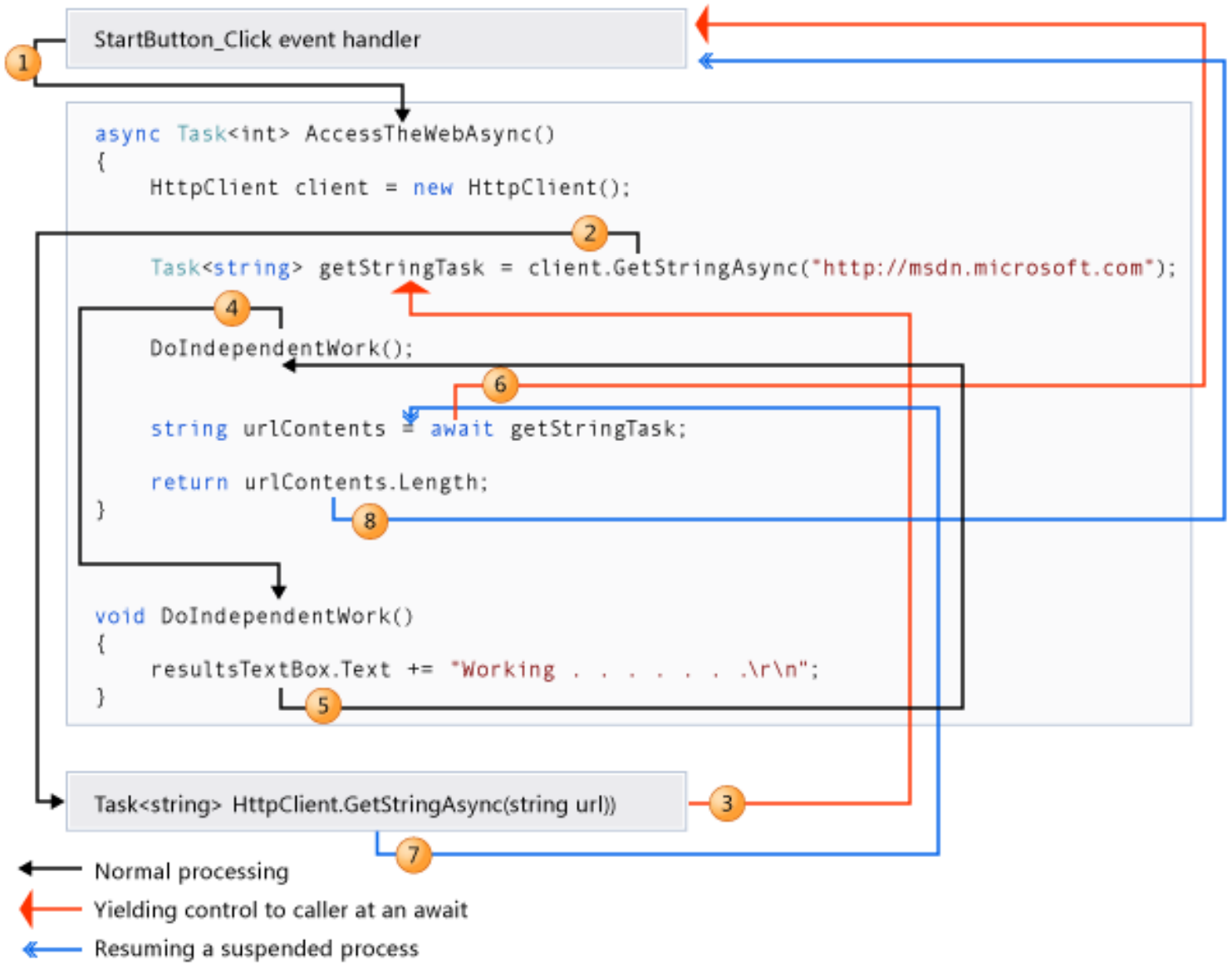
Kentico

Asynchronous programming

Tomas Hruby

2015

Kentico


StartButton_Click event handler
1
async Task<int> AccessTheWebAsync()
{
HttpClient client = new HttpClient();
2
Task<string> getStringTask = client.GetStringAsync("http://msdn.microsoft.com");
4
DoIndependentWork();
6
string urlContents = await getStringTask;
return urlContents.Length;
}
8
void DoIndependentWork()
{
resultsTextBox.Text += "Working . . . . . .\r\n";
}
5
Task<string> HttpClient.GetStringAsync(string url))
3
7
Normal processing
Yielding control to caller at an await
Resuming a suspended process

Async-Await

- Put async-await only across **the whole** application
- **Don't mix** manipulation with threads with async-await
  - Blocking context threads causes deadlocks
  - To use async code from normal method use **Task.Run**
- Method can continue in a **different thread** after awaiting
  - Don't depend on the thread context
  - HTTP context copies context information to new thread from the thread pool

https://msdn.microsoft.com/en-us/library/hh191443.aspx
http://blog.stephencleary.com/2012/07/dont-block-on-async-code.html

Kentico

REST APIs

Tomas Hruby
2015

The brightest future of .NET

- REST API is in fact the only thing that for example a React application needs as a backend.
- REST API is platform agnostic
- REST API is easy to read and edit/debug
- .NET accelerates development of REST APIs
  - Programming—Web API, Newtonsoft.Json, OWIN, …
  - Deployment—Microsoft Azure
- Scalability (horizontal)
  - Stateless services + REST API + Azure

Basics revisited

- Stateless HTTP communication (Request-Response)
- URL identifies an object
  - ~/api/users/ngk67
  - ~/api/users/ngk67/roles
  - Use plural for endpoints (/api/users, /api/projects)
- Method identifies action (GET, POST, PUT, PATCH, DELETE)
  - GET ~/api/users (get all users)
  - DELETE ~/api/users/ngld456 (delete user with given ID)
- Request body represents the object identified by URI
  - POST ~/api/users (object in request body is inserted as user)
- Server answers with HTTP Status Code and response body containing requested object(s)

HTTP Status Codes

Method	URL pattern	Statuses	(Notes)
GET	/api/users	200	
GET	/api/users/{id}	200, 404	
POST	/api/users	201, 400	201 – Created
PUT	/api/users/{id}	200, 404, 400	400 – Bad request
DELETE	/api/users/{id}	204, 404	204 – No content

401 – Unauthorized
500 – Internal Server Error
429 – Too many requests (response should contain time of renewal)

http://www.restapitutorial.com/httpstatuscodes.html

Filtering & Paging

- Filtering using query parameters
  - GET ~/api/students?semester=spring
- Filtering using query objects
  - GET ~/api/students?filter={“subjects”=[“PE”,“maths”],“age”=18}}
  - REST calls remain the same even if new filter is added (forward compatible)
  - Easy to write by hand in browser URL bar
- Paging request
  - GET ~/api/students?pagesize=10&page=3
- Paging responses
  - Pagination info in header (X-Total-Count, X-Page-Size, X-Current-Page)
  - Pagination envelope
    - { “total”: 40, “pageSize”: 10, “page”: 1, “data”: [...] }
  - Cursor based pagination (link to next page)

Versioning

- Using URL prefixes
  - ~/api/v1/users
- Using HTTP headers
  - X-API-Version: 7.1.3

Special actions

- Don’t use special query parameters (as flags). Use special endpoints
  - ~/api/students/{id}/disable
  - ~/api/users/resetpassword

Few other design tips

- Use pretty printed JSON in responses
- Put examples into documentation
- Use projections (do not return all attributes in case of large objects)
- Implement only necessary methods

http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

ASP.NET WebAPI 2

- Uses the same engine as MVC (controllers, filters, …)
  - Controllers inherit from **System.Web.Http.ApiController**
- Has different types of action results
- Has similar request lifecycle
  - http://www.asp.net/media/4071077/aspnet-web-api-poster.pdf

http://www.asp.net/web-api