Budget: Ett PHP-projekt för ekonomihantering

1 **Namn**

Budget

2 **Ägare**

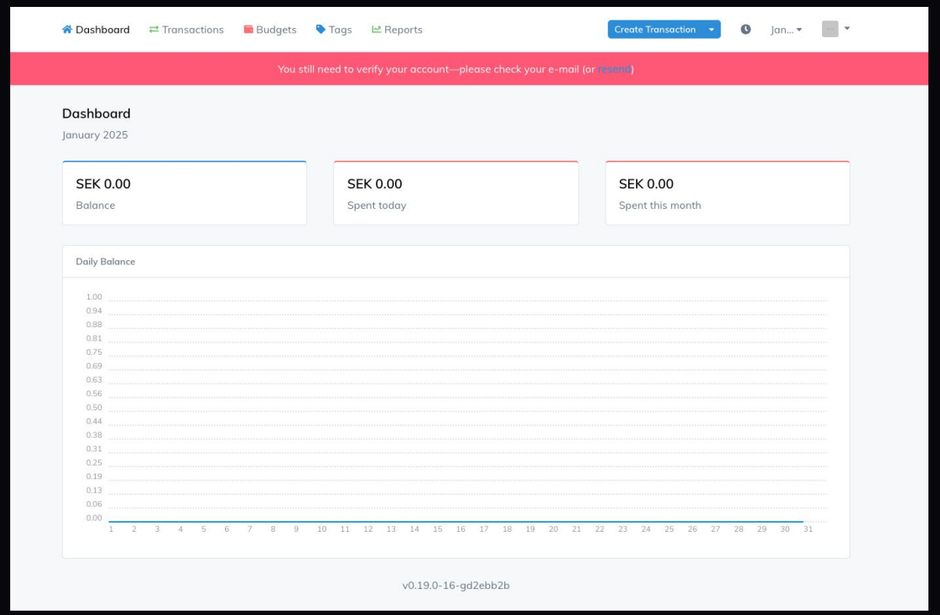
range-of-motion

3 **Språk**

PHP

4 **Syfte**

Hantera ekonomi (inkomster, utgifter, budgetar)



Kravspecifikation

Funktionella krav

Budget inkluderar användar- och transaktionshantering, stöd för kvitton och CSV-import. Projektet erbjuder också fleranvändarstöd, hantering av olika valutor och språk för en bred användargrupp.

Tekniska krav

Projektet har Docker-stöd och använder MySQL som databas. Det är värt att notera att det inte finns några explicita kravdokument, vilket ger flexibilitet i utvecklingen men kan också innebära utmaningar för testning och kvalitetssäkring.

Utvecklingsresa

1

Start: Oktober 2014

Projektet påbörjades med grundläggande funktioner som användarhantering, taggning och transaktionshantering.

2

Kontinuerlig utveckling

Över åren har projektet vuxit och förbättrats med nya funktioner och optimeringar.

3

Senaste ändring: 2024

Stöd för mexikanska pesos lades till, vilket visar på fortsatt utveckling och anpassning till användarnas behov.

4

Framtid

Planerad uppgradering till Laravel 11 för att hålla projektet uppdaterat och säkert.

Nuvarande teststrategi

Enhetstester

Använder PHPUnit för att testa enskilda komponenter och funktioner.

Funktions-/ Integrationstester

Testar samspelet mellan olika delar av systemet.

E2E-tester

Använder Laravel Dusk för att simulera användarinteraktioner.

Bugghantering

Använder GitHub Issues för att spåra och hantera buggar som upptäcks genom tester och manuella kontroller.

Testning

Enhetstester

Exempel inkluderar SpaceTest och HelperTest, som fokuserar på att verifiera funktionaliteten hos enskilda komponenter.

Feature/Integration

CreateEarningTest är ett exempel på hur större funktioner testas i samspel med andra delar av systemet.

Browser/E2E

SmokeTest via Dusk används för att simulera användarinteraktioner och verifiera systemets övergripande funktionalitet.

Exploratory testing

Manuell testning ledde till upptäckten av ett 500-fel på /budgets-sidan, vilket visar vikten av olika testmetoder.

Exempel på Enhetstest

```php
public function testAbbreviatedNameAttribute(): void
{
    $space = Space::factory()
        ->create([
            'name' => 'Hello world',
        ]);

    $this->assertEquals('Hel...', $space->abbreviated_name);
}
```

- Det testar en enda funktion (Single Responsibility)
- Det har ett tydligt förväntat resultat
- Det använder Laravel's factory-system för att skapa testdata

Exempel på feature test

```php
public function testSuccessfulCreation(): void
{
    $response = $this
        ->followingRedirects()
        ->actingAs($this->user)
        ->withSession(['space_id' => $this->space->id])
        ->postJson('/earnings', [
            'date' => date('Y-m-d'),
            'description' => 'Something for the test',
            'amount' => '9.99'
        ]);

    $response
        ->assertStatus(200);
}
```

- Det testar hela flödet från HTTP-request till respons
- Det simulerar en autentiserad användare med `actingAs()`
- Det testar API-endpoints med JSON-data

Exempel på browser - End to End test

```php
public function testBoot(): void
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/')
                ->assertSee('Log in');
    });
}
```

Detta är ett så kallat "smoke test" som verifierar att applikationen åtminstone startar och visar inloggningssidan.

Exempel på mockning

```php
public function testSuccessfullySent(): void
{
    $firstVerificationMailSentAt = date('Y-m-d H:i:s', strtotime('7 minute ago'));

    $user = User::factory()->create([
        'verification_token' => 'abc123',
        'last_verification_mail_sent_at' => $firstVerificationMailSentAt
    ]);

    // Förhindrar mailet från att sändas
    Mail::fake();

    (new SendVerificationMailAction())->execute($user->id);

    $this->assertGreaterThan(
        $firstVerificationMailSentAt,
        User::find($user->id)->last_verification_mail_sent_at
    );
}
```

Explorativ Testning

Upptäckter: 500-fel på /budgets-sidan, inkonsekventa felmeddelanden, problem med filstorlekar, e-postformat och språkhantering.

Rekommendationer: Robust felhantering, striktare validering, täckning av edge cases.

initiala tester och resultat till en början

92

Totalt antal tester

83

Gröna tester

3

Röda tester

157

Assertions

Resultat efter justering

93

Totalt antal tester

91

Gröna tester

02

Röda tester

164

Assertions

Initial test coverage

39.88%

Linjetäckning

Indikerar hur stor del av kodbasen som nås av tester.

35.35%

Funktions-

Täckning av funktioner och metoder.

22.81%

Klass-

Täckning av klasser och traits.

Nya tester i BudgetControllerTest

- user_can_view_budget_index: Besöker /budgets, förväntar sig HTTP 200, verifierar åtkomst till rätt space.
- user_can_view_create_budget_page: Besöker /budgets/create, förväntar sig HTTP 200, kontrollerar behörighet att skapa budget.
- user_can_store_new_budget: Skapar ny budget, sparar data i DB, omdirigerar och kopplar budget till rätt space/tag.

```php
class BudgetControllerTest extends TestCase
{
    protected function setUp(): void
    {
        parent::setUp();

        // Skapa en currency först (eftersom space behöver en currency_id)
        $currency = Currency::factory()->create();

        // Skapa space med den skapade currency:n
        $space = Space::factory()->create([
            'currency_id' => $currency->id
        ]);

        session(['space_id' => $space->id]);
    }

    /** @test */
    public function user_can_view_budget_index()
    {
        // Arrange
        $user = User::factory()->create();
        $user->spaces()->attach(session('space_id'));

        // Act
        $response = $this->actingAs($user)
            ->get(route('budgets.index'));

        // Assert
        $response->assertStatus(200);
    }
```

Efter tillägg av några tester

40.55%

Linjetäckning

Indikerar hur stor del av kodbasen som nås av tester.

36.03%

Funktions-

Täckning av funktioner och metoder.

22.81%

Klass-

Täckning av klasser och traits.

Styrkor i Testningen

1 **Mappstruktur**

Lätt att följa och hitta i strukturen

2 **Mockning**

Isolerade tester utan beroenden.

3 **Factories**

Konsekvent hantering av testdata.

Identifierade Problem

- Låg testtäckning, särskilt för controllers och API
- Bristfällig felhantering i tester
- Problem med icke-unika värden i testdatabasen
- Avsaknad av dokumentation, ingen information om testning i contributors.md

Sammanfattning

Status

Budget-projektet visar potential men har för närvarande låg testtäckning. Det finns goda möjligheter till förbättring och vidareutveckling.

Åtgärder

Prioritera uppgradering till Laravel 11 och stärk den övergripande teststrategin för att öka projektets robusthet och tillförlitlighet.

Värde

Projektet har en strukturerad kodbas, en solid grund för vidareutveckling och praktiskt Docker-stöd, vilket underlättar deployment och utveckling.

Nästa steg

Fokusera på att åtgärda identifierade buggar och öka den totala kodtäckningen för att säkerställa projektets långsiktiga hållbarhet och kvalitet.