FHV
Vorarlberg University
of Applied Sciences

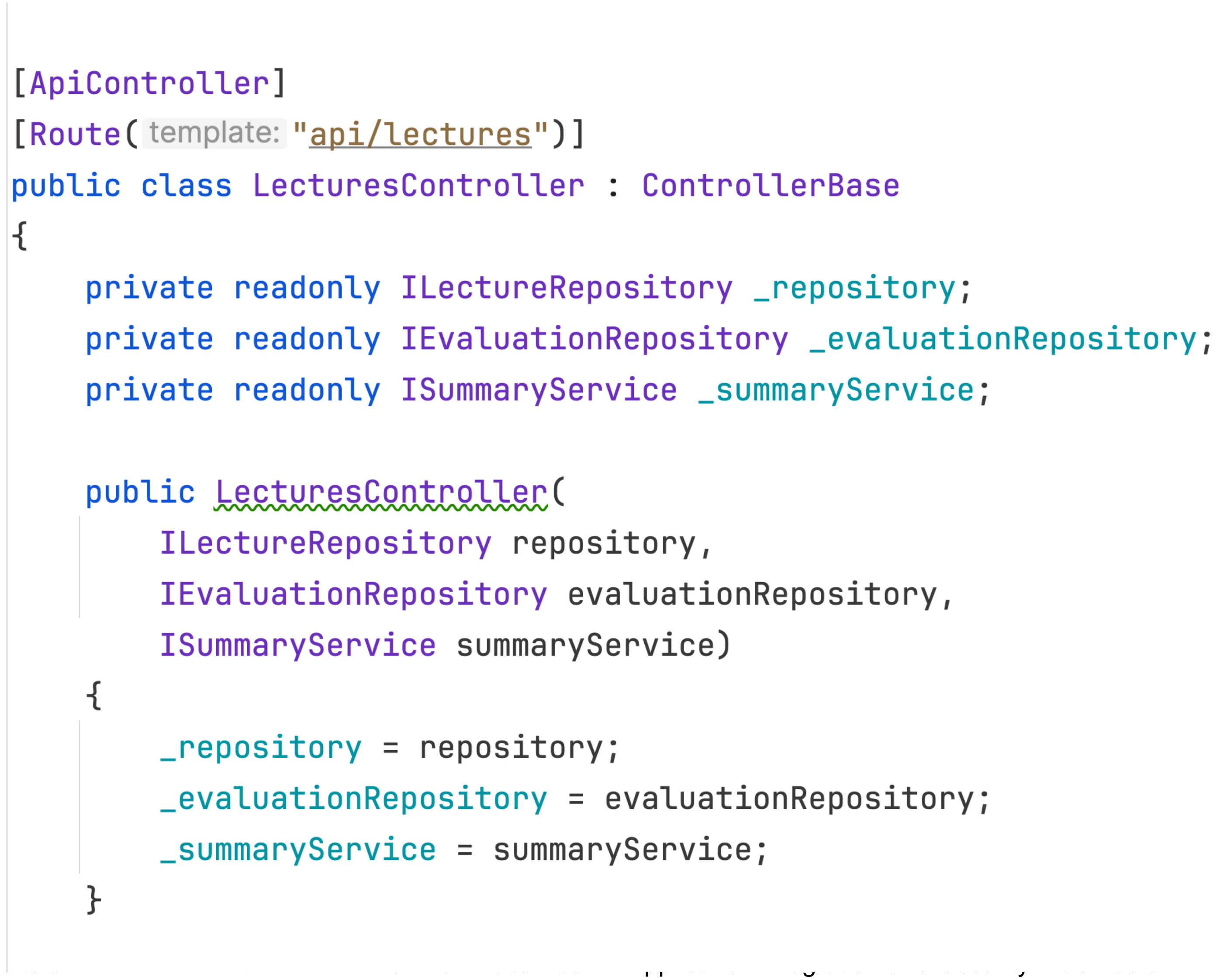
Application Integration and Security

Valmir Bekiri

Philipp Scambor

Code examples
Web-API incl. ORM
(LectureEvaluationAPI)

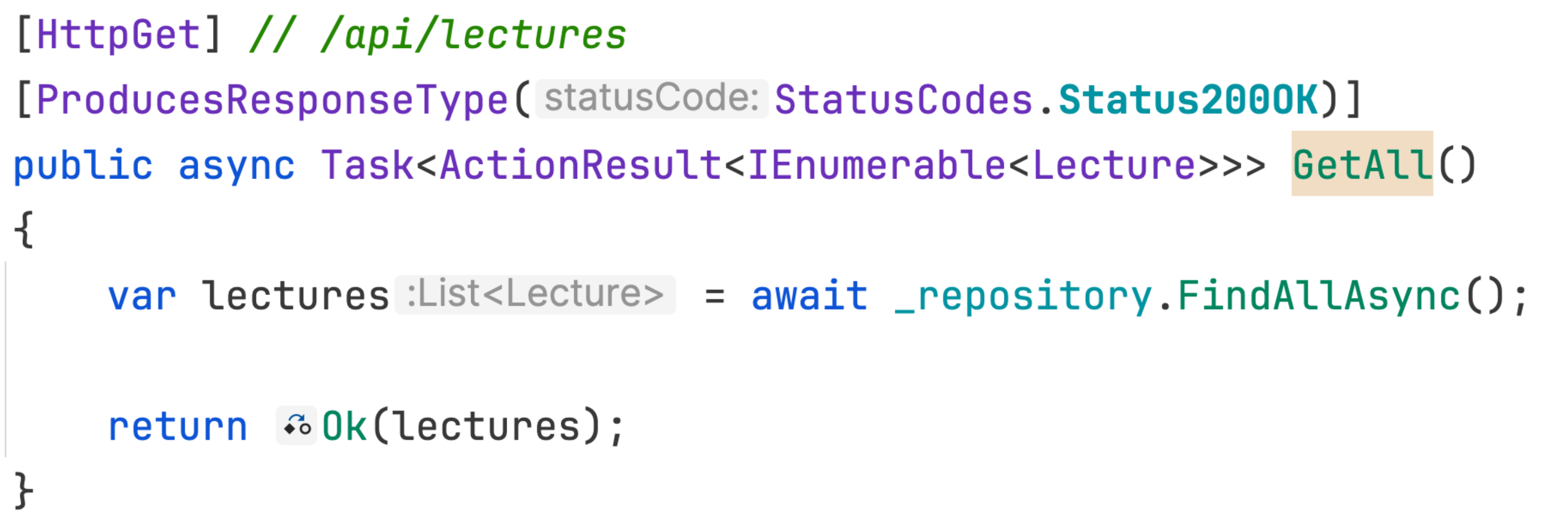
LecturesController



```
[ApiController]
[Route( template: "api/lectures")]
public class LecturesController : ControllerBase
{
    private readonly ILectureRepository _repository;
    private readonly IEvaluationRepository _evaluationRepository;
    private readonly ISummaryService _summaryService;

    public LecturesController(
        ILectureRepository repository,
        IEvaluationRepository evaluationRepository,
        ISummaryService summaryService)
    {
        _repository = repository;
        _evaluationRepository = evaluationRepository;
        _summaryService = summaryService;
    }
```

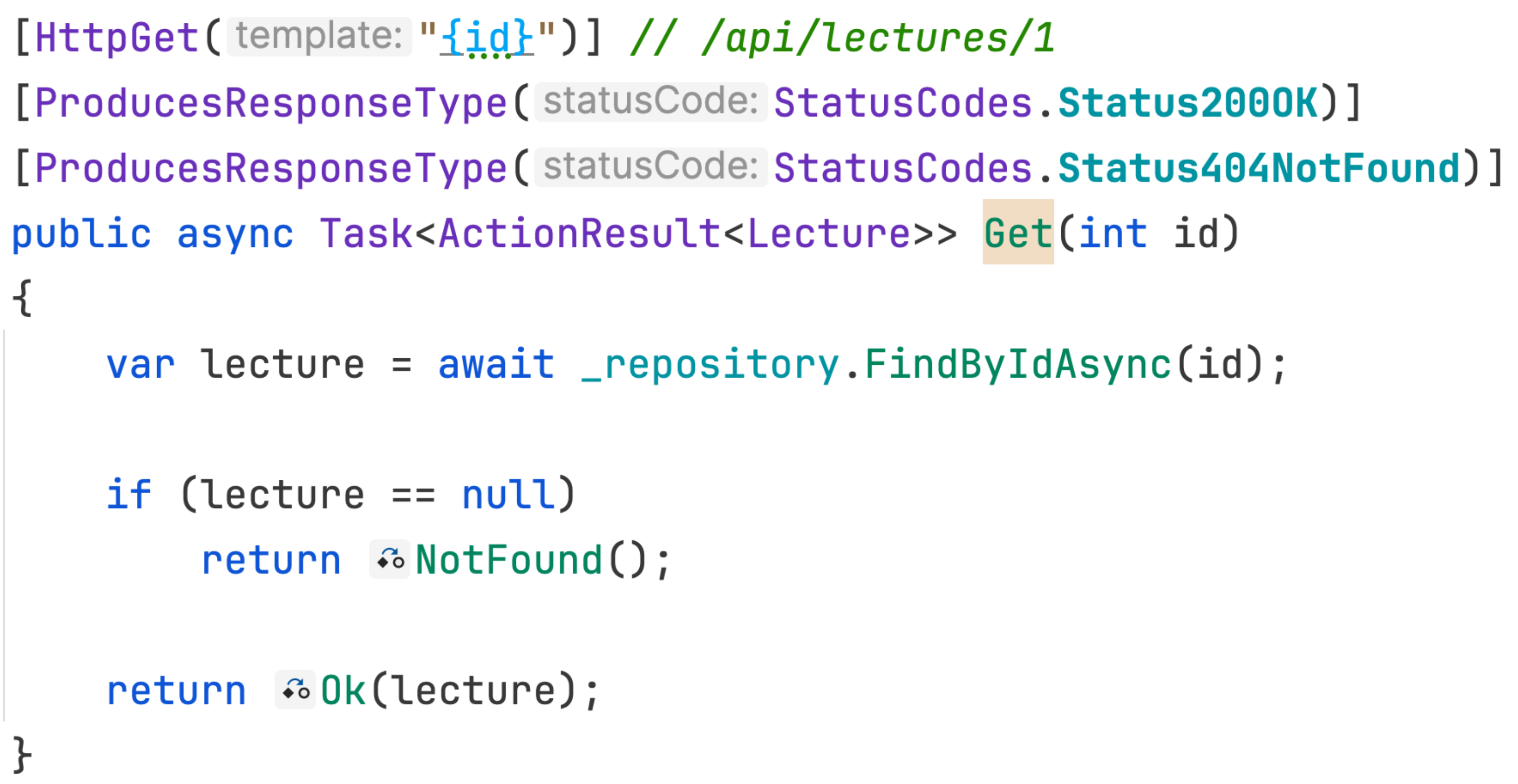
LecturesController



```
[HttpGet] // /api/lectures
[ProducesResponseType( statusCode: StatusCodes.Status200OK)]
public async Task<ActionResult<IEnumerable<Lecture>>> GetAll()
{
    var lectures :List<Lecture> = await _repository.FindAllAsync();

    return Ok(lectures);
}
```

LecturesController

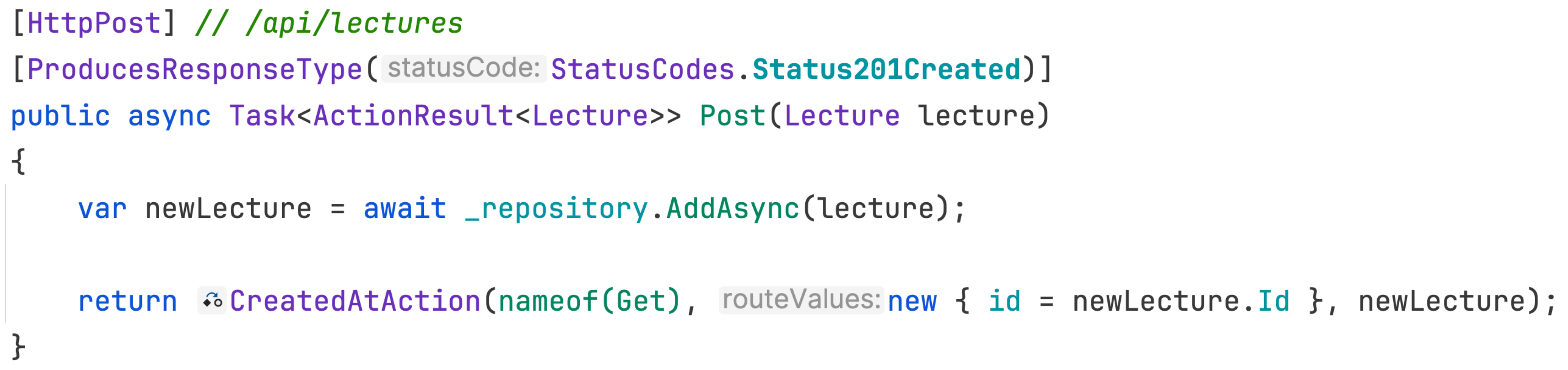


```
[HttpGet( template: "{id}")] // /api/lectures/1
[ProducesResponseType( statusCode: StatusCodes.Status200OK)]
[ProducesResponseType( statusCode: StatusCodes.Status404NotFound)]
public async Task<ActionResult<Lecture>> Get(int id)
{
    var lecture = await _repository.FindByIdAsync(id);

    if (lecture == null)
        return NotFound();

    return Ok(lecture);
}
```

LecturesController



```
[HttpPost] // /api/lectures
[ProducesResponseType( statusCode: StatusCodes.Status201Created)]
public async Task<ActionResult<Lecture>> Post(Lecture lecture)
{
    var newLecture = await _repository.AddAsync(lecture);

    return CreatedAtAction(nameof(Get), routeValues: new { id = newLecture.Id }, newLecture);
}
```

LecturesController

```
[HttpPut( template: "{id}")] // /api/lectures/1
[ProducesResponseType( statusCode: StatusCodes.Status200OK)]
[ProducesResponseType( statusCode: StatusCodes.Status404NotFound)]
public async Task<ActionResult<Lecture>> Put(int id, Lecture lecture)
{
    var existingLecture = await _repository.FindByIdAsync(id);

    if (existingLecture == null)
        return NotFound();

    lecture.Id = existingLecture.Id;

    return Ok(await _repository.UpdateAsync(lecture));
}
```

LecturesController

```
[HttpDelete(template: "{id}")] // /api/lectures/1
[ProducesResponseType(statusCode: StatusCodes.Status200OK)]
[ProducesResponseType(statusCode: StatusCodes.Status404NotFound)]
public async Task<ActionResult<Lecture>> Delete(int id)
{
    var existingLecture = await _repository.FindByIdAsync(id);

    if (existingLecture == null)
        return NotFound();

    await _repository.DeleteAsync(existingLecture);

    return Ok(); // NoContent() would be fine too (204)
}
```

LecturesController

```
[HttpGet( template: "{id}/evaluations")] // /api/lectures/1/evaluations
[ProducesResponseType( statusCode: StatusCodes.Status200OK)]
[ProducesResponseType( statusCode: StatusCodes.Status404NotFound)]
public async Task<ActionResult<IEnumerable<Evaluation>>> GetEvaluations(int id)
{
    var lecture = await _repository.FindByIdAsync(id);

    if (lecture == null)
        return NotFound();

    var evaluations :List<Evaluation>  = await _evaluationRepository.FindByLectureAsync(id);

    return Ok(evaluations);
}
```

EvaluationsController

```
[HttpPost] // /api/evaluations
[ProducesResponseType( statusCode: StatusCodes.Status200OK)]
[ProducesResponseType( statusCode: StatusCodes.Status400BadRequest)]
public async Task<ActionResult<Evaluation>> Post(Evaluation evaluation)
{
    var lecture = await _lectureRepository.FindByIdAsync(evaluation.LectureId);

    if (lecture == null)
        return BadRequest( error: "Lecture not found");

    var createdEvaluation = await _repository.AddAsync(evaluation);

    // notify connected clients that there was a new evaluation added
    _lectureHub.Clients.Group(lecture.Id.ToString()).EvaluationAdded(
        new ILectureHubClient.EvaluationAddedParams() { Evaluation = createdEvaluation }
    );

    return CreatedAtAction(
        nameof(Get), routeValues: new { id = createdEvaluation.Id }, createdEvaluation);
}
```

MySqlLectureRepository

```csharp
public class MySqlLectureRepository : ILectureRepository
{
    private readonly MySqlDbContext _context;

    public MySqlLectureRepository(MySqlDbContext mysqlDbContext)
    {
        _context = mysqlDbContext;
    }

    public async Task<List<Lecture>> FindAllAsync()
    {
        return await _context.Lectures.ToListAsync();
    }
```

MySqlLectureRepository

```
public async Task<Lecture> AddAsync(Lecture entity)
{
    _context.Lectures.Add(entity);

    await _context.SaveChangesAsync();

    return entity;
}

public async Task<Lecture?> FindByIdAsync(int id)
{
    return await _context.Lectures.FindAsync(id);
}
```

MySqlLectureRepository

```
public async Task<Lecture> UpdateAsync(Lecture entity)
{
    // in order for ef core update to work we need to retrieve the entity from the
    // database first and then proceed to make changes to it
    var existing :Lecture? = await _context.Lectures.FindAsync(entity.Id);

    if (existing == null)
        throw new ArgumentException("Lecture does not exist");

    // we simply map all values from the domain entity to the ef core entity
    _context.Entry(existing).CurrentValues.SetValues(entity);

    // update might fail if we pass "entity" here because entity might be a POCO instance
    // update only works if we pass an ef core initialised entity
    _context.Lectures.Update(existing);

    await _context.SaveChangesAsync();

    return existing;
}
```

MySqlLectureRepository

```
    public async Task DeleteAsync(Lecture entity)
    {
        // in order for ef core update to work we need to retrieve the entity from the
        // database first and then proceed to make changes to it
        var existing :Lecture? = await _context.Lectures.FindAsync(entity.Id);

        if (existing == null)
            throw new ArgumentException("Lecture does not exist");

        _context.Lectures.Remove(existing);
        await _context.SaveChangesAsync();
    }
}
```

MySqlEvaluationRepository

```
public async Task<List<Evaluation>> FindByLectureAsync(int lectureId)
{
    return await _context.Evaluations // DbSet<Evaluation>
        .Where(e :Evaluation  => e.LectureId == lectureId) // IQueryable<Evaluation>
        .ToListAsync(); // Task<List<...>>
}
```

EF-Core commands (Package Manager Console)

- Add a new migration
  - Add-Migration InitialCreate

- Update the database
  - Update-Database

That‘s it ☺
…for now