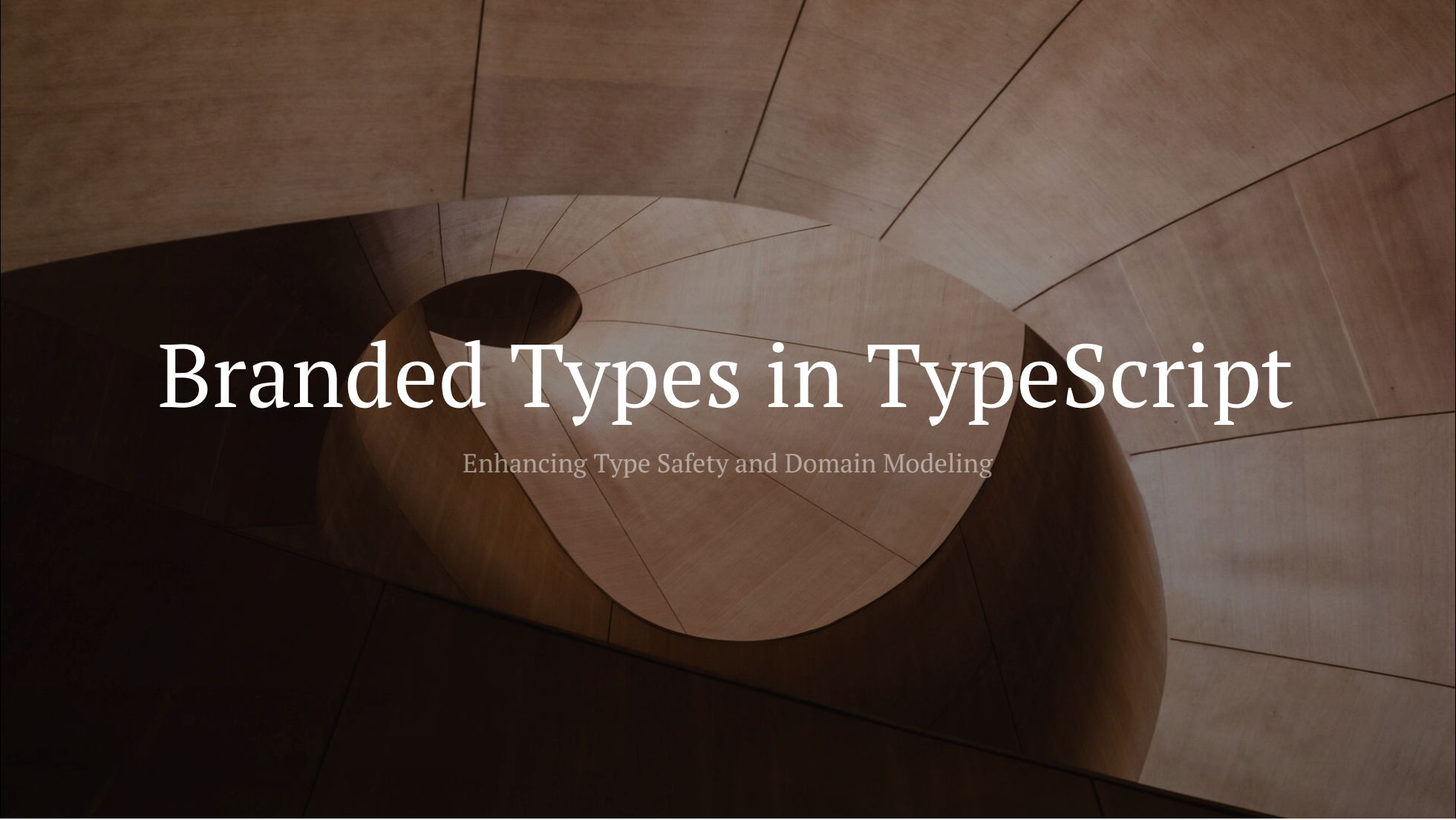

Branded Types in TypeScript

Enhancing Type Safety and Domain Modeling

What are Branded Types?

- A technique to create nominally typed values
- Adds an extra layer of type safety
- Helps prevent mixing up semantically different values of the same primitive type

Why Use Branded Types?

- Catch more errors at compile-time
- Improve code readability and self-documentation
- Model your domain more accurately

Basic Example: User ID

```
type UserId = string & { readonly brand: unique symbol }

function createUserID(id: string): UserId {
  // Type assertion to create a branded type
  return id as UserId
}

function deleteUser(id: UserId) {
  // Implementation...
}

const someString: string = 'PhyberApex'
deleteUser(someString)
```

```
Argument of type 'string' is not assignable to parameter of type 'UserId'.
  Type 'string' is not assignable to type '{ readonly brand: unique symbol; }'.
```

```
const validUserId = createUserID(someString)
deleteUser(validUserId)

export {}
```

Benefits of Branded Types

1. Type safety: Can’t pass a regular string as a UserID
2. Intention revealing: Clear distinction between different ID types
3. Compile-time checks: Errors caught early in the development process

Going Generic

```
type UserId = string & { readonly brand: unique symbol }

function createUserID(id: string): UserId {
  // Type assertion to create a branded type
  return id as UserId
}

function deleteUser(id: UserId) {
  // Implementation...
}

const someString: string = 'PhyberApex'
deleteUser(someString)

const validUserId = createUserID(someString)
deleteUser(validUserId)

export {}
```

Type Guards with Branded Types

```
type EmailAddress = string & { readonly brand: unique symbol }

function isEmailAddress(value: string): value is EmailAddress {
  return /^[^\s@]+@[^\s@][^\s.@]*\.[^\s@]+$/.test(value)
}

function sendEmail(email: EmailAddress) {
  console.log(`Sending email to ${email}`)
}

const email: string = 'user@example.com'
if (isEmailAddress(email)) {
  sendEmail(email)
}
else {
  console.log('Invalid email address')
}

export {}
```

Type Assertions with Branded Types

```
type EmailAddress = string & { readonly brand: unique symbol }

function isEmailAddress(value: string): value is EmailAddress {
  return /^[^\s@]+@[^\s@][^\s.@]*\.[^\s@]+$/.test(value)
}

function sendEmail(email: EmailAddress) {
  console.log(`Sending email to ${email}`)
}

const email: string = 'user@example.com'
if (isEmailAddress(email)) {
  sendEmail(email)
}
else {
  console.log('Invalid email address')
}

export {}
```

Type Assertions with Branded Types

```
type EmailAddress = string & { readonly brand: unique symbol }

function assertsEmailAddress(value: string): asserts value is EmailAddress {
  if (!/^[^\s@]+@[^\s@][^\s.@]*\.[^\s@]+$/.test(value))
    throw new Error('Not a valid email')
}

function sendEmail(email: EmailAddress) {
  console.log(`Sending email to ${email}`)
}

const email: string = 'user@example.com'
assertsEmailAddress(email)
sendEmail(email)

export {}
```

Best Practices

1. Use branded types for important domain concepts
2. Create factory functions for type safety
3. Use type guards and assertions to enhance runtime safety
4. With great power comes great responsibility!

Questions?

Thank you for your attention!