

Azure WebApps + Storage + Functions Hands On

Folgender UseCase wird implementiert:

- Angular Frontend mit einfachem ImageUpload
- ImageUpload in BlobStorage über ASP.NET Core Middleware
- Azure Function
 - Trigger auf Änderungen im Blob Storage
 - Resizen hochgeladener Bilder auf Thumbnail Größe
 - Ablegen im Thumbnail Container im Blob Storage

Requirements:

- Node.js
<https://nodejs.org/en/>
- .NET Core
<https://www.microsoft.com/net/download>
- Angular CLI
npm install -g @angular/cli
- Azure Resources (Web App + Storage + Function App)
- Azure Functions and WebJob Tools (Visual Studio -> Tools -> Extension and Updates)

1 Project scaffolding

- `dotnet new angular --name file-upload`

Im Ordner "file-upload"

- `dotnet add package WindowsAzure.Storage`

Im Ordner "file-upload\ClientApp"

- `npm install`
- `ng g component fileUpload`
- `npm install primeng --save`
- `npm install primeicons --save`

Projekt mit Visual Studio öffnen und ausführen.

2 Middleware Controller

appsettings.json:

```
"ConnectionStrings": {  
  "StorageAccount": "XXX"  
},
```

Im Ordner „Contollers“ neuen „FileUploadController.cs“ erstellen:

```
[Route("api/[controller]")]  
1 reference  
public class FileUploadController : Controller  
{  
    //Appsettings Configuration  
    private readonly IConfiguration _configuration;  
    0 references | 0 exceptions  
    public FileUploadController(IConfiguration config)  
    {  
        _configuration = config;  
    }  
  
    [HttpPost]  
    0 references | 0 requests | 0 exceptions  
    public async Task<IActionResult> UploadFileAsync([FromForm]IFormFile file)  
    {  
        //Parse ConnectionString  
        if (CloudStorageAccount.TryParse(_configuration.GetConnectionString("StorageAccount"), out CloudStorageAccount storageAccount))  
        {  
            //Create client and create BlobContainer  
            var client = storageAccount.CreateCloudBlobClient();  
            var container = client.GetContainerReference("originalfile");  
            await container.CreateIfNotExistsAsync();  
  
            //Creates a Blob and uploads file into Blob  
            var blob = container.GetBlockBlobReference(file.FileName);  
            await blob.UploadFromStreamAsync(file.OpenReadStream());  
  
            return Ok(blob.Uri);  
        }  
  
        return StatusCode(StatusCodes.Status500InternalServerError);  
    }  
}
```

3 Frontend

Bereitstellen des IconSets und Styles in „angular-cli.json“

```
"styles": [
  "../node_modules/primeicons/primeicons.css",
  "../node_modules/primeng/resources/themes/nova-light/theme.css",
  "../node_modules/primeng/resources/primeng.min.css",
  "styles.css",
  "../node_modules/bootstrap/dist/css/bootstrap.min.css"
],
```

Bereitstellen benötigter Module und Komponenten in „app.module.ts“

```
import { FileUploadModule } from 'primeng/fileupload';

@NgModule({
  declarations: [
    AppComponent,
    NavMenuComponent,
    HomeComponent,
    CounterComponent,
    FetchDataComponent,
    FileUploadComponent
  ],
  imports: [
    BrowserModule.withServerTransition({ appId: 'ng-cli-universal' }),
    HttpClientModule,
    FormsModule,
    FileUploadModule,
    RouterModule.forRoot([
      { path: 'file-upload', component: FileUploadComponent },
      { path: '', component: HomeComponent, pathMatch: 'full' },
      { path: 'counter', component: CounterComponent },
      { path: 'fetch-data', component: FetchDataComponent },
    ])
  ],
})
```

Implementierung eines Menüverweises zum FileUpload in „nav-menu.component.html“

```
<div class='navbar-collapse collapse' [ngClass]='{ "in": isExpanded }'>
  <ul class='nav navbar-nav'>
    <li [routerLinkActive]='["link-active"]'>
      <a [routerLink]='["/file-upload"]' (click)='collapse()'>
        <span class='glyphicon glyphicon-education'></span> Upload
      </a>
    </li>
    <li [routerLinkActive]='["link-active"]' [routerLinkActiveOptions]='{ exact: true }'>
      <a [routerLink]='["/"]' (click)='collapse()'>
        <span class='glyphicon glyphicon-home'></span> Home
      </a>
    </li>
    <li [routerLinkActive]='["link-active"]'>
      <a [routerLink]='["/counter"]' (click)='collapse()'>
        <span class='glyphicon glyphicon-education'></span> Counter
      </a>
    </li>
  </ul>
</div>
```

Implementierung des FileUpload Controls in „file-upload.component.html“

```
<p>  
  file-upload works!  
</p>  
<p-fileUpload name="file" url="/api/FileUpload" maxFileSize="1000000" auto="true" chooseLabel="Browse">  
</p-fileUpload>
```

4 Deployment der WebApp

Projektkontextmenü -> Publish


Publish

Publish your app to Azure or another host. [Learn more](#)

 wdara - Web Deploy ▼

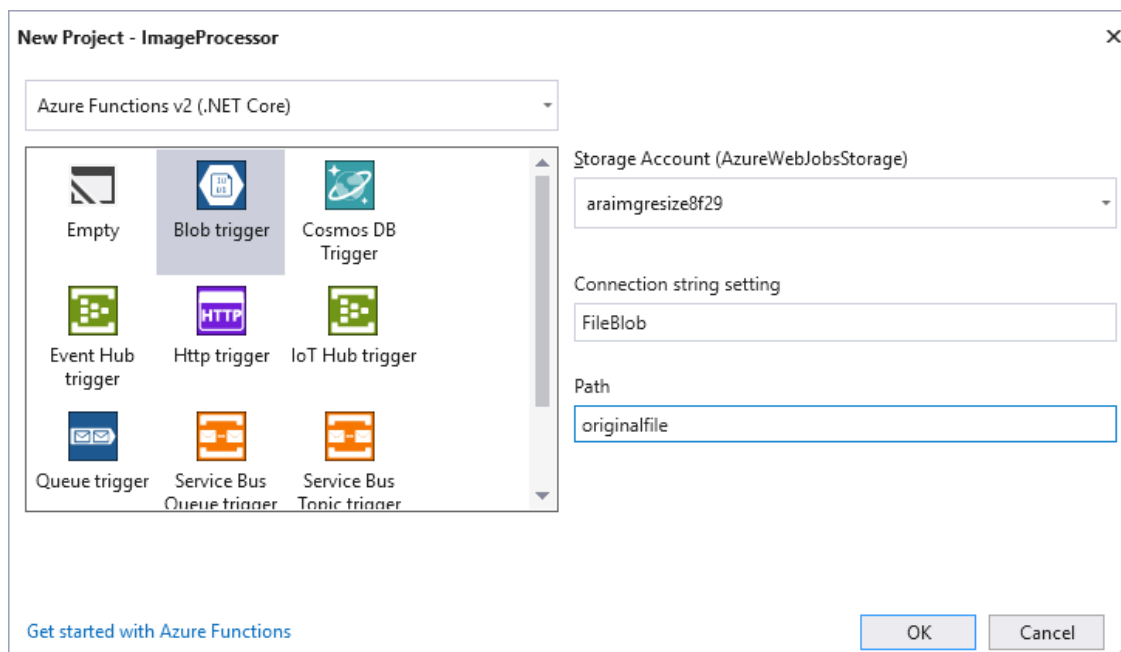
Publish

[New Profile...](#)[Actions ▼](#)

Site URL	http://wdara.azurewebsite... 	Edit App Service Settings...
Resource Group	wdara	Manage In Cloud Explorer
Configuration	Release	Preview...
Troubleshooting Info	See Guide	Configure...

5 Azure Functions

Neues Azure Functions Projekt erstellen



In "local.settings.json" ConnectionString hinzufügen

```
"IsEncrypted": false,  
"Values": {  
  "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;AccountName=araimgresize8f29;AccountKey=araimgupload;AccountKey=Zhd9rzKSJ",  
  "FileBlob": "DefaultEndpointsProtocol=https;AccountName=araimgupload;AccountKey=Zhd9rzKSJ",  
  "FUNCTIONS_WORKER_RUNTIME": "dotnet"  
}
```

Lokal ausführen und mit Bildupload testen.

ÜBUNG

- Bild prozessieren und in einen neuen BlobContainer ablegen
- Azure Function deployen

Tipps:

Bildbearbeitung: Nuget Package „ImageSharp“ <https://github.com/SixLabors/ImageSharp> (siehe API Beispiel)

Analog zum Controller in der Middleware das prozessierte Bild in einen neuen Container im Blob ablegen.

Die Appsetting „FileBlob“ muss nach dem deployen in die Appsettings der „online“ Function App eingetragen werden.