoft Azure if that's what your school uses.

Note that Google (and probably Microsoft) no longer let you use their authentication unless you are running an https server. It is therefore necessary to configure https before you can set up authentication.

8.2 Seed data

The Scheduler application comes with seed data which sets your system up to look just like the Scheduler demonstration site.

This is handy for initial proof that your installation works, but you need a clean system into which to put your real data.

Edit the file db/seeds.rb and find the following lines

```
seeder = Seeder.new(
  public_title:    "Xronos Scheduler",
  internal_title:  "Scheduler - Lorem Ipsum Academy",
  dns_domain_name: "schedulerdemo.xronos.uk",
  auth_type:       Setting.auth_types[:google_auth]
)

#
#  First, some which are intrinsic to the functioning of the system.
#
```

(continues on next page)

(continued from previous page)

```
seeder.create_essentials

#
#  And the rest fall under a heading of likely to be useful.
#
seeder.create_usefuls

#=============================================================================
#
#  Everything from here on down is demonstration data.  You probably don't
#  want to load it in a new live system.
#
#=============================================================================
```

You'll probably want to change the two titles to something more suitable for your school. The public one is shown to casual visitors, whilst the internal one is shown to users who have logged in.

Typical examples might be "My School Calendar" for the public one, and "Scheduler - My School", for the internal one.

Change "schedulerdemo.xronos.uk" to the domain name of your server, and if you are using Azure for your authentication then change ":google_auth" in the fourth line to ":azure_auth". Then delete the final comment and everything which follows it.

If you now re-run the seeding process, you will have a completely blank system, ready for your data - don't do that yet.

8.3 Initial user

You're going to need to be able to log on to your system, which means you need an initial user.

When a user logs in using your chosen authentication system, Scheduler checks to see whether it recognizes that user (by e-mail address) as a member of staff or pupil. If it does, the new user gets set up with appropriate access to the system.

We thus need at least one staff record in the system to be able to log on.

Still editing the same file - db/seeds.rb - add an extra line at the end like this:

```
seeder.new_staff("Mr", "Able", "Baker", "AB", [], "able.baker@gmail.com")
```

The fields here are as follows:

- Title
- Given name
- Family name
- Initials
- Array of subjects taught (don't worry about this)
- Email address

The crucial one is the e-mail address. Scheduler will compare this with what it gets back from your chosen authenticator, and if the two match then it will treat the user as a member of staff. Obviously, use a real person's name and e-mail address.

Note: Whichever authenticator you choose, anyone with an account on that service can log in to your system. Authorization however is a different step from Authentication. If an arbitrary individual authenticates into your system, they won't have a matching staff or pupil record, and so will be treated as a guest - they will have no more access to the system than if they weren't logged in.

When Mr Able Baker (set up above) logs in, the system will recognize him as a member of staff, and give him whatever permissions staff are configured as having.

With that in place you can then re-seed the database with:

```
$ export RAILS_ENV=production          # If you want to do the production d/b
$ rake db:schema:load
$ rake db:seed
```

And you'll have a nice blank database ready to use with some fundamental records already in place.

8.4 Configure authentication

8.4.1 Via Google

To set up Google Authentication, you'll need an account with Google. Visit the Google Developers Console and log in as your chosen user.

The first thing which you need to do is create a Project. At the top left of the screen, just to the right of "Google APIs" there is a pull-down menu - currently empty. Click on that, and then on the + button to create a new project.

Give your project a name - e.g. Scheduler - and then save it.

Click on the pull-down menu again and choose Scheduler as your current project. A pop-up will appear, prompting you to create credentials. Click on the "Create credentials" button and choose "OAuth client ID".

At this point you will then be told you need to configure your application's consent screen. Click on the "Configure consent screen" button and fill in the "Product name shown to users" field. You would probably want to call it something like "<My school's name> Scheduler". The idea is to make it clear to users exactly what system they are logging in to.

Save that form and you'll be taken back the credentials creation process. Choose "Web application" as the application type, give it a name (e.g. "Scheduler" again) and then you need to provide a couple of URLs.

The first of these is the address that requests to Google will be coming *from* - i.e. the basic URL of your web server. Using our example address that would be:

```
https://scheduler.myschool.org.uk
```

The second field is to tell Google where to redirect users who have completed the login process. This would be:

```
https://scheduler.myschool.org.uk/auth/google_oauth2/callback
```

You can put more than one value in each of these fields. If you're going to be doing development work on a local system you might want to add also

```
http://localhost
```

and

```
http://localhost/auth/google_oauth2/callback
```

Similarly, if you want to be able to use real authentication before you have set up https, you might want to add http: versions of the above.

Altogether, that would give you the following authorised origins:

```
https://scheduler.myschool.org.uk
http://localhost
```

and the following authorised redirects.

```
https://scheduler.myschool.org.uk/auth/google_oauth2/callback
http://localhost/auth/google_oauth2/callback
```

Obviously, use your own domain name and not the sample one given here.

When you click the "Create" button, a fresh pop up window will appear telling you the Client ID and Client Secret for your application.

Copy both of these into your ~/etc/authcredentials file which you created earlier. By default it contains this:

```
#
#  If you are using Google Authentication then uncomment and fill
#  in the three following lines.
#
#GOOGLE_CLIENT_ID=""
#GOOGLE_CLIENT_SECRET=""
#export GOOGLE_CLIENT_ID GOOGLE_CLIENT_SECRET
```

Put the newly acquired items between the quotation marks and uncomment the three relevant lines.

You need to enable the "Contacts API" in the Google Developers Console in order for authentication to work.

Warning: It seems it can take a few minutes for your new credentials to propagate to Google's servers. When you first create them, try waiting 5 minutes before attempting to log in to the Scheduler application.

You should now be able to log in to your Scheduler installation using the gmail address of the staff member which you set up earlier.

8.4.2 Via Microsoft Azure

The process for setting up authentication through Microsoft's Azure is much the same as for Google, although you need specify only the callback URL. The callback to use is:

```
https://scheduler.myschool.org.uk/auth/azure_activedirectory_v2/callback
```

although obviously change "scheduler.myschool.org.uk" to the fully qualified domain name of your server.

And if you want to use a development system as well:

```
http://localhost:3000/auth/azure_activedirectory_v2/callback
```

You need three pieces of information from your Azure instance's console:

- A Client Id

- A Client Secret
- A Tenant Id

As before, these go in the file ~/etc/authcredentials which you created earlier. Look for the following lines:

```
#
#  Similarly, if you are using Microsoft's Azure for authentication
#  then uncomment and fill in the following four.
#
#  Note that Microsoft have set a trap for users of the AZURE_CLIENT_SECRET.
#  When you create one it is shown with both an ID and a VALUE.  It's
#  tempting to use the ID, but what you need is the VALUE.
#
#AZURE_CLIENT_ID=""
#AZURE_CLIENT_SECRET=""
#AZURE_TENANT_ID=""
#export AZURE_CLIENT_ID AZURE_CLIENT_SECRET AZURE_TENANT_ID
```

Warning: As noted in the comment above, when you create a Client Secret in the Microsoft console, it has both an ID and a VALUE. What you want here is the VALUE.

8.4.3 Common to both

To be able to pick up the secrets when Scheduler is run from the command line, edit the scheduler user's ~/.profile file and add the following lines.

```
. $HOME/etc/authcredentials
```

If you're still running the application in development mode then you'll need to log out and back in again for these to take effect. If you're running in production mode then restart the server with "sudo systemctl restart puma.service".

8.5 Administrator access

Having logged in as your new user, you should find that you are recognized as a member of staff and can create events etc.

Note: If you can log in (your name appears at the top right) but the rest of the screen looks just the same as it did before - no menus or anything - then there is probably a mismatch between the e-mail address which you gave your staff member above, and the e-mail address which you used to log in. They must be exactly the same for Scheduler to recognize your user.

However, you really need an admin user - one who can change anything within the system. For now, this is a manual process. Proceed as follows:

```
$ cd ~/Work/Coding/scheduler
$ export RAILS_ENV=production        # Or development
$ rails c
2.6.6 :001 > u = User.first
...                                  # Output suppressed
2.6.6 :002 > u.permissions[:admin] = true
```

(continues on next page)

(continued from previous page)

```
...
2.6.6 :003 > u.save
...
2.6.6 :004 > exit
$
```

What you are doing here is to invoke the rails console, which gives you direct access to the database and the means to type in Ruby code to be executed immediately.

What you are doing here is to:

- Select the first (only) user in the system
- Give him or her admin privileges
- Save the record back to the database
- And exit

You should then find that your user has full admin privileges and you can proceed to the Scheduler Configuration Guide.

CHAPTER

NINE

MAINTENANCE (CRON) JOBS

There are various tasks which you can run in the background at regular intervals to update or maintain your Scheduler system.

These make use of the standard cron facility of UNIX-like systems. A sample crontab is provided in the support directory of your scheduler installation.

9.1 Configuration prerequisites

When cron jobs run on your system they run as your selected user but they aren't automatically given quite the same run-time environment as an interactive user session. For this reason, a couple of extra steps are needed to enable them to run.

At the end of page on setting up Scheduler you were told to create an rvm alias. Make sure you did actually do that because without it the cron jobs won't know what Ruby environment to use.

Likewise, your cron jobs need to know where your Scheduler installation is, and whether you're running it in development or production mode. Make a directory called "etc" in your Scheduler user's home directory and copy the file support/whichsystem into it. Then edit that file to match your system. There are comments within it to tell you what to change.

```
$ mkdir ~/etc
$ cp support/whichsystem ~/etc
$ vi ~/etc/whichsystem
```

9.2 Create crontab

A sample crontab is provided - support/crontab - and it can be installed very simply by typing:

```
$ crontab support/crontab
```

You will probably want to review it first to see that the times given suit you, and you want all the jobs which it creates. Any which you don't want can be commented out. If you've not used the recommended user name or directory structure then you can do a global edit to this file to change it to match your system.

If you're not familiar with cron then you can type "man cron" to see its documentation, but the lines are fairly easy to understand. Here's a sample:

```
# m h   dom mon dow   command
0 3 * * 0-5 /home/scheduler/Work/Coding/scheduler/utils/checkclashes
```

The first line is a comment, with a mnemonic for remembering the order of items on the next line:

- Minute - 0-59
- Hour - 0-23
- Day of Month - 1-31
- Month - 1-12
- Day of week - 0-6 (0 is Sunday)
- Command - whatever you want to run

The sample above says to run the command at 03:00 on days 0-5 (Sun to Fri) regardless of the date.

9.3 List of jobs

The following sections are best read with reference to the sample crontab provided.

9.3.1 MIS import

Scheduler can import automatically each night from your MIS. Currently supported are SchoolBase, iSAMS and WCBS/Pass. This task is sufficiently large to warrant its own separate documentation.

9.3.2 SOCS import

Likewise, if you use SOCS for your sports fixtures, Scheduler can import them automatically. Again, documented separately.

9.3.3 Clash checking

There are two cron jobs - checkclashes and checkclashesahead which run each night to identify double bookings of pupils. Typically these occur where, for instance, a group of pupils are going on a trip or are involved in an extra-curricular activity. Provided the trip information has been entered into Scheduler, the cron jobs will then identify exactly which pupils will be missing which lessons, and annotate the lessons accordingly. A red blob appears on the lesson in the general calendar view and the teacher can click on the lesson to discover exactly who will be missing.

This can be very useful in lesson planning. 1 or 2 pupils missing out of 28 probably needs no special adjustment, but if 22 of them are missing there is little point in running a conventional lesson.

The jobs are quite long-winded, since they check every lesson for every pupil in the school and thus are best run at night. The checkclashes job does the current and following weeks, whilst the checkclashesahead one does the following 4 weeks.

9.3.4 Clash notification

Staff can opt to receive e-mail notifications of projected absences from their lessons, either immediately they are identified, at the start of the week, or daily (or any combination).

Immediate notification is handled by the clash checking jobs themselves. (Although the jobs run every night, staff are notified only when the absence is first calculated.)

Daily and weekly notifications are triggered by the dailyclashsummary and weeklyclashsummary cron jobs.

9.3.5 Daily reports

The daily_report job generates notification e-mails on the following topics:

- Resource requests awaiting approval
- Loadings for pooled resources
- Forms waiting to be filled in
- Resource requests to be re-confirmed
- Events in need of staff (e.g. mini-bus drivers)

9.3.6 Invigilation notification

In a similar way to clash notifications, the system can send out reminders to individuals who have been scheduled to do exam invigilation. The person allocating the invigilation slots can choose to send out notifications at the time of allocation, but then the system will also send out daily and weekly reminders if desired. Again, each individual member of staff can turn these on or off according to preference. They default to on.

9.3.7 Daily maintenance

This job runs each night to do some housekeeping tasks within the system. Specifically:

- Adjust the currency of groups
- Check the consistency of user files
- Purge old e-mails from the journal

The first item above warrants a little extra explanation. Each group within the Scheduler system has a start date and potentially an end date. Between those dates it is “current”, but once the end date has passed it ceases to be current. Typically this is used for things like teaching groups or activity groups which come to an end at the end of the academic year. The cron job simply checks for any groups which have passed their end dates and switches them off if they have.

CHAPTER

TEN

AVAILABLE DOCUMENTS

- Scheduler Admin Guide
- Scheduler Advanced User Guide
- Scheduler API Guide
- Scheduler Installation Guide
- Scheduler User Guide

CHAPTER

ELEVEN

INDICES AND TABLES

- search