4. Implementación

En este apartado se describen los componentes más relevantes del proyecto, tanto del **backend en Laravel** como del **frontend en Vue 3**, junto con ejemplos reales de código utilizados.

4.1. Autenticación con Laravel Sanctum

Agatha-api utiliza **Laravel Sanctum** para gestionar autenticación mediante tokens.

El flujo es el siguiente:

1. El usuario inicia sesión desde el frontend.
2. El backend valida las credenciales.
3. Si son correctas, genera un **token con duración de 12 horas**.
4. El token se guarda en LocalStorage y se envía en cada petición con Axios.

Controlador AuthController

```php
public function login(Request $request)
{
    $data = $request->validate([
        'email' => 'required|email',
        'password' => 'required',
    ]);

    if (!Auth::attempt($data)) {
        return response()->json(['success' => false, 'message' => 'Incorrecto'], 401);
    }

    $user = $request->user();
    $token = $user->createToken('api-token', ['*'], now()->addHours(12))->plainTextToken;

    return response()->json([
        'success' => true,
        'token' => $token,
        'user' => $user,
    ]);
}
```

4.2. Registro y actualización de usuario

El registro exige contraseñas fuertes (mínimo 8 caracteres, mayúsculas, minúsculas, números y símbolos).

Validación de registro:

```
$data = $request->validate([
    'name' => 'required|string|max:255',
    'email' => 'required|string|email|max:255|unique:users,email',
    'password' => [
        'required',
        'confirmed',
        Password::min(8)->letters()->mixedCase()->numbers()->symbols(),
    ],
]);
```

4.3. Frontend de Autenticación (Vue 3 + Pinia)

El frontend almacena el token en LocalStorage y lo añade automáticamente con un interceptor Axios.

Interceptor en api.js

```
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('token')
  if (token) {
    config.headers.Authorization = `Bearer ${token}`
  }
  return config
})
```

Ejemplo real: vista LoginView.vue

```
async function handleLogin() {
  const data = await login(email.value, password.value)

  if (!data.success) {
    message.value = 'Credenciales incorrectas'
    return
  }

  authStore.setUser(data.user)
  authStore.setToken(data.token)
  router.push('/home')
}
```

4.4. Rutas API

```
Route::post('/register', [AuthController::class, 'register']);
Route::post('/login', [AuthController::class, 'login']);

Route::middleware('auth:sanctum')->group(function () {
    Route::prefix('story')->group(function () {
        Route::get('/list', [StoryController::class, 'index']);
        Route::post('/store', [StoryController::class, 'store']);
        Route::get('/show', [StoryController::class, 'show']);
        Route::put('/update', [StoryController::class, 'update']);
        Route::delete('/destroy', [StoryController::class, 'destroy']);
        Route::get('/random', [StoryController::class, 'randomWords']);
    });

    Route::post('/logout', [AuthController::class, 'logout']);
    Route::get('/me', [AuthController::class, 'me']);
    Route::put('/user', [AuthController::class, 'update']);
});
```

4.5. Generación de palabra y lugar aleatorios

Agatha usa una única tabla random_words, con dos tipos:

word → palabras

place → lugares

```
$word = RandomWord::where('type', 'word')->inRandomOrder()->value('value');
$place = RandomWord::where('type', 'place')->inRandomOrder()->value('value');
```

4.6. Validación de historias

Se exige que la historia incluya la palabra y el lugar proporcionados:

```
if (
    stripos($request->content, $word) === false ||
    stripos($request->content, $place) === false
) {
    return response()->json([
        "success" => false,
        "message" => "La historia debe incluir la palabra '{$word}' y el lugar '{$place}'."
    ], 422);
}
```

4.7. Guardado de historias con token único

```php
$story = Story::create([
    'story_token' => Str::uuid()->toString(),
    'random_word' => $request->word,
    'random_place' => $request->place,
    'title' => $request->title,
    'content' => $request->content,
    'word_count' => $this->countWords($request->content),
    'user_id' => $request->user_id,
]);
```

4.8. Reset de inactividad al escribir

```
$status = $story->user->inactivity;

if ($status) {
    $status->update([
        'last_story_at' => now(),
        'first_email_sent_at' => null,
        'second_email_sent_at' => null,
    ]);
} else {
    $story->user->inactivity()->create([
        'last_story_at' => now(),
    ]);
}
```

4.9. Frontend de historias (Vue)

Obtener historias

```
export async function getStories() {
  const { data } = await api.get('/story/list')
  return data
}
```

Crear historia

```
export async function createStory(story) {
  return api.post('/story/store', {
    title: story.title,
    word: story.word,
    place: story.place,
    content: story.content,
    user_id: story.user_id,
  })
}
```

4.10. Formateo de fechas (frontend)

```
function formatDate(dateString) {
  return new Date(dateString).toLocaleDateString('es-ES', {
    year: 'numeric',
    month: 'numeric',
    day: 'numeric'
  })
}
```

4.11. Sistema de avisos automáticos por inactividad

El comando users:check-inactive recorre todos los usuarios y:

- Calcula días sin escribir.
- Envía primer aviso.
- Envía segundo aviso.
- Registra fechas en la tabla inactivities.

Código real del comando:

```php
$lastStory = $user->stories()->latest()->first();
$days = $lastStory->created_at->diffInDays(now());

$status = $user->inactivity ?: $user->inactivity()->create([
    'last_story_at' => $lastStory->created_at
]);
```

Primer aviso:

```php
if ($days >= 1 && $days < 5) {
    if (!$status->first_email_sent_at) {
        Mail::to($user->email)->send(new FirstInactiveUserMail($user));
        $status->update(['first_email_sent_at' => now()]);
    }
}
```

Segundo aviso:

```
if ($days >= 5) {
    if (!$status->second_email_sent_at) {
        Mail::to($user->email)->send(new SecondInactiveUserMail($user));
        $status->update(['second_email_sent_at' => now()]);
    }
}
```

4.12. Mailable real

```php
class FirstInactiveUserMail extends Mailable
{
    use Queueable, SerializesModels;

    public function __construct(public $user) {}

    public function content(): Content
    {
        return new Content(
            markdown: 'emails.inactive_first',
            with: ['user' => $this->user]
        );
    }
}
```

4.13. Plantilla Markdown del email

```
@component('mail::message')
# ¡Te echamos de menos, {{ $user->name }}!

Hace más de una semana que no escribes una historia.

@component('mail::button', ['url' => config('app.url').'/home'])
Volver a escribir
@endcomponent

Gracias,<br>
El equipo de Agatha
@endcomponent
```

4.14. Estructura de carpetas del backend (Laravel)

app/	database/
├── Actions/Fortify	├── factories/UserFactory.php
├── Console/Commands	├── migrations/
├── Http/Controllers/	│ ├── 0001_01_01_000000_create_users_table.php
│ ├── Controller.php	│ ├── 2025_10_10_095825_create_stories_table.php
│ ├── StoryController.php	│ ├── 2025_10_19_162629_add_two_factor_columns...
│ └── Api/	│ ├── 2025_11_12_132255_create_random_words_table.php
│ ├── Controller.php	│ └── 2025_11_26_184153_create_user_inactivity_status_table.php
├── Http/Resources	├── seeders/
├── Mail/	│ ├── DatabaseSeeder.php
│ ├── FirstInactiveUserMail.php	│ ├── RandomWordSeeder.php
│ ├── InactiveUserMail.php	│ └── StorySeeder.php
│ └── SecondInactiveUserMail.php	
├── Models/	resources/views/emails/
│ ├── RandomWord.php	├── inactive_first.blade.php
│ ├── Story.php	└── inactive_second.blade.php
│ ├── User.php	
│ └── UserInactivityStatus.php	routes/
├── Providers/	├── api.php
└── Traits/WordCountTrait.php	├── web.php
	└── console.php

storage/	tests/ y resto
├── app/private + public	├── tests/Feature/
├── framework/	├── tests/Unit/
│ ├── cache/data	├── public/
│ ├── sessions	├── docs/ (esta documentación)

4.16. Estructura de carpetas del frontend (Vue)

Raíz del proyecto	src/ (código principal)
├── node_modules/	├── App.vue
├── public/	├── main.js
├── .gitignore	├── assets/style.css
├── index.html	├── components/Sidebar.vue
├── jsconfig.json	├── router/index.js
├── package.json	├── services/api.js
├── package-lock.json	├── stores/
├── postcss.config.js	│ ├── auth.js
├── tailwind.config.js	│ └── stories.js
├── vite.config.js	├── views/
└── README.md	│ ├── DashboardView.vue
	│ ├── LoginView.vue
	│ ├── MainLayout.vue
	│ ├── MainView.vue
	│ ├── RegisterView.vue
	│ ├── StoriesView.vue
	│ └── StoryDetailView.vue