Вау! ИИ готовит к ЕГЭ по информатике Попробовать

Урок Спринт 1.10

Дженерики

Дженерики позволяют писать код, который сможет работать с различными типами данных. Дублировать его для каждого типа не придётся. Это упрощает и ускоряет процесс разработки.

Предположим, у нас есть функция, которая принимает слайс чисел и возвращает их сумму:

```
func sum(nums []int) int {
    total := 0
    for _, n := range nums {
        total += n
    }
    return total
}
```

Эта функция работает только со слайсами чисел. Если мы хотим использовать её для слайсов строк или других типов данных, нам придётся писать отдельную функцию для каждого типа данных. (Подробнее рассмотрим это на следующем уроке про сортировки.)

С помощью дженериков мы можем написать одну функцию, которая будет работать со слайсами любых типов данных:

```
func sum[T any](nums []T) T {
    var total T
    for _, n := range nums {
        total += n
    }
    return total
}
```

Здесь мы используем ключевое слово any и таким образом указываем, что функция принимает слайс любого типа данных. Затем пишем тип T, чтобы указать тип элементов в слайсе и тип возвращаемого значения. T может быть `int`, `string`, `byte`, и читать это можно без содержимого квадратных скобок. В функции мы используем оператор `+`, чтобы сложить элементы слайса.

Давайте перепишем пример с операциями базы данных на дженериках:

```
// Repository — обобщённый тип для работы с базой данных
type Repository[T any] struct {
    db *sqlx.DB
}

// NewRepository — создаёт новый экземпляр репозитория
func NewRepository[T any](db *sqlx.DB) *Repository[T] {
    return &Repository[T]{
        db: db,
    }
}

// Add — добавляет запись в базу данных
func (r *Repository[T]) Add(entity T, ctx context.Context) error {
    _, err := r.db.NamedExecContext(ctx, "INSERT INTO your_table_name_here (field1, field2) VALUES (:field1, :field2)", entity)
    return err
}

// GetById — получает запись из базы данных по идентификатору
func (r *Repository[T]) GetById(id int, ctx context.Context) (T, error) {
    var entity T
    err := r.db.GetContext(ctx, &entity, "SELECT * FROM your_table_name_here WHERE id = ? AND is_active = ?", id, true)
    if err != nil {
```

```
        return entity, err
    }

    return entity, nil
}

// Get — выполняет поиск записи в базе данных по параметрам
func (r *Repository[T]) Get(params T, ctx context.Context) T {
    var entity T
    r.db.GetContext(ctx, &entity, "SELECT * FROM your_table_name_here WHERE field1 = :field1 AND field2 = :field2", params)
    return entity
}

// GetAll — получает все записи из базы данных
func (r *Repository[T]) GetAll(ctx context.Context) ([]T, error) {
    var entities []T
    err := r.db.SelectContext(ctx, &entities, "SELECT * FROM your_table_name_here")
    if err != nil {
        return entities, err
    }
    return entities, nil
}

// Update — обновляет запись в базе данных
func (r *Repository[T]) Update(entity T, ctx context.Context) error {
    _, err := r.db.NamedExecContext(ctx, "UPDATE your_table_name_here SET field1 = :field1, field2 = :field2 WHERE id = :id", entity)
    return err
}
```

Теперь нам не нужно работать с шаблонизатором — достаточно просто использовать дженерик-данные.

Справка

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках сервиса, принадлежат АНО ДПО «Образовательные технологии Яндекса». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «Образовательные технологии Яндекса».
Пользовательское соглашение.

© 2018 – 2024 ООО «Яндекс»