DDD в Golang

Дмитрий Гонозов, Yandex

Дмитрий Гонозов

- 5+ лет опыта на python

Дмитрий Гонозов

- 5+ лет опыта на python
- 2 года на golang

Дмитрий Гонозов

- 5+ лет опыта на python
- 2 года на golang
- переписывал legacy монолит на python

Дмитрий Гонозов

- 5+ лет опыта на python
- 2 года на golang
- переписывал legacy монолит на python
- запускал новые микросервисы на golang

Дмитрий Гонозов

- 5+ лет опыта на python
- 2 года на golang
- переписывал legacy монолит на python
- запускал новые микросервисы на golang
- все это с применением подхода DDD

Цель

- распространить подход DDD

Цель

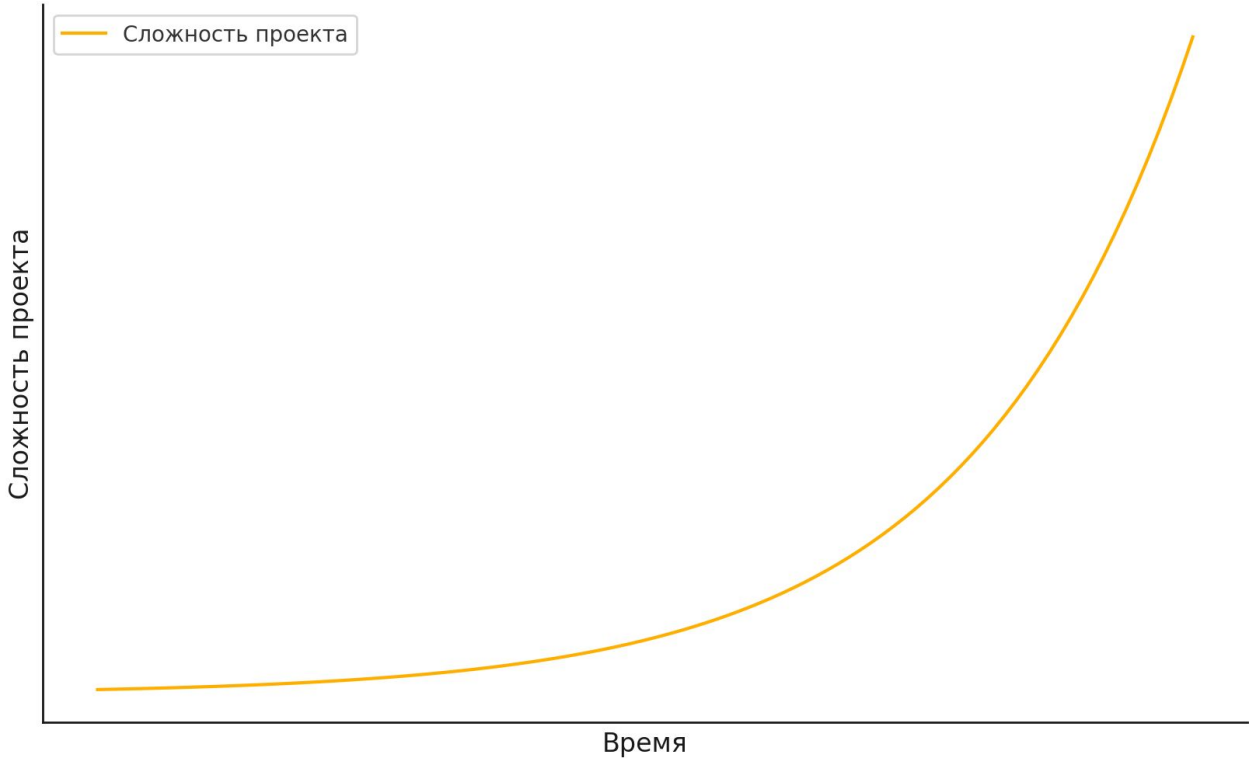
- распространить подход DDD
- скорее всего не для всех подойдет

Цель

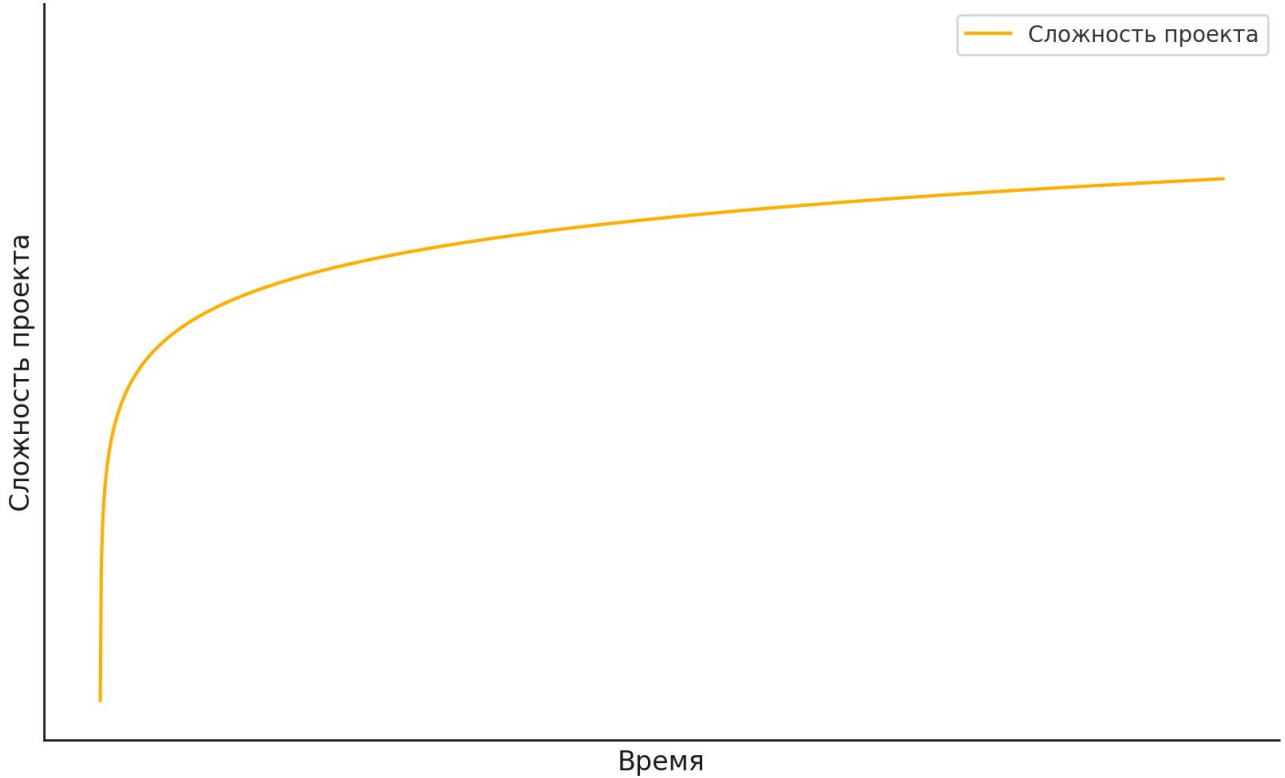
- распространить подход DDD
- скорее всего не для всех подойдет
- но для продуктовых команд подойдет точно

Зачем?

Рост сложности проекта со временем



Рост сложности проекта с DDD



Что такое DDD?

Domain-Driven Design (DDD) - это метод разработки, который помогает создавать проекты, хорошо отражающее реальный бизнес.

Он включает:

Что такое DDD?

Domain-Driven Design (DDD) - это метод разработки, который помогает создавать проекты, хорошо отражающее реальный бизнес.

Он включает:

- понимание бизнеса

Что такое DDD?

Domain-Driven Design (DDD) - это метод разработки, который помогает создавать проекты, хорошо отражающее реальный бизнес.

Он включает:

- понимание бизнеса
- общий язык

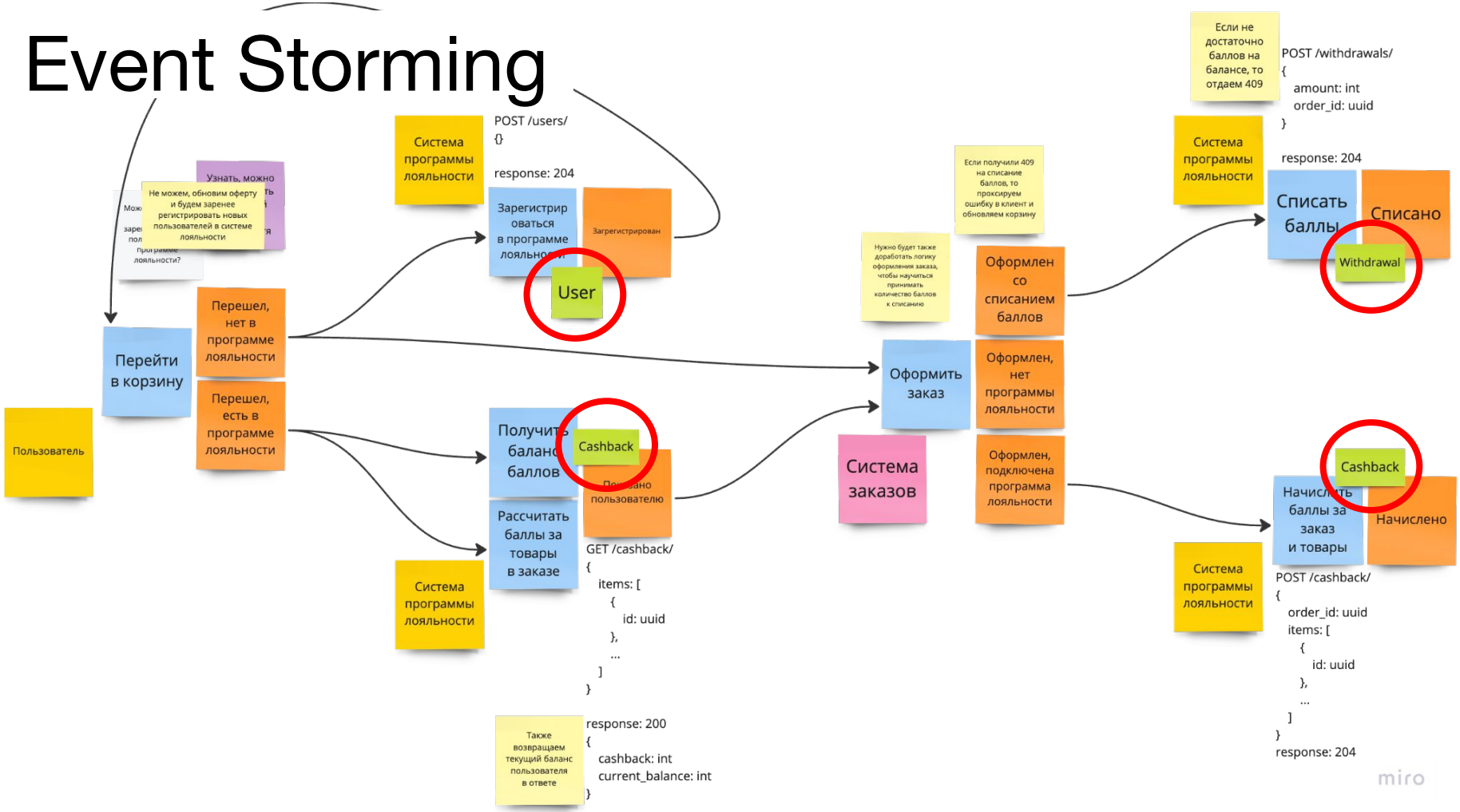
Что такое DDD?

Domain-Driven Design (DDD) - это метод разработки, который помогает создавать проекты, хорошо отражающее реальный бизнес.

Он включает:

- понимание бизнеса
- общий язык
- структурирование кода

Event Storming



Агрегат

```go
type Order struct {
    id     uuid.UUID
    userID uuid.UUID
    status OrderStatus
    items  []Item
}

func NewOrder(id uuid.UUID, userID uuid.UUID, status OrderStatus, items []Item)
(*Order, error) {
    if err := validateItems(items); err != nil {return nil, err}
    return &Order{
        id:     id,
        userID: userID,
        status: status,
        items:  items,
    }, nil
}
```

Агрегат

```go
func CreateOrder(userID uuid.UUID, items []Item) (*Order, error) {
    return NewOrder(uuid.New(), userID, OrderStatusCreated, items)
}

func (o *Order) ID() uuid.UUID {
    return o.id
}
func (o *Order) UserID() uuid.UUID {
    return o.userID
}
func (o *Order) Status() OrderStatus {
    return o.status
}
func (o *Order) Items() []Item {
    return o.items
}
```

Агрегат

```go
func (o *Order) Price() float64 {
    var total float64
    for _, item := range o.items {
        total += item.Price()
    }
    return total
}

func (o *Order) Pay(provider PaymentProvider) error {...}

func (o *Order) Deliver(provider DeliveryProvider) error {...}
```

SOLID

Single responsibility

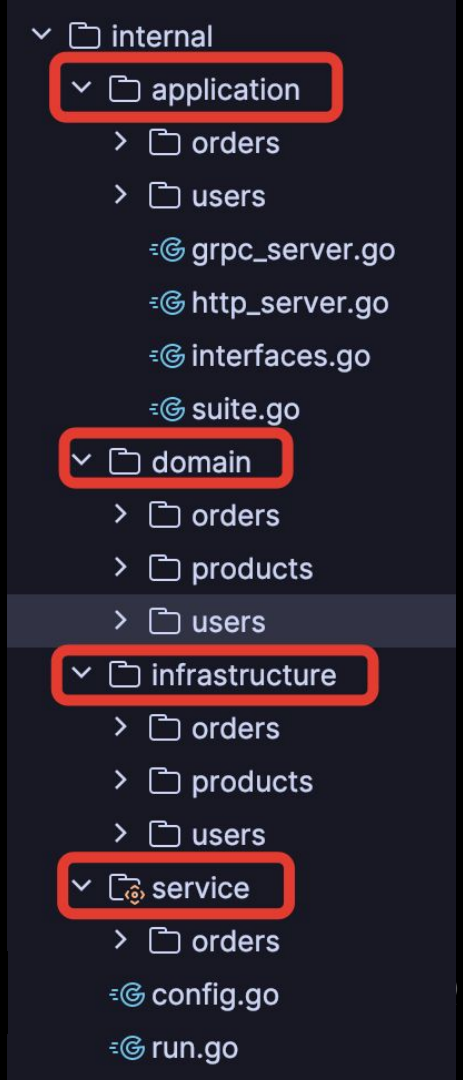
```go
func (o *Order) Price() float64 {
    var total float64
    for _, item := range o.items {
        total += item.Price()
    }
    return total
}

func (o *Order) Pay(provider PaymentProvider) error {...}

func (o *Order) Deliver(provider DeliveryProvider) error {...}
```

Open-Closed

```
func CreateOrder(userID uuid.UUID, items []Item) (*Order, error) {
    return NewOrder(uuid.New(), userID, OrderStatusCreated, items)
}

func (o *Order) ID() uuid.UUID {
    return o.id
}
func (o *Order) UserID() uuid.UUID {
    return o.userID
}
func (o *Order) Status() OrderStatus {
    return o.status
}
func (o *Order) Items() []Item {
    return o.items
}
```

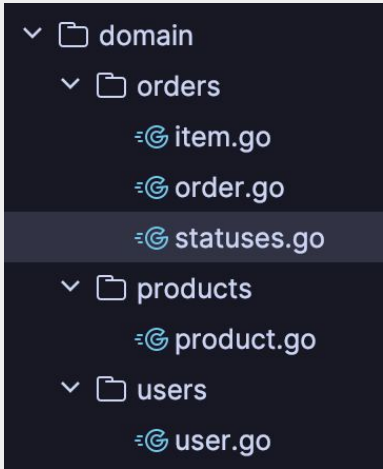
Слои

- domain
- infrastructure
- application
- service



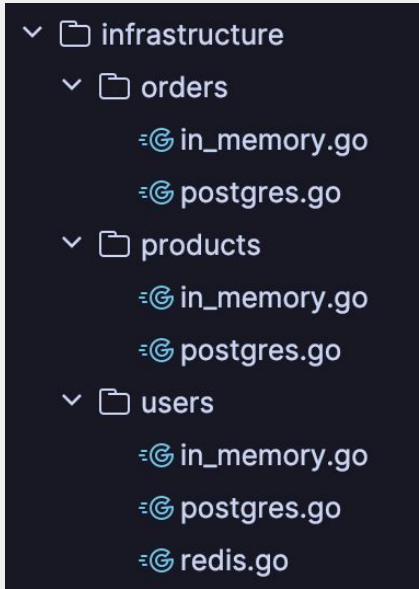
Domain

```go
type Order struct {
    id     uuid.UUID
    userID uuid.UUID
    status OrderStatus
    items  []Item
}

func NewOrder(id uuid.UUID, userID uuid.UUID, status OrderStatus, items []Item)
(*Order, error) {
    if err := validateItems(items); err != nil {return nil, err}
    return &Order{
        id:     id,
        userID: userID,
        status: status,
        items:  items,
    }, nil
}
```



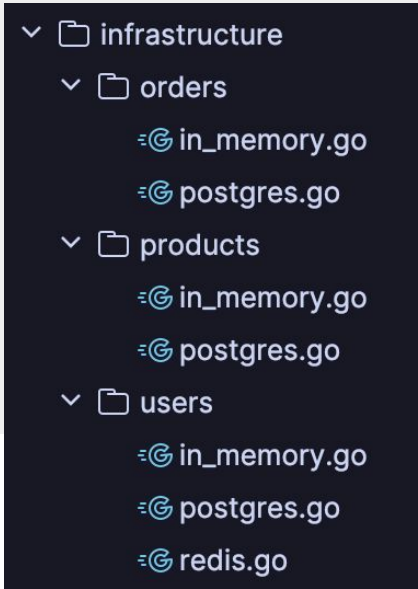
Infrastructure

```go
type order struct {
    userID uuid.UUID
    status orders.OrderStatus
    items  []orders.Item
}

type InMemoryRepo struct {
    orders map[uuid.UUID]order
    mu     sync.RWMutex
}

func NewInMemoryRepo() *InMemoryRepo {
    return &InMemoryRepo{
        orders: make(map[uuid.UUID]order),
    }
}
```



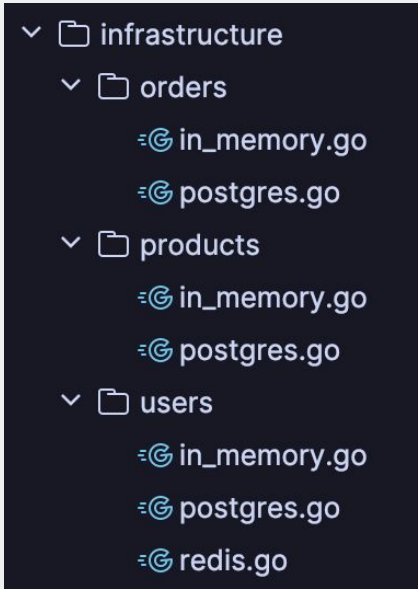
Infrastructure

```go
func (r *InMemoryRepo) SaveOrder(o orders.Order) error {
    r.mu.Lock()
    defer r.mu.Unlock()

    r.orders[o.ID()] = order{
        userID: o.UserID(),
        status: o.Status(),
        items:  o.Items(),
    }

    return nil
}
```



Infrastructure

```go
func (r *InMemoryRepo) GetOrder(id uuid.UUID) (*orders.Order, error) {
    r.mu.RLock()
    defer r.mu.RUnlock()

    o, ok := r.orders[id]
    if !ok {
        return nil, orders.ErrOrderNotFound
    }

    order, err := orders.NewOrder(id, o.userID, o.status, o.items)
    if err != nil {
        return nil, err
    }

    return order, nil
}
```

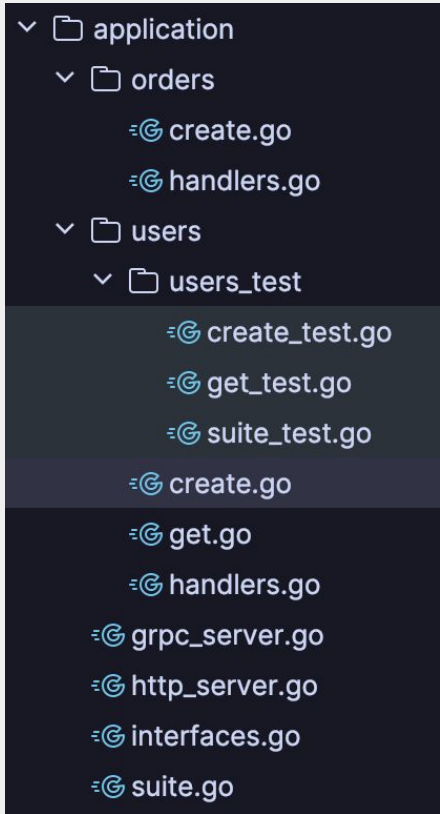


Mocks

```
@patch("src...get_start_system_context")
def test_get_start_content(get_start_system_context_mock: Mock):
    task = TaskDTO(
        id=TASK_ID,
        description=TASK_DESCRIPTION,
        start_files=[JS_START_FILE],
        user_files=[JS_START_FILE],
        answer_files=[JS_SOLUTION],
    )
    system_context, _ = get_content(task=task, prev_hint=None)
    assert isinstance(system_context, Mock)
    get_start_system_context_mock.assert_called_once_with(
        task.description,
        JS_DIFF,
        task.start_files,
        "",
    )
```

Application

```go
type Repository interface {
    SaveUser(u users.User) error
    GetUser(id uuid.UUID) (*users.User, error)
}

type UserHandlers struct {
    protobuf.UnimplementedUserServiceServer
    repo Repository
}

func SetupHandlers(repo Repository) UserHandlers {
    return UserHandlers{
        repo: repo,
    }
}
```

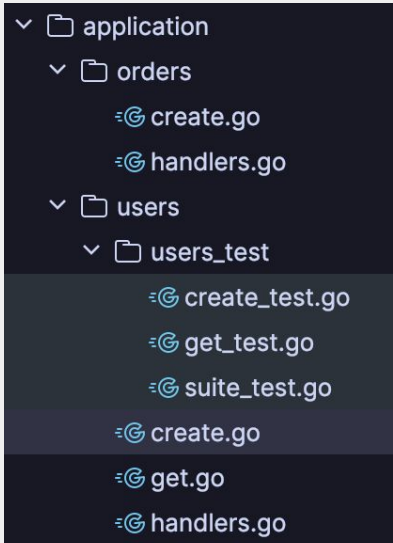


Application

```go
func (h UserHandlers) PostUsers(c echo.Context) error {
   var req openapi.CreateUserRequest
   if err := c.Bind(&req); err != nil {
      return err
   }
   user, err := domain.CreateUser(req.Name, string(req.Email))
   if err != nil {
      msg := err.Error()
      if errors.Is(err, domain.ErrInvalidUser) || errors.Is(err, domain.ErrUserValidation) {
         return c.JSON(http.StatusBadRequest, openapi.ErrorResponse{Message: &msg})
      }
      return c.JSON(http.StatusInternalServerError, openapi.ErrorResponse{Message: &msg})
   }
   if err := h.repo.SaveUser(*user); err != nil {
      msg := err.Error()
      return c.JSON(http.StatusInternalServerError, openapi.ErrorResponse{Message: &msg})
   }
   id := user.ID()
   return c.JSON(http.StatusCreated, openapi.CreateUserResponse{Id: &id})
}
```

Application

```go
func (h UserHandlers) CreateUser(
    _ context.Context,
    req *protobuf.CreateUserRequest,
) (*protobuf.CreateUserResponse, error) {
    user, err := domain.CreateUser(req.GetName(), req.GetEmail())
    if err != nil {
        if errors.Is(err, domain.ErrInvalidUser) || errors.Is(err, domain.ErrUserValidation) {
            return nil, status.Error(codes.InvalidArgument, err.Error())
        }
        return nil, status.Error(codes.Internal, err.Error())
    }

    if err := h.repo.SaveUser(*user); err != nil {
        return nil, status.Error(codes.Internal, err.Error())
    }

    return &protobuf.CreateUserResponse{
        Id: user.ID().String(),
    }, nil
}
```



SOLID

Interface Segregation

```
type Repository interface {
   SaveUser(u users.User) error
   GetUser(id uuid.UUID) (*users.User, error)
}

type UserHandlers struct {
   protobuf.UnimplementedUserServiceServer
   repo Repository
}

func SetupHandlers(repo Repository) UserHandlers {
   return UserHandlers{
      repo: repo,
   }
}
```

Dependency Inversion

```
type ServerSuite struct {
    suite.Suite
    HTTPServer   *echo.Echo
    UsersRepo    *usersInfra.InMemoryRepo
    OrdersRepo   *ordersInfra.InMemoryRepo
    ProductsRepo *productsInfra.InMemoryRepo
}

func (s *ServerSuite) SetupSuite() {
    s.UsersRepo = usersInfra.NewInMemoryRepo()
    s.OrdersRepo = ordersInfra.NewInMemoryRepo()
    s.ProductsRepo = productsInfra.NewInMemoryRepo()
    s.HTTPServer = SetupHTTPServer(s.UsersRepo, s.OrdersRepo, s.ProductsRepo)
}
```

Dependency Inversion

```
type UsersSuite struct {
    suite.Suite
    application.ServerSuite
    UserHandlers users.UserHandlers
}

func (s *UsersSuite) SetupSuite() {
    s.ServerSuite.SetupSuite()
    s.UserHandlers = users.SetupHandlers(s.UsersRepo)
}

func TestUsersSuite(t *testing.T) {
    suite.Run(t, new(UsersSuite))
}
```

Dependency Inversion

```go
func (s *UsersSuite) TestCreateUser() {
   s.Run("HTTP", func() {
      userReq := openapi.CreateUserRequest{
         Name:  "John Doe",
         Email: "john.doe@example.com",
      }
      reqBody, _ := json.Marshal(userReq)

      req := httptest.NewRequest(http.MethodPost, "/users", bytes.NewBuffer(reqBody))
      req.Header.Set(echo.HeaderContentType, echo.MIMEApplicationJSON)
      rec := httptest.NewRecorder()
      s.HTTPServer.ServeHTTP(rec, req)

      s.Require().Equal(http.StatusCreated, rec.Code)
      var resp openapi.CreateUserResponse
      err := json.Unmarshal(rec.Body.Bytes(), &resp)
      s.Require().NoError(err)
      s.Require().NotEqual("", resp.Id)
      s.Require().NotEqual(uuid.Nil, resp.Id)
   })
```

Dependency Inversion

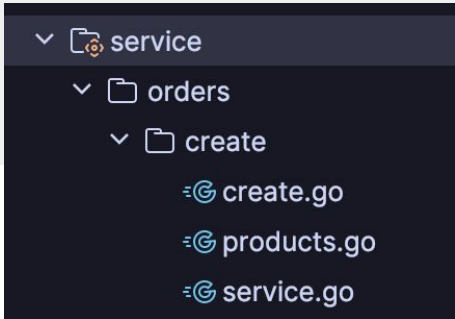
```
    s.Run("GRPC", func() {
        userReq := protobuf.CreateUserRequest{
            Name:  "John Doe",
            Email: "john.doe@example.com",
        }
        resp, err := s.UserHandlers.CreateUser(context.Background(), &userReq)

        s.Require().NoError(err)
        s.Require().NotEqual("", resp.GetId())
        s.Require().NotEqual(uuid.Nil, resp.GetId())
    })
}
```

Dependency Inversion

```
func startServers(ctx context.Context, g *errgroup.Group, cfg Config) error {
   listener, err := net.Listen("tcp", "0.0.0.0:"+cfg.Server.Port)
   if err != nil {
      return fmt.Errorf("failed to listen: %w", err)
   }

   m := cmux.New(listener)
   grpcListener :=
m.MatchWithWriters(cmux.HTTP2MatchHeaderFieldSendSettings("content-type",
"application/grpc"))
   httpListener := m.Match(cmux.Any())

   userRepo := usersInfra.NewPostgresRepo()
   productRepo := productsInfra.NewPostgresRepo()
   orderRepo := ordersInfra.NewPostgresRepo()

   httpServer := application.SetupHTTPServer(userRepo, orderRepo, productRepo)
   grpcServer := application.SetupGRPCServer(userRepo, orderRepo, productRepo)
```

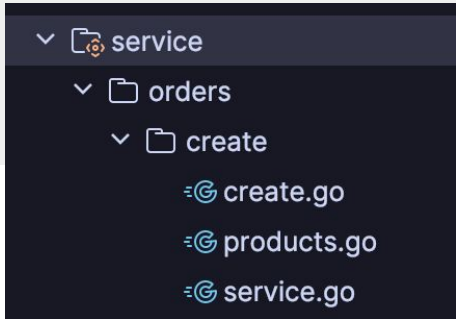
Service

```go
type orderRepo interface {
    SaveOrder(o orders.Order) error
    GetOrder(id uuid.UUID) (*orders.Order, error)
}

type userRepo interface {
    GetUser(id uuid.UUID) (*users.User, error)
}

type productRepo interface {
    GetProductsForUpdate(ids []uuid.UUID) ([]products.Product, error)
    SaveProducts(ps []products.Product) error
    CancelUpdate()
}
```



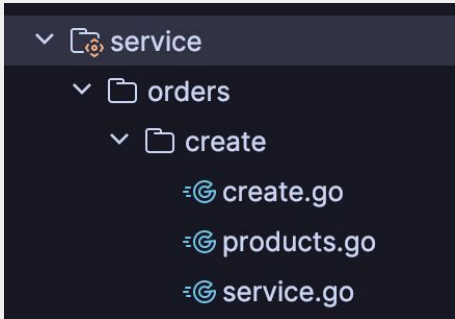
Service

```go
type OrderCreationService struct {
    orderRepo   orderRepo
    userRepo    userRepo
    productRepo productRepo
}

func NewOrderCreationService(or orderRepo, ur userRepo, pr productRepo)
*OrderCreationService {
    return &OrderCreationService{
        orderRepo:   or,
        userRepo:    ur,
        productRepo: pr,
    }
}
```



Service

```go
func (s *OrderCreationService) CreateOrder(userID uuid.UUID, items []Item) (*orders.Order,
error) {
    // check if user exists
    _, err := s.userRepo.GetUser(userID)
    if err != nil {return nil, err}

    defer s.productRepo.CancelUpdate()
    ps, err := s.reserveProducts(items)
    if err != nil {return nil, err}

    orderItems, err := makeOrderItems(items, ps)
    if err != nil {return nil, err}
    order, err := orders.CreateOrder(userID, orderItems)
    if err != nil {return nil, err}
    if err = s.orderRepo.SaveOrder(*order); err != nil {return nil, err}

    if err = s.productRepo.SaveProducts(ps); err != nil {return nil, err}

    return order, nil
}
```



Структуры

- aggregate
- value object
- dto (data transfer object)

Агрегат

```go
type Order struct {
    id     uuid.UUID
    userID uuid.UUID
    status OrderStatus
    items  []Item
}

func NewOrder(id uuid.UUID, userID uuid.UUID, status OrderStatus, items []Item)
(*Order, error) {
    if err := validateItems(items); err != nil {return nil, err}
    return &Order{
        id:     id,
        userID: userID,
        status: status,
        items:  items,
    }, nil
}
```

Value object

```go
type Item struct {
    productID uuid.UUID
    name      string
    price     float64
    count     int
}

func NewItem(productID uuid.UUID, name string, price float64, count int) (*Item, error) {
    if name == "" {
        return nil, fmt.Errorf("%w: invalid name", ErrInvalidItem)
    }
    if price <= 0 {
        return nil, fmt.Errorf("%w: invalid price: %f", ErrInvalidItem, price)
    }
    return &Item{
        productID: productID,
        name:      name,
        price:     price,
        count:     count,
    }, nil
}
```

Value object

```go
func (i Item) ProductID() uuid.UUID {return i.productID}

func (i Item) Name() string {return i.name}

func (i Item) Price() float64 {return i.price}

func (i Item) Count() int {return i.count}

func (i Item) FullPrice() float64 {
   return i.price * float64(i.count)
}

func (i Item) WithName(name string) (*Item, error) {
   return NewItem(i.productID, name, i.price, i.count)
}
```

Value object

```
func (i Item) IsEqual(item Item) bool {
   return i.productID == i.ProductID() && i.name == item.Name() && i.price ==
item.Price() && i.count == item.Count()
}
```

DTO

```
type Item struct {
    ID    uuid.UUID
    Count int
}

func (s *OrderCreationService) CreateOrder(userID uuid.UUID, items []Item)
(*orders.Order, error) {
    ...
```

Итог

Итог

Преимущества:

- контроль сложности проекта

Итог

Преимущества:

- контроль сложности проекта
- абстракция от конкретных БД

Итог

Преимущества:

- контроль сложности проекта
- абстракция от конкретных БД
- тесты, которым можно доверять

Итог

Преимущества:

- контроль сложности проекта
- абстракция от конкретных БД
- тесты, которым можно доверять

Недостатки:

- порог вхождения

Итог

Преимущества:

- контроль сложности проекта
- абстракция от конкретных БД
- тесты, которым можно доверять

Недостатки:

- порог вхождения
- больше работ на старте проекта

Старт проекта

https://github.com/gonozov0/go-echo-ddd-template

- больше примеров кода по DDD
- HTTP и gRPC сервер
- golangci-lint с golden config
- ci для github
- precommit hooks
- README по настройке

Рефакторинг существующих проектов

Рефакторинг существующих проектов

- выделение доменов (платежи, заказы, cms)

Рефакторинг существующих проектов

- выделение доменов (платежи, заказы, cms)
- рефакторинг самого узкого из доменов:

Рефакторинг существующих проектов

- выделение доменов (платежи, заказы, cms)
- рефакторинг самого узкого из доменов:
  - оставить только тесты на ручки (интеграционные и желательно без моков)

Рефакторинг существующих проектов

- выделение доменов (платежи, заказы, cms)
- рефакторинг самого узкого из доменов:
  - оставить только тесты на ручки (интеграционные и желательно без моков)
  - разделение на слои

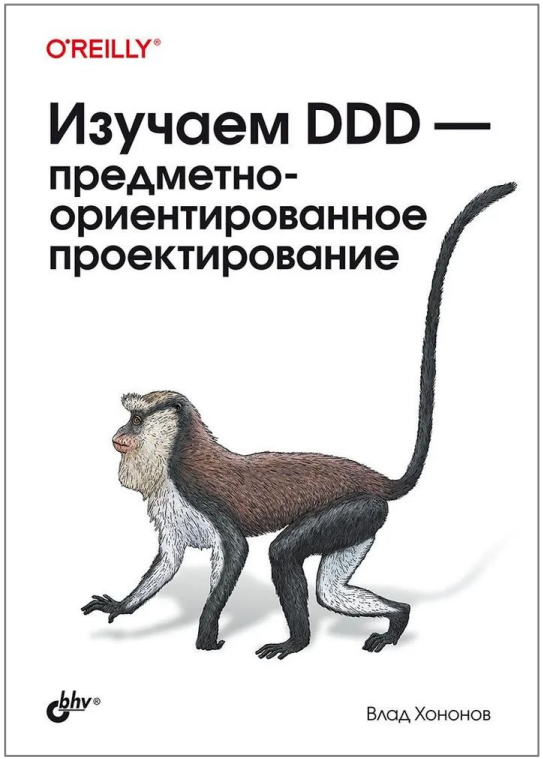
Рефакторинг существующих проектов

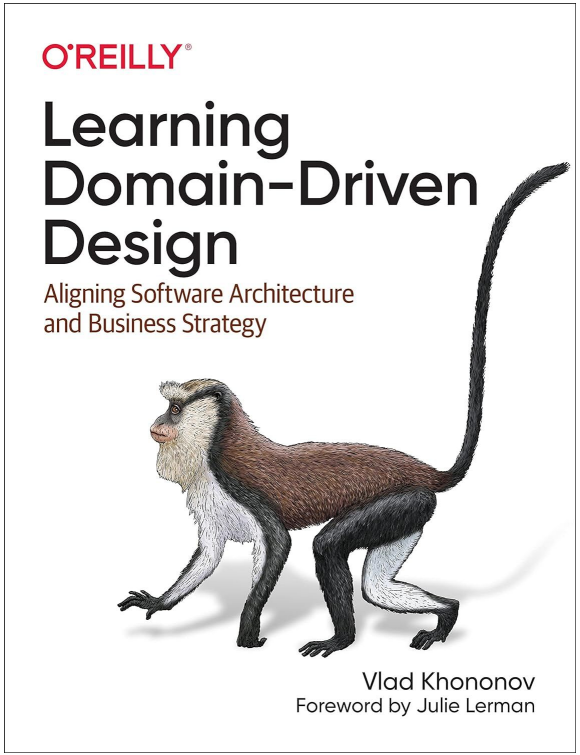
- выделение доменов (платежи, заказы, cms)
- рефакторинг самого узкого из доменов:
  - оставить только тесты на ручки (интеграционные и желательно без моков)
  - разделение на слои
  - написание агрегатов и репозиториев

Рефакторинг существующих проектов

- выделение доменов (платежи, заказы, cms)
- рефакторинг самого узкого из доменов:
  - оставить только тесты на ручки (интеграционные и желательно без моков)
  - разделение на слои
  - написание агрегатов и репозиториев
- и так по очереди все остальные домены

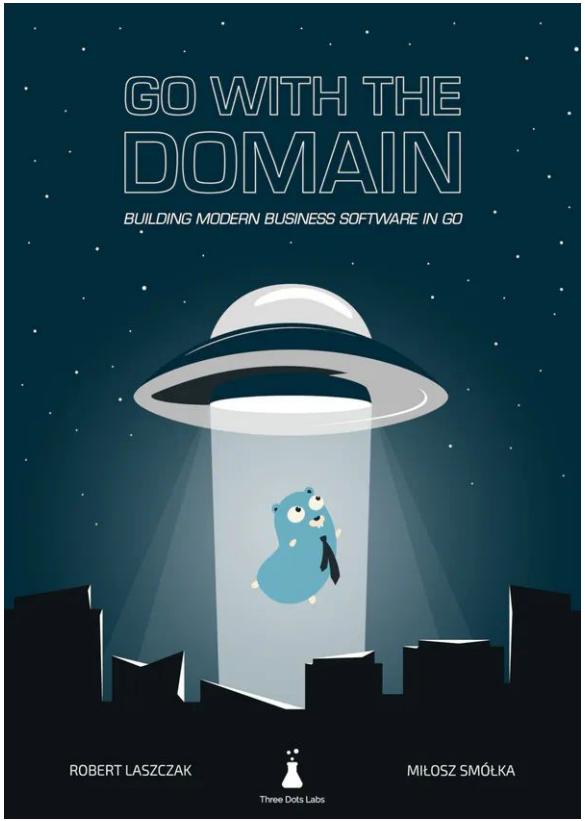
Roadmap


O'REILLY
Изучаем DDD —
предметно-
ориентированное
проектирование
bhv
Влад Хононов


O'REILLY
Learning
Domain-Driven
Design
Aligning Software Architecture
and Business Strategy
Vlad Khononov
Foreword by Julie Lerman

Roadmap

https://threedots.tech/go-with-the-domain/



Дмитрий Гонозов

Вопросы: t.me/gonozov0

Презентация: clck.ru/3CvVnQ