Look, no Mocks!
Functional TDD with F#

Mark Seemann
http://blog.ploeh.dk
@ploeh

TDD is dead!

http://david.heinemeierhansson.com/2014/tdd-is-dead-long-live-testing.html

Is TDD dead?

A system you can’t test

Detestable

http://martinfowler.com/bliki/Detestable.html

```
public IHttpActionResult Post(ReservationRendition rendition)
{
    DateTimeOffset requestedDate;
    if (!DateTimeOffset.TryParse(rendition.Date, out requestedDate))
        return this.BadRequest("Invalid date.");

    var min = requestedDate.Date;
    var max = requestedDate.Date.AddDays(1);

    using (var ctx = new ReservationsContext())
    {
        var reservedSeats = (from r in ctx.Reservations
                             where min <= r.Date && r.Date < max
                             select r.Quantity)
                            .DefaultIfEmpty(0)
                            .Sum();
        if (rendition.Quantity + reservedSeats > capacity)
            return this.StatusCode(HttpStatusCode.Forbidden);
```

```
public IHttpActionResult Post(ReservationRendition rendition)
{
    DateTimeOffset requestedDate;
    if (!DateTimeOffset.TryParse(rendition.Date, out requestedDate))
        return this.BadRequest("Invalid date.");

    var min = requestedDate.Date;
    var max = requestedDate.Date.AddDays(1);

    using (var ctx = new ReservationsContext())
    {
        var reservedSeats = (from r in ctx.Reservations
                             where min <= r.Date && r.Date < max
                             select r.Quantity)
                            .DefaultIfEmpty(0)
                            .Sum();
        if (rendition.Quantity + reservedSeats > capacity)
            return this.StatusCode(HttpStatusCode.Forbidden);

        ctx.Reservations.Add(new Reservation
```

```
public IHttpActionResult Post(ReservationRendition rendition)
{
    DateTimeOffset requestedDate;
    if (!DateTimeOffset.TryParse(rendition.Date, out requestedDate))
        return this.BadRequest("Invalid date.");

    var min = requestedDate.Date;
    var max = requestedDate.Date.AddDays(1);

    using (var ctx = new ReservationsContext())
    {
        var reservedSeats = (from r in ctx.Reservations
                             where min <= r.Date && r.Date < max
                             select r.Quantity)
                            .DefaultIfEmpty(0)
                            .Sum();
        if (rendition.Quantity + reservedSeats > capacity)
            return this.StatusCode(HttpStatusCode.Forbidden);

        ctx.Reservations.Add(new Reservation
        {
```

```
public IHttpActionResult Post(ReservationRendition rendition)
{
    DateTimeOffset requestedDate;
    if (!DateTimeOffset.TryParse(rendition.Date, out requestedDate))
        return this.BadRequest("Invalid date.");

    var min = requestedDate.Date;
    var max = requestedDate.Date.AddDays(1);

    using (var ctx = new ReservationsContext())
    {
        var reservedSeats = (from r in ctx.Reservations
                             where min <= r.Date && r.Date < max
                             select r.Quantity)
                            .DefaultIfEmpty(0)
                            .Sum();
        if (rendition.Quantity + reservedSeats > capacity)
            return this.StatusCode(HttpStatusCode.Forbidden);

        ctx.Reservations.Add(new Reservation
        {
            Date = requestedDate,
```

```
public IHttpActionResult Post(ReservationRendition rendition)
{
    DateTimeOffset requestedDate;
    if (!DateTimeOffset.TryParse(rendition.Date, out requestedDate))
        return this.BadRequest("Invalid date.");

    var min = requestedDate.Date;
    var max = requestedDate.Date.AddDays(1);

    using (var ctx = new ReservationsContext())
    {
        var reservedSeats = (from r in ctx.Reservations
                             where min <= r.Date && r.Date < max
                             select r.Quantity)
                            .DefaultIfEmpty(0)
                            .Sum();
        if (rendition.Quantity + reservedSeats > capacity)
            return this.StatusCode(HttpStatusCode.Forbidden);

        ctx.Reservations.Add(new Reservation
        {
            Date = requestedDate,
            Name = rendition.Name,
```

```
public IHttpActionResult Post(ReservationRendition rendition)
{
    DateTimeOffset requestedDate;
    if (!DateTimeOffset.TryParse(rendition.Date, out requestedDate))
        return this.BadRequest("Invalid date.");

    var min = requestedDate.Date;
    var max = requestedDate.Date.AddDays(1);

    using (var ctx = new ReservationsContext())
    {
        var reservedSeats = (from r in ctx.Reservations
                             where min <= r.Date && r.Date < max
                             select r.Quantity)
                            .DefaultIfEmpty(0)
                            .Sum();
        if (rendition.Quantity + reservedSeats > capacity)
            return this.StatusCode(HttpStatusCode.Forbidden);

        ctx.Reservations.Add(new Reservation
        {
            Date = requestedDate,
            Name = rendition.Name,
            Email = rendition.Email,
```

```
public IHttpActionResult Post(ReservationRendition rendition)
{
    DateTimeOffset requestedDate;
    if (!DateTimeOffset.TryParse(rendition.Date, out requestedDate))
        return this.BadRequest("Invalid date.");

    var min = requestedDate.Date;
    var max = requestedDate.Date.AddDays(1);

    using (var ctx = new ReservationsContext())
    {
        var reservedSeats = (from r in ctx.Reservations
                             where min <= r.Date && r.Date < max
                             select r.Quantity)
                            .DefaultIfEmpty(0)
                            .Sum();
        if (rendition.Quantity + reservedSeats > capacity)
            return this.StatusCode(HttpStatusCode.Forbidden);

        ctx.Reservations.Add(new Reservation
        {
            Date = requestedDate,
            Name = rendition.Name,
            Email = rendition.Email,
            Quantity = rendition.Quantity
```

```
    DateTimeOffset requestedDate;
    if (!DateTimeOffset.TryParse(rendition.Date, out requestedDate))
        return this.BadRequest("Invalid date.");

    var min = requestedDate.Date;
    var max = requestedDate.Date.AddDays(1);

    using (var ctx = new ReservationsContext())
    {
        var reservedSeats = (from r in ctx.Reservations
                             where min <= r.Date && r.Date < max
                             select r.Quantity)
                            .DefaultIfEmpty(0)
                            .Sum();
        if (rendition.Quantity + reservedSeats > capacity)
            return this.StatusCode(HttpStatusCode.Forbidden);

        ctx.Reservations.Add(new Reservation
        {
            Date = requestedDate,
            Name = rendition.Name,
            Email = rendition.Email,
            Quantity = rendition.Quantity
        });
```

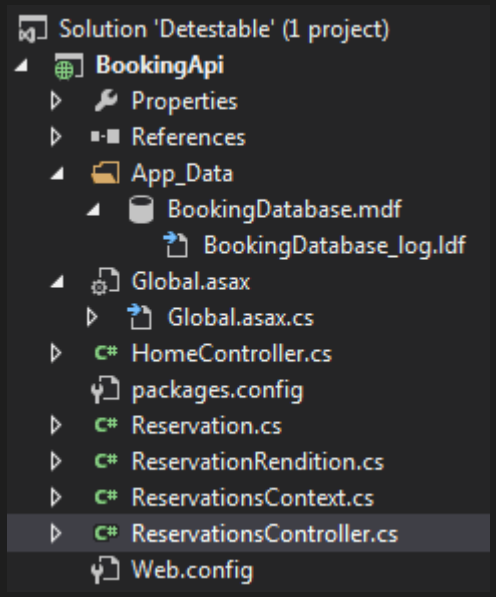
```
DateTimeOffset requestedDate;
if (!DateTimeOffset.TryParse(rendition.Date, out requestedDate))
    return this.BadRequest("Invalid date.");

var min = requestedDate.Date;
var max = requestedDate.Date.AddDays(1);

using (var ctx = new ReservationsContext())
{
    var reservedSeats = (from r in ctx.Reservations
                         where min <= r.Date && r.Date < max
                         select r.Quantity)
                        .DefaultIfEmpty(0)
                        .Sum();
    if (rendition.Quantity + reservedSeats > capacity)
        return this.StatusCode(HttpStatusCode.Forbidden);

    ctx.Reservations.Add(new Reservation
    {
        Date = requestedDate,
        Name = rendition.Name,
        Email = rendition.Email,
        Quantity = rendition.Quantity
    });
    ctx.SaveChanges();
```

```
    return this.BadRequest("Invalid date.");

var min = requestedDate.Date;
var max = requestedDate.Date.AddDays(1);

using (var ctx = new ReservationsContext())
{
    var reservedSeats = (from r in ctx.Reservations
                         where min <= r.Date && r.Date < max
                         select r.Quantity)
                        .DefaultIfEmpty(0)
                        .Sum();
    if (rendition.Quantity + reservedSeats > capacity)
        return this.StatusCode(HttpStatusCode.Forbidden);

    ctx.Reservations.Add(new Reservation
    {
        Date = requestedDate,
        Name = rendition.Name,
        Email = rendition.Email,
        Quantity = rendition.Quantity
    });
    ctx.SaveChanges();
}
```

```
    return this.BadRequest("Invalid date.");

var min = requestedDate.Date;
var max = requestedDate.Date.AddDays(1);

using (var ctx = new ReservationsContext())
{
    var reservedSeats = (from r in ctx.Reservations
                         where min <= r.Date && r.Date < max
                         select r.Quantity)
                        .DefaultIfEmpty(0)
                        .Sum();
    if (rendition.Quantity + reservedSeats > capacity)
        return this.StatusCode(HttpStatusCode.Forbidden);

    ctx.Reservations.Add(new Reservation
    {
        Date = requestedDate,
        Name = rendition.Name,
        Email = rendition.Email,
        Quantity = rendition.Quantity
    });
    ctx.SaveChanges();
}
```

```
var min = requestedDate.Date;
var max = requestedDate.Date.AddDays(1);

using (var ctx = new ReservationsContext())
{
    var reservedSeats = (from r in ctx.Reservations
                         where min <= r.Date && r.Date < max
                         select r.Quantity)
                        .DefaultIfEmpty(0)
                        .Sum();
    if (rendition.Quantity + reservedSeats > capacity)
        return this.StatusCode(HttpStatusCode.Forbidden);

    ctx.Reservations.Add(new Reservation
    {
        Date = requestedDate,
        Name = rendition.Name,
        Email = rendition.Email,
        Quantity = rendition.Quantity
    });
    ctx.SaveChanges();
}

return this.Ok();
```

```
    var min = requestedDate.Date;
    var max = requestedDate.Date.AddDays(1);

    using (var ctx = new ReservationsContext())
    {
        var reservedSeats = (from r in ctx.Reservations
                             where min <= r.Date && r.Date < max
                             select r.Quantity)
                            .DefaultIfEmpty(0)
                            .Sum();
        if (rendition.Quantity + reservedSeats > capacity)
            return this.StatusCode(HttpStatusCode.Forbidden);

        ctx.Reservations.Add(new Reservation
        {
            Date = requestedDate,
            Name = rendition.Name,
            Email = rendition.Email,
            Quantity = rendition.Quantity
        });
        ctx.SaveChanges();
    }

    return this.Ok();
}
```


Solution 'Detestable' (1 project)
BookingApi
Properties
References
App_Data
BookingDatabase.mdf
BookingDatabase_log.ldf
Global.asax
Global.asax.cs
HomeController.cs
packages.config
Reservation.cs
ReservationRendition.cs
ReservationsContext.cs
ReservationsController.cs
Web.config

Test-Induced Damage

```
    public IHttpActionResult Post(ReservationRendition rendition)
    {
        if (!this.apiValidator.Validate(rendition))
            return this.BadRequest("Invalid date.");

        var r = this.mapper.Map(rendition);
        var reservedSeats = this.repository.GetReservedSeats(r.Date);
        if (!this.maîtreD.CanAccept(rendition.Quantity, reservedSeats))
            return this.StatusCode(HttpStatusCode.Forbidden);

        this.repository.Save(r);

        return this.Ok();
    }
}
```

```
public interface IApiValidator
{
    bool Validate(ReservationRendition rendition);
}

public interface IReservationMapper
{
    Reservation Map(ReservationRendition rendition);
}

public interface IReservationRepository
{
    int GetReservedSeats(DateTimeOffset date);

    void Save(Reservation reservation);
}

public interface IMaîtreD
{
    bool CanAccept(int requestedSeats, int reservedSeats);
}
```

```
public interface IApiValidator
{
    bool Validate(ReservationRendition rendition);
}

public interface IReservationMapper
{
    Reservation Map(ReservationRendition rendition);
}

public interface IReservationRepository
{
    int GetReservedSeats(DateTimeOffset date);

    void Save(Reservation reservation);
}

public interface IMaîtreD
{
    bool CanAccept(int requestedSeats, int reservedSeats);
}

public class ApiValidator : IApiValidator
{
    public bool Validate(ReservationRendition rendition)
```

```
public interface IApiValidator
{
    bool Validate(ReservationRendition rendition);
}

public interface IReservationMapper
{
    Reservation Map(ReservationRendition rendition);
}

public interface IReservationRepository
{
    int GetReservedSeats(DateTimeOffset date);

    void Save(Reservation reservation);
}

public interface IMaîtreD
{
    bool CanAccept(int requestedSeats, int reservedSeats);
}

public class ApiValidator : IApiValidator
{
    public bool Validate(ReservationRendition rendition)
```

Unit Tests!

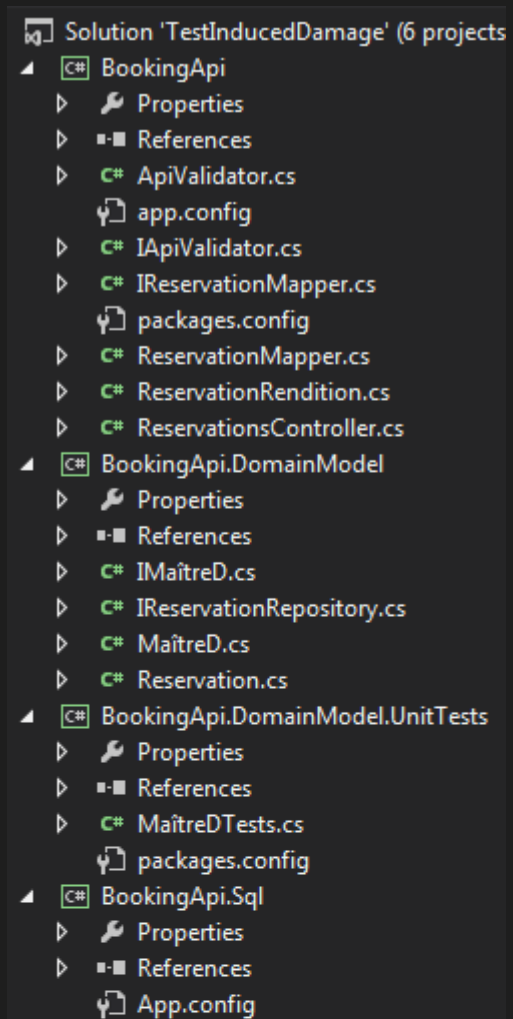
```
public class ApiValidatorTests
{
    [Fact]
    public void ValidateReturnsTrueWhenRenditionHasValidDate()
    {
        var date = new DateTimeOffset(
            new DateTime(2014, 7, 27, 21, 17, 35),
            TimeSpan.FromHours(2));
        var rendition = new ReservationRendition
        {
            Date = date.ToString("o"),
            Name = "Foo Baz",
            Email = "foo@fnaah.dk",
            Quantity = 9
        };
        IApiValidator sut = new ApiValidator();

        var actual = sut.Validate(rendition);

        Assert.True(actual);
    }

    [Fact]
    public void ValidateReturnsFalseWhenRenditionHasInvalidDate()
    {
```

```
public class ReservationMapperTests
{
    [Fact]
    public void MapReturnsCorrectResult()
    {
        var date = new DateTimeOffset(
            new DateTime(2014, 7, 27, 21, 25, 3),
            TimeSpan.FromHours(2));
        var rendition = new ReservationRendition
        {
            Date = date.ToString("o"),
            Name = "Ndøh Ploeh",
            Email = "ndøh@ploeh.dk",
            Quantity = 6
        };
        IReservationMapper sut = new ReservationMapper();

        var actual = sut.Map(rendition);

        Assert.Equal(date, actual.Date);
        Assert.Equal(rendition.Name, actual.Name);
        Assert.Equal(rendition.Email, actual.Email);
        Assert.Equal(rendition.Quantity, actual.Quantity);
    }
}
```

```
public class MaîtreDTests
{
    [Fact]
    public void CanAcceptReturnsTrueForFirstRequestUnderCapacity()
    {
        var capacity = 10;
        var sut = new MaîtreD(capacity);

        bool actual = sut.CanAccept(5, 0);

        Assert.True(actual);
    }

    [Fact]
    public void CanAcceptReturnsFalseForRequestAboveCapacity()
    {
        var capacity = 20;
        var sut = new MaîtreD(capacity);

        var actual = sut.CanAccept(30, 0);

        Assert.False(actual);
    }

    [Fact]
```

```
public class ReservationsControllerTests
{
    [Fact]
    public void SutIsApiController()
    {
        var sut =
            new ReservationsController(
                new Mock<IApiValidator>().Object,
                new Mock<IReservationRepository>().Object,
                new Mock<IMaîtreD>().Object,
                new Mock<IReservationMapper>().Object);
        Assert.IsAssignableFrom<ApiController>(sut);
    }

    [Fact]
    public void PostReturnsCorrectResponseForInvalidRendition()
    {
        var invalidRendition = new ReservationRendition
        {
            Date = "Not a valid date",
            Name = "Foo Bar",
            Email = "foo@ploeh.dk",
            Quantity = 3
        };
```

```
[Fact]
public void PostDoesNotSaveReservationWhenMaîtreCannotAccept()
{
    var date = new DateTimeOffset(
        new DateTime(2014, 7, 27, 20, 56, 57),
        TimeSpan.FromHours(2));
    var rendition = new ReservationRendition
    {
        Date = date.ToString("o"),
        Name = "Fnaah Ploeh",
        Email = "fnaah@ploeh.dk",
        Quantity = 2
    };
    var validatorStub = new Mock<IApiValidator>();
    validatorStub
        .Setup(v => v.Validate(rendition))
        .Returns(true);
    var repositoryMock = new Mock<IReservationRepository>();
    repositoryMock
        .Setup(r => r.GetReservedSeats(date))
        .Returns(10);
    var maîtreDStub = new Mock<IMaîtreD>();
    maîtreDStub
        .Setup(m => m.CanAccept(rendition.Quantity, 10))
        .Returns(false);
```


Solution 'TestInducedDamage' (6 projects
BookingApi
Properties
References
ApiValidator.cs
app.config
IApiValidator.cs
IReservationMapper.cs
packages.config
ReservationMapper.cs
ReservationRendition.cs
ReservationsController.cs
BookingApi.DomainModel
Properties
References
IMaîtreD.cs
IReservationRepository.cs
MaîtreD.cs
Reservation.cs
BookingApi.DomainModel.UnitTests
Properties
References
MaîtreDTests.cs
packages.config
BookingApi.Sql
Properties
References
App.config

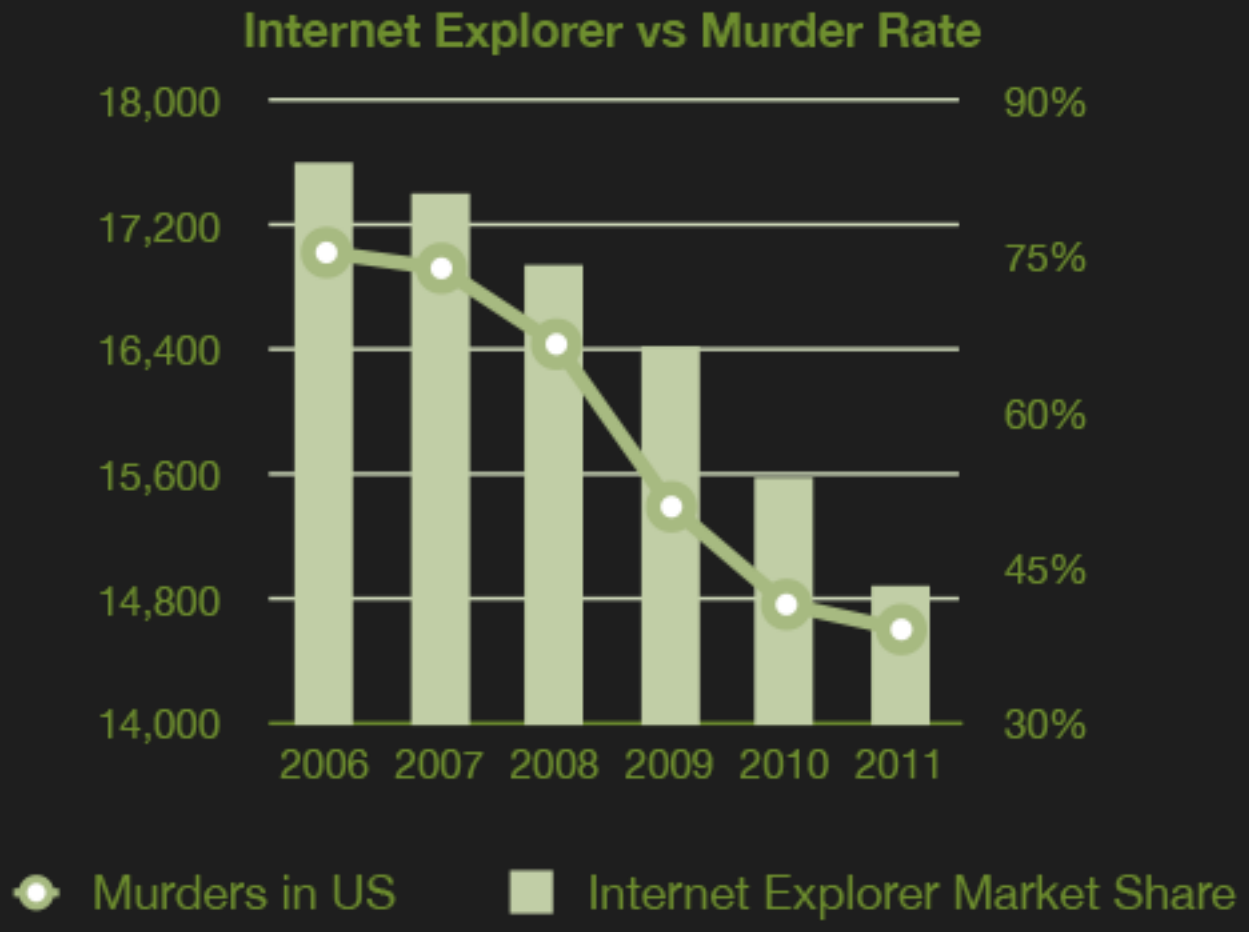
TDD

Test-Induced Damage

TDD

Test-Driven Design

Correlation



TDD

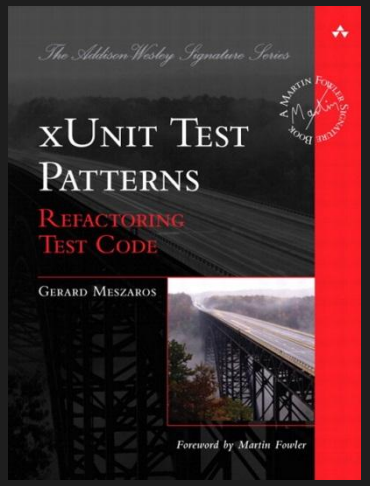
Test-Driven Design

Test-Induced Damage

TDD

Test-Driven Development

Functional Programming



http://bit.ly/xunitpatterns

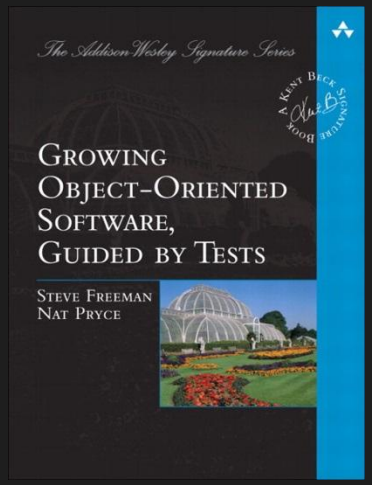
Stubs	Mocks

Stubs

Mocks


Stubs
Mocks

Stubs	Mocks
Queries	Commands
No side-effects	Side-effects



http://bit.ly/growingoos

F# is a Functional-First language

Stubs	Mocks
Queries	Commands
No side-effects	Side-effects

```
type Result<'TSuccess,'TFailure> =
    | Success of 'TSuccess
    | Failure of 'TFailure
```

```
[<Fact>]
let ``Post returns correct result on success`` () =
    let imp _ = Success ()
    use sut = new ReservationsController(imp)
    let rendition : ReservationRendition = {
        Date = "2014-08-09+2:00"
        Name = "Mark Seemann"
        Email = "mark@ploeh.dk"
        Quantity = 4 }

    let result : IHttpActionResult = sut.Post rendition

    test <@ result :? Results.OkResult @>
```

```
[<CLIMutable>]
type ReservationRendition = {
    Date : string
    Name : string
    Email : string
    Quantity : int }
```

```
type ReservationsController(imp) =
    inherit ApiController()
    member this.Post(rendition : ReservationRendition) =
        this.Ok() :> IHttpActionResult
```

```
[<Fact>]
let ``Validate.reservation returns correct result on valid date`` () =
    let rendition : ReservationRendition = {
        Date = "2014-08-09+2:00"
        Name = "Mark Seemann"
        Email = "mark@ploeh.dk"
        Quantity = 4 }

    let actual = Validate.reservation rendition

    let expected = Success {
        Date =
            DateTimeOffset(DateTime(2014, 8, 9), TimeSpan.FromHours 2.)
        Name = "Mark Seemann"
        Email = "mark@ploeh.dk"
        Quantity = 4 }
    test <@ expected = actual @>
```

```
type Reservation = {
    Date : DateTimeOffset
    Name : string
    Email : string
    Quantity : int }
```

```
let reservation (rendition : ReservationRendition) = Success {
    Date = rendition.Date |> DateTimeOffset.Parse
    Name = rendition.Name
    Email = rendition.Email
    Quantity = rendition.Quantity }
```

```
[<Fact>]
let ``Validate.reservation returns right result on invalid date`` () =
    let rendition : ReservationRendition = {
        Date = "Not a date"
        Name = "Mark Seemann"
        Email = "mark@ploeh.dk"
        Quantity = 4 }

    let actual = Validate.reservation rendition

    let expected : Result<Reservation, Error> =
        Failure(ValidationError("Invalid date."))
    test <@ expected = actual @>
```

```
type Error =
    | ValidationError of string
```

```
let reservation (rendition : ReservationRendition) =
    match rendition.Date |> DateTimeOffset.TryParse with
    | (true, date) -> Success {
        Date = date
        Name = rendition.Name
        Email = rendition.Email
        Quantity = rendition.Quantity }
    | _ -> Failure(ValidationError "Invalid date.")
```

```
[<Fact>]
let ``check returns right result at no prior reservations`` () =
    let capacity = 10
    let getReservedSeats _ = 0
    let reservation = {
        Date =
            DateTimeOffset(DateTime(2014, 8, 10), TimeSpan.FromHours 2.)
        Name = "Mark Seemann"
        Email = "mark@ploeh.dk"
        Quantity = 4 }

    let actual = Capacity.check capacity getReservedSeats reservation

    let expected : Result<Reservation, Error> = Success reservation
    test <@ expected = actual @>
```

```
let check capacity getReservedSeats reservation = Success reservation
```

```
[<Fact>]
let ``check returns right result at too little remaining capacity`` () =
    let capacity = 10
    let getReservedSeats _ = 7
    let reservation = {
        Date =
            DateTimeOffset(DateTime(2014, 8, 10), TimeSpan.FromHours 2.)
        Name = "Mark Seemann"
        Email = "mark@ploeh.dk"
        Quantity = 4 }

    let actual = Capacity.check capacity getReservedSeats reservation

    let expected : Result<Reservation, Error> = Failure CapacityExceeded
    test <@ expected = actual @>
```

```
type Error =
    | ValidationError of string
    | CapacityExceeded
```

```
let check capacity getReservedSeats reservation =
    let reservedSeats = getReservedSeats reservation.Date
    if capacity < reservation.Quantity + reservedSeats
    then Failure CapacityExceeded
    else Success reservation
```

```
[<Fact>]
let ``Post returns correct result on validation error`` () =
    let imp _ = Failure(ValidationError("Invalid date."))
    use sut = new ReservationsController(imp)
    let rendition : ReservationRendition = {
        Date = "2014-08-09+2:00"
        Name = "Mark Seemann"
        Email = "mark@ploeh.dk"
        Quantity = 4 }

    let result : IHttpActionResult = sut.Post rendition

    test <@ result :? Results.BadRequestErrorMessageResult @>
```

```
type ReservationsController
    (
        imp : ReservationRendition -> Result<unit, Error>
    ) =
    inherit ApiController()
    member this.Post(rendition : ReservationRendition) =
        match imp rendition with
        | Failure(ValidationError msg) ->
            this.BadRequest msg :> IHttpActionResult
        | _ -> this.Ok() :> IHttpActionResult
```

```
[<Fact>]
let ``Post returns correct result on capacity exceeded`` () =
    let imp _ = Failure CapacityExceeded
    use sut = new ReservationsController(imp)
    let rendition : ReservationRendition = {
        Date = "2014-08-09+2:00"
        Name = "Mark Seemann"
        Email = "mark@ploeh.dk"
        Quantity = 4 }

    let result : IHttpActionResult = sut.Post rendition

    test <@ result :? Results.StatusCodeResult @>
    test <@ (result :?> Results.StatusCodeResult).StatusCode =
                HttpStatusCode.Forbidden @>
```

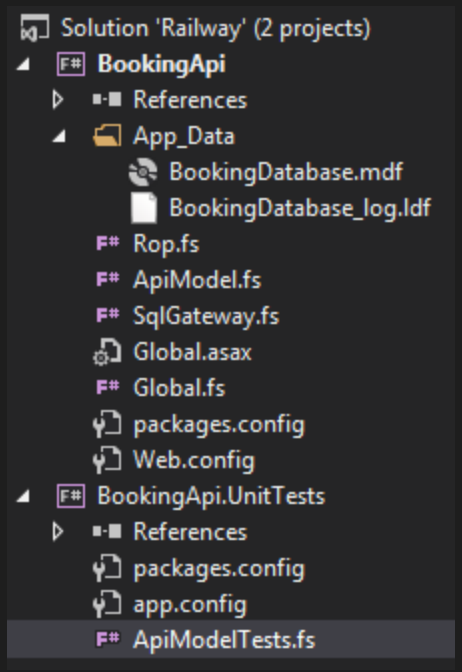
```
type ReservationsController(imp) =
    inherit ApiController()
    member this.Post(rendition : ReservationRendition) =
        match imp rendition with
        | Failure(ValidationError msg) ->
            this.BadRequest msg :> IHttpActionResult
        | Failure CapacityExceeded ->
            this.StatusCode HttpStatusCode.Forbidden :> IHttpActionResul
        | Success () -> this.Ok() :> IHttpActionResult
```

```
let getReservedSeats (date : DateTimeOffset) =
    let min = DateTimeOffset(date.Date           , date.Offset)
    let max = DateTimeOffset(date.Date.AddDays 1., date.Offset)

    use ctx = new ReservationsContext()
    query {
        for r in ctx.Reservations do
        where (min <= r.Date && r.Date < max)
        select r.Quantity }
    |> Seq.sum

let saveReservation (reservation : BookingApi.Reservation) =
    use ctx = new ReservationsContext()
    ctx.Reservations.Add(
        Reservation(
            Date = reservation.Date,
            Name = reservation.Name,
            Email = reservation.Email,
            Quantity = reservation.Quantity)) |> ignore
    ctx.SaveChanges() |> ignore
```

```fsharp
type Result<'TSuccess,'TFailure> =
    | Success of 'TSuccess
    | Failure of 'TFailure



let bind f x =
    match x with
    | Success s -> f s
    | Failure f -> Failure f



let map f x =
    match x with
    | Success s -> Success(f s)
    | Failure f -> Failure f
```

http://fsharpforfunandprofit.com/posts/recipe-part2

```
let imp =
    Validate.reservation
    >> Rop.bind (Capacity.check 10 SqlGateway.getReservedSeats)
    >> Rop.map SqlGateway.saveReservation
new ReservationsController(imp) :> _
```


Solution 'Railway' (2 projects)
BookingApi
References
App_Data
BookingDatabase.mdf
BookingDatabase_log.ldf
Rop.fs
ApiModel.fs
SqlGateway.fs
Global.asax
Global.fs
packages.config
Web.config
BookingApi.UnitTests
References
packages.config
app.config
ApiModelTests.fs

Functional-First: Few Mocks

Functions: Few interfaces

Function composition: Flexible

Q.E.D.

TDD is not dead