FULL-STACK ASP.NET CORE

BUILD FULL-STACK WEB APPLICATIONS WITH ASP.NET CORE 3.1

- WEB API
- BLAZOR
- HANGFIRE

MIRCEA TEODOR OPREA

Contents

1. Introduction 1
- 1.1. What the Book Does and Does Not Cover 1
- 1.2. Prerequisites 2
- 1.3. Challenges 2
- 1.4. Community and Discussions 3

2. Planning and Setup 4
- 2.1. Project Description 4
- 2.2. Requirement Design 4
- 2.3. System Architecture 5
  - 2.3.1. MVC Architecture 5
  - 2.3.2. Three-tier Architecture 6
  - 2.3.3. Making the Decision 6
- 2.4. Setting up the Projects 6
- 2.5. Understanding ASP.NET Core 12

3. User Story 01: Saving Basic Data 13
- 3.1. Model and Database 13
  - 3.1.1. Create the Migration 14
  - 3.1.2. Applying the migration 16
- 3.2. The Controller 16
- 3.3 The Frontend 21
- Challenge 29

4. Sending Emails 30
- 4.1. Setting up the Email Sender 30
- 4.2. Setting up the Background Tasks 32
- Challenge 35

5. Sending Confirmations 36
5.1. Sending Verification Emails 36
5.2. The Authentication Service 40
5.3. Updating the RemindersController 45
5.4. The Authentication Controller 48
5.5. The Confirmation Page 49
Challenge 52

6. Seeing All Future Reminders 53
6.1. The New Login Email 53
6.2. Generating the Token 55
6.3. Authentication with JWT 60
6.4. Getting the reminders 69
Challenge 77

Conclusion 78

04

Sending Emails

As the first user story is done and the basic foundation for the application is functioning, it's time to move to the next user story:

- As a user, I want to receive my reminder at the date I have set.

This only means adding the back-end functionality to actually send the reminders at the date and time the user chooses—there is no front-end component for this task since it was already done in the previous story.

There are two parts when it comes to this task: sending emails and scheduling them. For sending emails, you can use an SMTP server that provides a free tier, such as Gmail- which you can use to send up to 2000 emails/day. For scheduling them at the right time, we are going to use Hangfire, a library that enables scheduling background tasks by using persistent storage—by default, and in our case, an SQL database.

4.1. Setting up the Email Sender

Let's start by setting up the class that will be used to send the emails. In the **EmailReminder.WebApi** project, create a new folder called **Services** and a new interface inside it called **IMailSender**. For now, this will only contain one method:

```
        public interface IMailSender
{
    void SendReminderAsync(Reminder reminder);
}
```

Next, let's create an implementation of this interface. Create a new class and name it according to what SMTP service you are using; in this case, **GmailMailSender**. Make it implement **IMailSender** and then add the following content to the method:

```
public class GmailMailSender : IMailSender
 {
    public async void SendReminderAsync(Reminder reminder)
    {
       using (var client = new SmtpClient("smtp.gmail.com", 587))
       {
          client.UseDefaultCredentials = false;
         client.Credentials = new NetworkCredential("YOUR_EMAIL", "YOUR_PASSWORD");
          client.DeliveryMethod = SmtpDeliveryMethod.Network;
          client.EnableSsl = true;
          MailMessage mailMessage = new MailMessage();
          mailMessage.From = new MailAddress("YOUR_EMAIL");
          mailMessage.To.Add(reminder.EmailAddress);
          mailMessage.Subject = $"Your reminder for {reminder.DateTime.ToShortDateString()}";
          mailMessage.Body = reminder.Message;
          client.Send(mailMessage);
       }
    }
 }
```

The method is using an **SmtpClient** object set to the server and port provided by GMail; it's important to keep the properties set in the orders you see them, as sometimes enabling SSL before setting the credentials will result in failure. Afterwards, the method constructs a **MailMessage** object and copies the necessary data from the provided **Reminder** object.

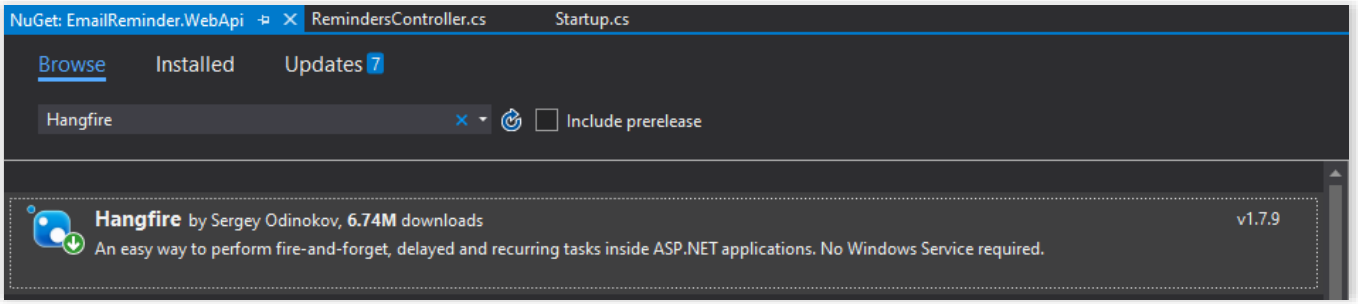
If you are using a service other than Gmail (such as Mailjet or Mandrill), they should provide you with their own network credentials which you can use.

Finally, setup the service injection in the **Startup** class by adding the following line in the **ConfigureServices** method:

```
services.AddScoped<IMailSender, GmailMailSender>();
```

4.2. Setting up the Background Tasks

As mentioned before, in order to schedule the background tasks that will take care of the emails, Hangfire will be used. To add the library, right-click on the **EmailReminder.WebApi** project and click "Manage Nuget Packages". Go to the "Browse" tab, search for **Hangfire** and install the first package:



Once the installation is done, go to the **Startup** class and add the following lines to the **ConfigureServices** method:

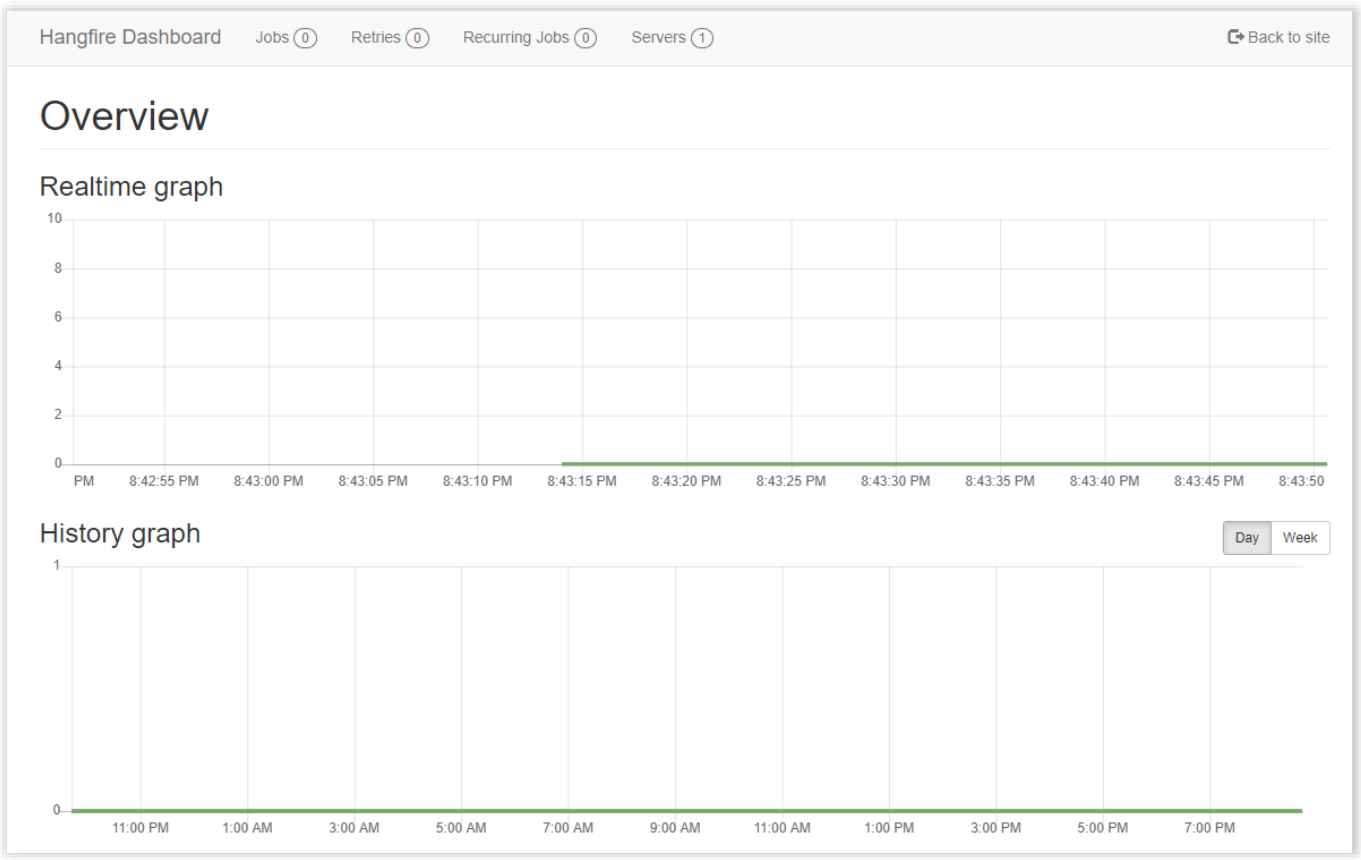
```
services.AddHangfire(x => 
x.UseSqlServerStorage(Configuration.GetConnectionString("DefaultConnection")));

services.AddHangfireServer();
```

and the following to the **Configure** method, as the **first line**:

```
app.UseHangfireDashboard();
```

Having it as the first line is important, as the middleware added in this method are called in the order that you add them; if you add it after the **UseEndpoints** method is called, ASP.NET will try to find a controller named **Hangfire**, which does not exist.

To check that everything was set up correctly, run the project and append "/hangfire" to the Web API URI, e.g.: http://localhost:44335/hangfire. You should see the following page:



Once tasks are actually scheduled, this page will show information about when they run, whether they succeeded or failed, and more debugging information.

To schedule the task when a **Reminder** is created, we need to make changes in the **RemindersController**. First of all, let's inject an **IBackgroundJobClient** into the controller:

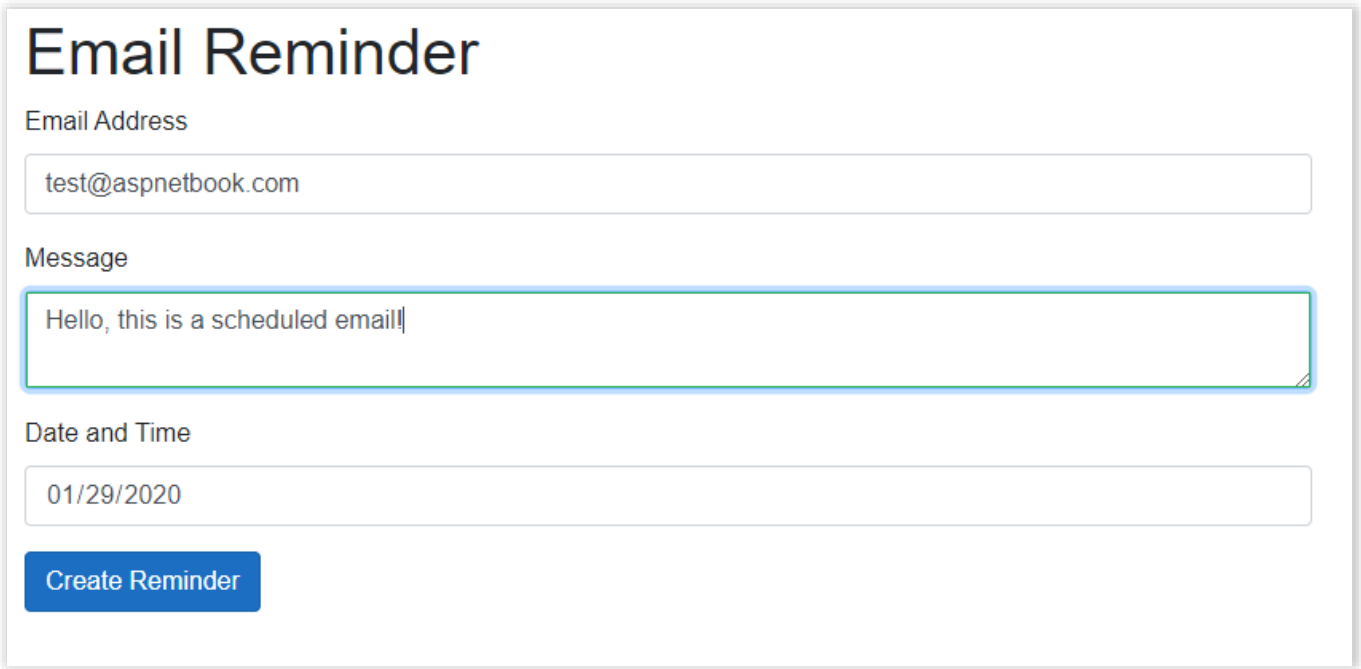
```
private readonly ApplicationDbContext _context;
private readonly IBackgroundJobClient _backgroundJobClient;
public RemindersController(
    ApplicationDbContext context,
    IBackgroundJobClient backgroundJobClient)
{
    _context = context;
    _backgroundJobClient = backgroundJobClient;
}
```

This interface is part of Hangfire, and it provides ways to enqueue and schedule background tasks. To actually do it, we just need to add an extra line in the **CreateReminder** method:

```
[HttpPost]
public async Task<IActionResult> CreateReminder([FromBody]Reminder reminder)
{
  try
  {
    _context.Reminders.Add(reminder);
    await _context.SaveChangesAsync();
  }
  catch
  {
    return BadRequest("Could not save reminder.");
  }

  _backgroundJobClient.Schedule<IMailSender>(x => x.SendReminderAsync(reminder), new DateTimeOffset(reminder.DateTime));

  return Ok(reminder);
}
```

The client will get an **IMailSender** instance by injection, then call the **SendReminderAsync** method at the required time and date. This task will be stored in a separate table in the database, so even if you restart your application, or move it to another server, the tasks will persist.

To test it, run the application and send yourself an email and set the current date:



After a few seconds, you should see it in your inbox!

Challenge

Having credentials randomly spread throughout the code is not ideal. In the **Startup** class, you can see a property called **Configuration**. You can see it used on the first few lines of the **ConfigureServices** method:

```
services.AddDbContext<ApplicationDbContext>(options =>
 options.UseSqlServer(
    Configuration.GetConnectionString("DefaultConnection")));
```

This reads data from the **appsettings.json** file, where you can see a **ConnectionStrings** object, with a **DefaultConnection** string inside.

The same principle can be used for the email credentials. Create a new class called **EmailCredentials**—you can place it either in the **EmailReminder.WebApi** or in the **EmailReminder.Shared** project—with a user and a password as properties. Add your credentials to the **appsettings.json** file, read them using the **Configuration** property, and then add it as a service in the **ConfigureServices** method. In the **GmailMailSender** class, add a field of the **EmailCredentials** type, inject it through the constructor and use it in the **SendReminderAsync** method.