Parallele Programmierung

Dipl.-Ing. Msc. Paul Panhofer Bsc.

15. Februar 2022

API - Programmschnittstelle

Eine **API** - Application Programming Interface - ist ein Satz von Befehlen, Funktionen, Protokollen und Objekten. APIs ermöglichen Anwendungen auf einfache Weise mit-einander zu kommunizieren.

API - Programmschnittstelle

Historische Entwicklung

Zu den frühesten und bekanntesten APIs gehört die API von ebay.

- Ebay stellte seinen Nutzern einen einfachen Zugriff auf seine Seiten zur Verfügung, um Massenuploads von Inseraten zu erleichtern. (2000).
- Zwei Jahre später erschien Amazon Web Services auf der Bildfläche. Seitdem ist die Anzahl der APIs exponentiell gestiegen.

API - Programmschnittstelle

Einsatzgebiete

- Frontend - Backendkommunikation
- Komponentenkommunikation
- DaaS

API - Programmschnittstelle

Einsatz: Frontend - Backendkommunikation

Frontend: `Angular.js, react.js, Vue.js`
Backend: `Spring, express.js, .net Core, laravel...`

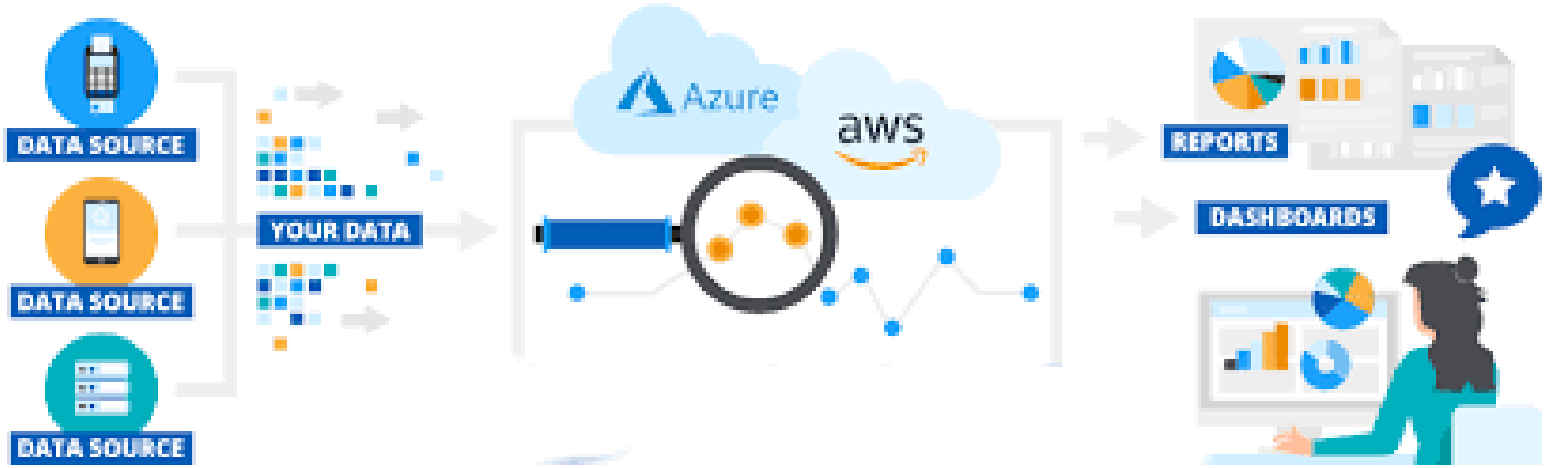
API - Programmschnittstelle

Einsatz: Komponentenkommunikation

API - Programmschnittstelle

Einsatz: DaaS

DaaS - Data as a Service - ist ein Ansatz Daten als Service in der Cloud zu Verfügung zu stellen.



API - Programmschnittstelle

Kommunikationsprotokolle

Für die Implementierung einer API können unterschiedliche Kommunikationsprotokolle verwendet werden.

1. Prinzipien verteilter Programmierung
   - API - Programmschnittstelle
   - Kommunikationsprotokolle
2. Serviceschicht - Programmierung
   - Controller

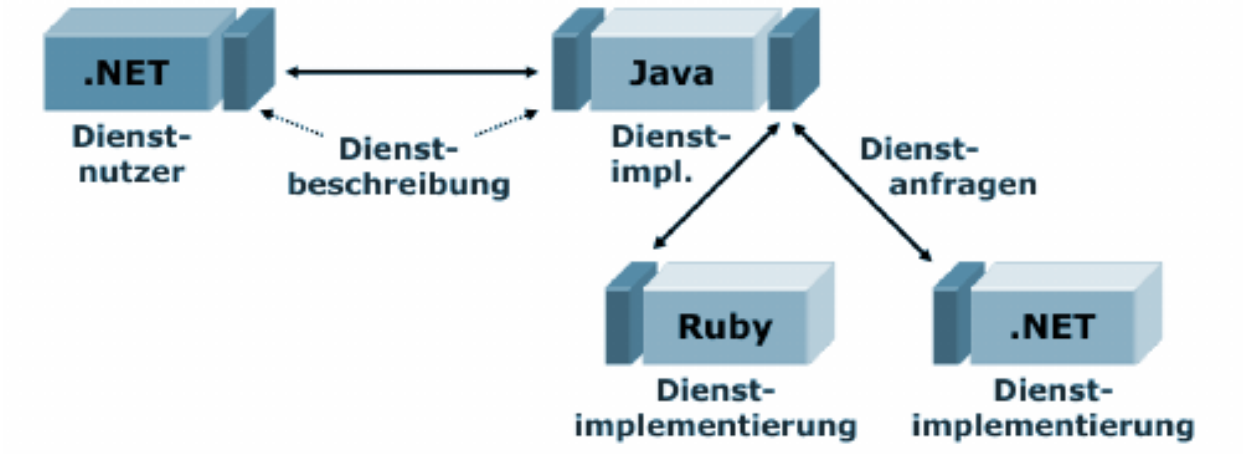
Kommunikationsprotokolle

Kommunikationsprotokolle ermöglichen das **Austauschen** von Nachrichten zwischen 2 Prozessen.

Diese Prozesse können dabei im Arbeitsspeicher unterschiedlicher Netzwerkknoten ausgeführt werden.

Kommunikationsprotokolle

Offenheit



Kommunikationsprotokolle

API Programmierung

Für die **Programmierung** von APIs werden unterschiedliche Protokolle verwendet:

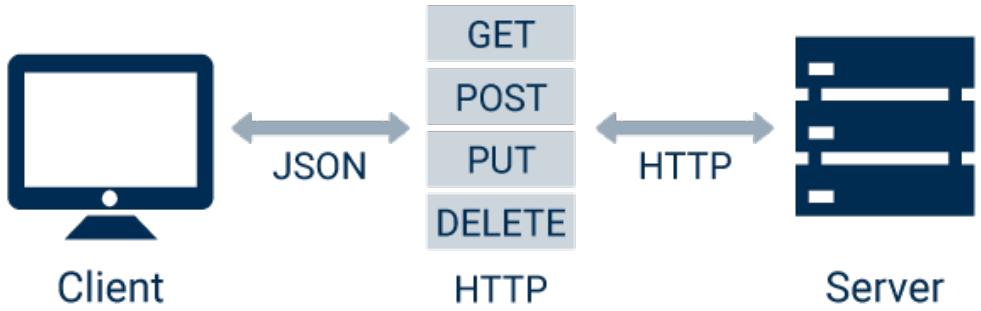
- Rest
- gRPC
- GraphQL

Kommunikationsprotokolle

Rest API

Rest versteht das Internet als eine Sammlung von Ressourcen. Die REST API ermöglicht die Verwaltung von Ressourcen.

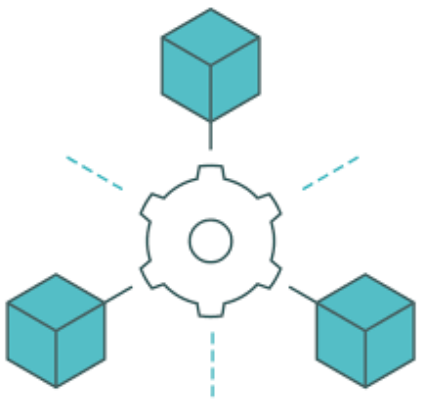
Rest verwendet zur Kommunikation das **HTTP Protokoll**.



Kommunikationsprotokolle

gRPC - Google Remote Procedure Call

gRPC ist ein Kommunikationsprotokoll zum Aufruf von Methoden/Funktionen in verteilten Systemen.



Kommunikationsprotokolle

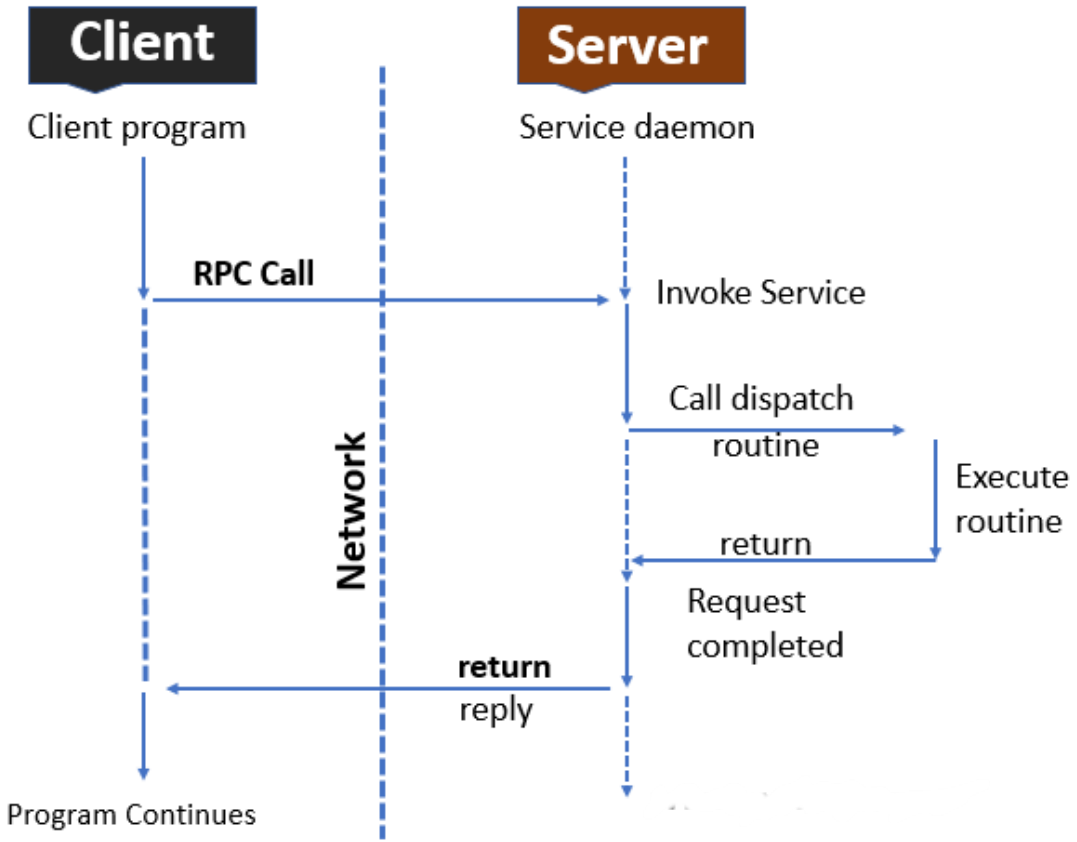
gRPC - Google Remote Procedure Call

gRPC ermöglicht den Aufruf von Methoden in Objekten im Speicher eines anderen Netzwerkknotens.

Für die Kommunikation verwendet gRPC ein eigenes binäres Protokoll.

Kommunikationsprotokolle

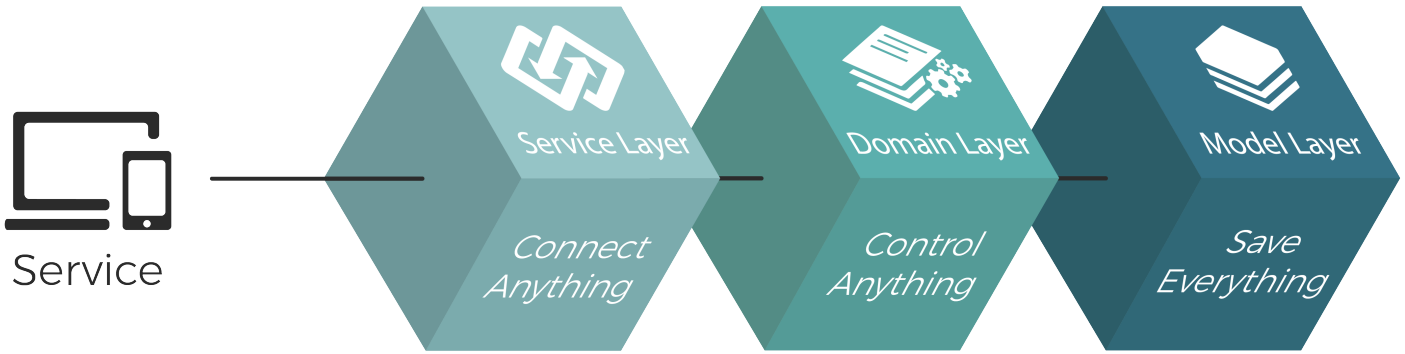
gRPC API



1. Prinzipien verteilter Programmierung
   - API - Programmschnittstelle
   - Kommunikationsprotokolle
2. Serviceschicht - Programmierung
   - Controller

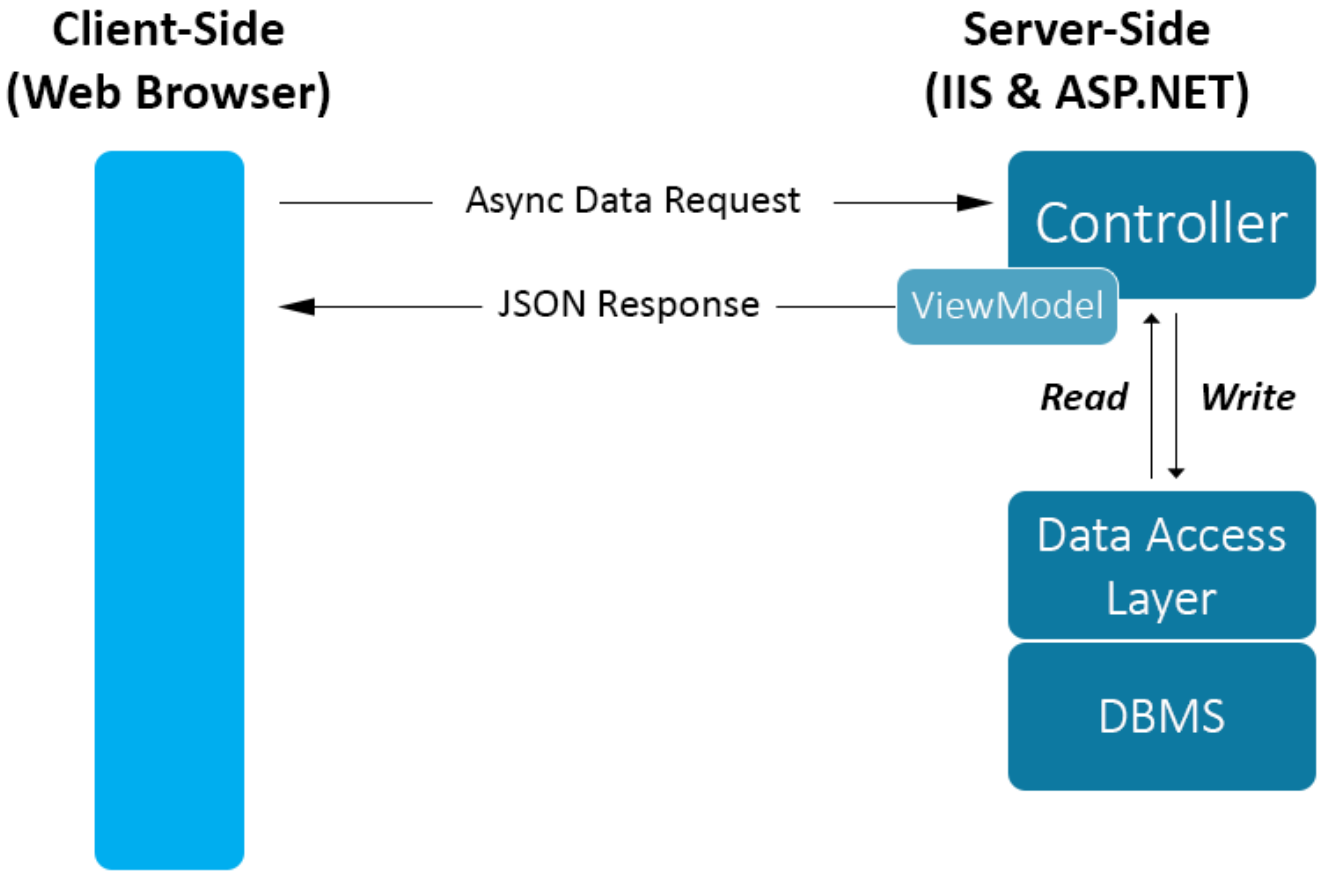
Controller

Schichtenmodell



Controller

Kommunikationsschnittstelle



Controller

HTTP Request


```
POST /api/recipes HTTP/1.1
Host: recipes.contoso.com
User-Agent: demo/client
Accept: application/json
Content-Type: application/json
```

Controller

HTTP Response


```
HTTP/1.1 200 OK
Connection: close
Date: Mon, 18 Jan 2021 18:13:48 GMT
Content-Type: application/json; charset=utf-8
Server: Kestrel
Content-Length: 1368

[
  {
```

Controller

Implementierung

```
// Modelklasse
public class Person {

    private Guid Id { get; set; }

    [Required]
    [StringLength(30)]
    private string FirstName { get; set; }

    [Required]
    [StringLength(30)]
    private string LastName { get; set; }

}
```

Controller

Implementierung - ApiController, Route

```
/* 1.) ApiController: AC ist ein FrameworkHook der AC
 *     Objekte als Externe Schnittstelle zur Verarbeitung
 *     von HTTP Requests bereitstellt.
 */
[ApiController]
/* 2.) Route: Das Route Attribut definiert die URL der
 *     Ressource.
 */
[Route("people")]
public class PersonController : ControllerBase {...}
```

Controller

Implementierung - ControllerBase

```
[ApiController]
[Route("people")]
/* 3.) ControllerBase: ControllerBase stellt eine
 *     Implementierung des HTTP Protokolls dar.
 *     Klassen die von ControllerBase erben koennen
 *     HttpRequests entgegennehmen bzw. HttpResponses
 *     absetzen
 */
public class PersonController : ControllerBase {...}
```

Controller

Implementierung - Call Method

```
[ApiController]
[Route("people")]
public class PersonController : ControllerBase {
    /* 1.) Damit eine bestimmte Methode in einem Contro-
     *      ller aufgerufen werden kann, muss ein HTTP
     *      Request die URL des Controllers und mit der
     *      entsprechenden HTTP Methode abgesetzt worden
     *      sein.
     */
    [HttpGet("ping")]
    public ActionResult<string> Ping(){
        ...
    }

}
```

Controller

Implementierung - HttpGet

```
[ApiController]
[Route("people")]
public class PersonController : ControllerBase {
    /* 2.) Das HttpGet Attribut definiert die HttpMethode
     *      auf die die Methode registriert ist.
     */
    [HttpGet("ping")]
    public ActionResult<string> Ping(){
        ...
    }

}
```

Controller

Implementierung - ActionResult

```
[ApiController]
[Route("people")]
public class PersonController : ControllerBase {
    /* 3.) Ueber den ActionResult wird der StatusCode
     *      des HTTP Responses gesteuert.
     */
    [HttpGet("ping")]
    public ActionResult<string> Ping(){
        ...
    }

}
```

Controller

HTTP Status Codes

Code	Status
100 - 199	Informational
200 - 299	Successful
300 - 399	Redirection
400 - 499	Client Errors
500 - 599	Server Errors

Controller

Action	Method	Success	Failure
Create	POST	201 (Created)	400 (Bad Request)
Read	GET	200 (Ok)	404 (Not Found)
Update	PUT / PATCH	204 (No Content)	404 (Not Found)
Delete	DELETE	204 (No Content)	400 (Bad Request)

Controller

Implementierung - Read

```
[ApiController]
[Route("people")]
public class PersonController : ControllerBase {

    // URL: ../people/{id}
    // HTTP METHOD: GET
    [HttpGet("{id}")]
    public async Task<ActionResult<Person>> ReadAsync(int
        id){
         var p = await personRepository.ReadAsync(id);

         if( p is null) return NotFound();

         return Ok(p);
    }
}
```

Controller

Implementierung - Create

```
[ApiController]
[Route("people")]
public class PersonController : ControllerBase {

    // URL: ../people
    // HTTP METHOD: POST
    [HttpPost]
    public async Task<ActionResult<Person>>
        CreateAsync(Person p){
         p = await personRepository.Create(p);

         return CreatedAtAction(
             nameof(Read), new {Id = p.Id}, p
         );
    }
}
```

Controller

Implementierung - Update

```
...
public class PersonController : ControllerBase {
    // URL: ../people/{id}
    // HTTP METHOD: PUT
    [HttpPut("{id}")]
    public async Task<ActionResult> UpdateAsync(
        int id, Person p
    ){
        var data = await personRepository.ReadAsync(id);

        if(data is null) return NotFound();
        await personRepository.UpdateAsync(p);

        return NoContent();
    }
}
```

Controller

Implementierung - Delete

```
...
public class PersonController : ControllerBase {
    // URL: ../people/{id}
    // HTTP METHOD: PUT
    [HttpDelete("{id}")]
    public async Task<ActionResult> UpdateAsync(
        int id, Person p
    ){
        var data = await personRepository.ReadAsync(id);

        if(data is null) return NotFound();
        await personRepository.UpdateAsync(p);

        return NoContent();
    }
}
```