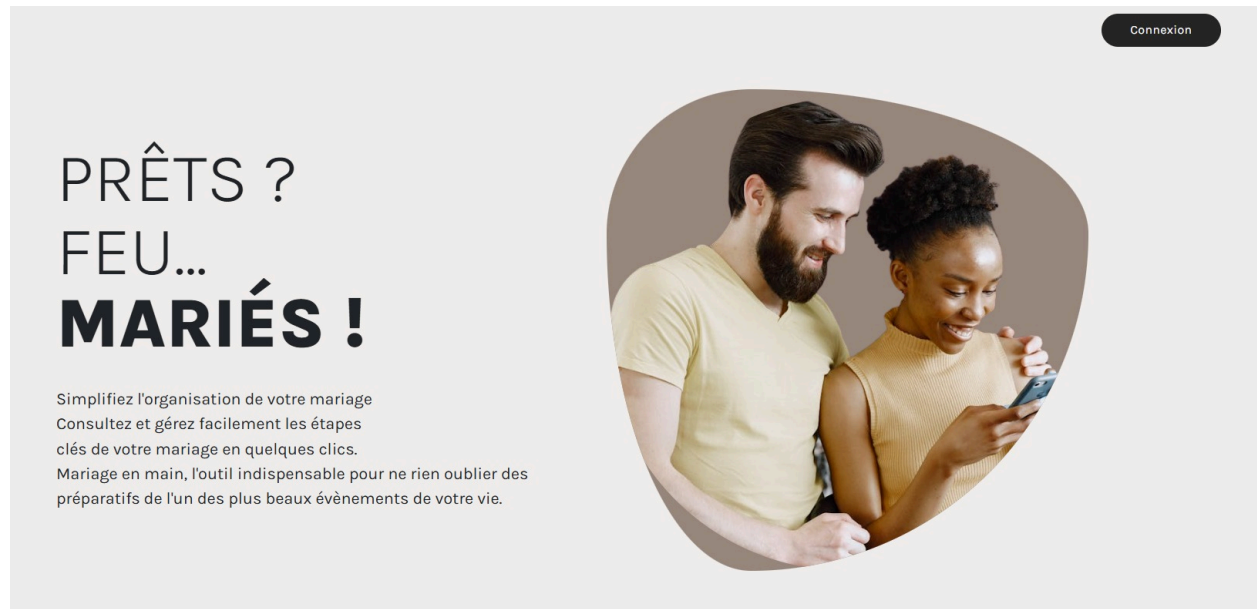


Plan de projet de test web – Desktop/Chrome

MARIAGE EN MAIN

Outil de gestion de projet mariage



Objectifs du projet

Créer une suite de tests exhaustive pour valider les fonctionnalités clés du site *Mariage en main*, en combinant des tests manuels et automatisés. Ce projet servira de démonstration concrète de mes compétences techniques et de ma capacité à organiser et exécuter un projet de test de A à Z.

MPACKO Diane

Objectifs spécifiques.....	4
Résultats attendus.....	5
Critères de succès.....	5
Livrables.....	5
Gestion de projet.....	7
Jira.....	7
Stratégie de tests.....	9
Objectifs de la stratégie de test.....	9
Types de tests à mettre en place.....	9
1. Tests d'API (Backend).....	9
2. Tests d'interface utilisateur (Frontend).....	9
3. Tests d'intégration.....	10
4. Tests de sécurité.....	10
Gestion de tests.....	11
AIO Tests.....	11
Jira.....	12
Périmètre et ensembles de test.....	12
Rapport.....	13
Automatisation.....	15
Cypress.....	15
Postman.....	16
Scripts.....	16
Cas de tests.....	20
Runner.....	22
Export de résultats.....	24
Newman.....	25
CLI et Automatisation.....	25
Rapport.....	27
AIO Tests.....	27
Intégration des tests d'API Postman.....	27
Exécution et résultats.....	29
Intégration des tests d'interface Cypress.....	30
API AIO Tests.....	33
Remontée d'anomalies.....	34
Rapports et métriques.....	37
Statistiques globales de la campagne.....	38

Répartition des anomalies par priorité.....	40
Couverture d'automatisation.....	41

Objectifs spécifiques

1. Créer une stratégie de test

2. Démontrer la maîtrise des outils de test :

- **AIO Tests** pour gérer l'exécution des tests et les rapports de cycles (campagnes)
- **Newman CLI** pour générer des rapports de cas de tests
- **Jira** pour planifier et suivre les tâches du projet
- **Postman** pour tester l'API MEM
- **Cypress** pour créer des scripts de tests

3. Utiliser mes compétences en développement :

- Écrire des scripts de test robustes, maintenables et bien structurés

4. Créer un projet reproductible et documenté :

- Structurer le projet pour qu'il puisse être facilement cloné et exécuté par d'autres personnes
- Documenter chaque étape du projet (installation, exécution des tests, interprétation des résultats)

5. Identifier et rapporter des bugs :

- Détecter les éventuels dysfonctionnements du site
- Documenter les bugs de manière claire et précise

Résultats attendus

- Une suite de tests complète couvrant les fonctionnalités principales du site.
- Des rapports de test détaillés (succès, échecs, temps d'exécution).
- Une documentation technique claire et accessible.
- Un dépôt GitHub bien organisé avec le code source des tests, les données de test et les instructions pour exécuter le projet.

Critères de succès

- **Couverture des tests** : Au moins 80 % des fonctionnalités clés du site sont testées.
- **Qualité des tests** : Les scripts de test sont robustes, maintenables et bien documentés.
- **Rapports de test** : Les rapports générés sont clairs et permettent d'identifier rapidement les problèmes.
- **Documentation** : Le projet est facile à reproduire et à comprendre pour un tiers.
- **Bugs rapportés** : Les éventuels bugs sont bien documentés.

Livrables

1. **Plan de test** : Document détaillant les fonctionnalités à tester, les types de tests (manuels/automatisés) et les outils utilisés.
2. **Scripts de test** :
 - Scripts Cypress pour les tests UI.
 - Collections Postman pour les tests API.

-
3. **Rapports de test** : Générés via Newman, Cypress et AIO Tests.
 4. **Documentation** :
 - README pour expliquer le projet.
 - Documentation technique pour expliquer la structure du projet.
 5. **Base de connaissances** : Liste des bugs trouvés avec des étapes pour les reproduire.




Gestion de projet

Jira

Jira est un outil de gestion de projet et de suivi des problèmes (issue tracking). Il est utilisé par les équipes de développement logiciel pour planifier, suivre et gérer les tâches, les bugs, les fonctionnalités et les projets.

Pour le projet QASM, il n'est pas nécessaire de créer un SCRUM. Le tableau Kanban est divisé en 3 colonnes: TO DO (à faire), WIP (Work in progress - En cours) et DONE (Terminé).

Tableau QASM



Étiquette ▼

Type ▼

REGROUPER PAR

Au

TO DO 6

En tant qu'utilisateur, je peux retourner sur la page d'accueil ou vers le tableau de bord, en cliquant sur le logo.
test-manuel
QASM-13

En tant qu'utilisateur, je peux me connecter à mon compte. ...
test-manuel
QASM-18

Créer un script Cypress pour tester le formulaire de connexion.
postman test-automatise api

WIP 4

En tant qu'utilisateur, je peux renseigner mon adresse mail dans le champ "Créer un compte" et le retrouver dans le formulaire complet de création de compte.
test-manuel
QASM-23

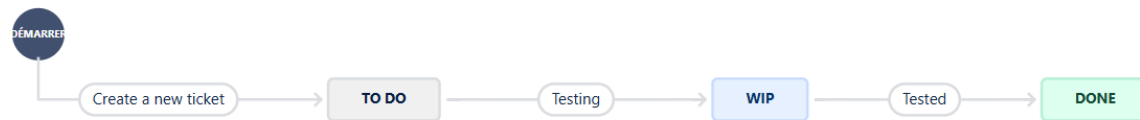
Vérifier que l'adresse mail renseignée dans le champ "Créer un compte" est retranscrite dans le formulaire complet de création de compte
test-automatise cypress
QASM-24

DONE 1 ✓

Tester le formulaire de Création de compte
postman test-automatise api
QASM-28 ✓

Q Afficher tous les tickets termi...

Afin de travailler de manière organisée, on crée un workflow pour la gestion des tickets.



Plusieurs EPICs sont créés pour découper et assurer un cadre pour les tickets.

- Tests de la page d'accueil
- Tests du tableau de bord
- Tests de fonctionnalités du tableau de bord
- Test de la navigation

Les trois autres types de tickets représentent:

- Les **Stories**, pour la description des user stories
- Les **Tâches**, essentiellement pour les tâches automatisées (scripts)
- Les **Bugs**, pour la remontée d'anomalies

Stratégie de tests

Intro: Comment j'ai déterminé les premiers tests à faire; Priorités (connexion, sécurité: authorization/token, etc...)

Objectifs de la stratégie de test

- Valider le bon fonctionnement de l'API (backend) et de l'interface utilisateur (frontend).
- Détecter les régressions en cas de modifications.
- Automatiser les tests pour gagner du temps et améliorer la fiabilité.
- Intégrer les tests dans le pipeline CI/CD pour une validation continue.
- Documenter et partager les résultats avec l'équipe via AIO Tests et JIRA.

Types de tests à mettre en place

1. Tests d'API (Backend)

- Outils : Postman, Newman.
- Objectif : Valider le comportement de l'API (ex: *création de compte, connexion, etc.*).
- Cas de test :
 - Vérifier les codes de statut HTTP (200, 201, 400, 500, etc.).
 - Valider les réponses JSON (structure et données).
 - Tester les cas d'erreur (ex: *données manquantes, formats invalides, etc.*).

2. Tests d'interface utilisateur (Frontend)

- Outils : Cypress.
- Objectif : Valider le comportement de l'interface utilisateur.

- **Cas de test :**

- Vérifier que les formulaires fonctionnent correctement (ex: *création de compte, connexion, gestion individuelle des fonctionnalités du tableau de bord, etc.*).
- Tester la navigation entre les pages.
- Valider les messages d'erreur affichés à l'utilisateur.
- Tester la compatibilité avec différents navigateurs et appareils.

3. Tests d'intégration

- Objectif : Valider l'interaction entre le frontend et le backend.
- Cas de test :
 - Vérifier que les données saisies dans le frontend sont correctement envoyées au backend.
 - Valider que les réponses de l'API sont correctement affichées dans l'interface utilisateur.

4. Tests de sécurité

- Objectif : Valider que l'application est sécurisée.
- Cas de test :
 - Vérifier que les endpoints sensibles (*Authentication*) sont protégés.
 - Vérifier que les accès aux contenus (*Authorization*) sont protégés.

Gestion de tests

AIO Tests

AIO Tests est un outil de gestion des tests (Test Management) conçu pour aider les équipes à planifier, exécuter, suivre et rapporter les tests logiciels. Il est souvent utilisé de pair avec des outils comme Jira pour intégrer la gestion des tests dans le cycle de développement logiciel.

C'est un outil flexible qui, tout comme Jira, permet l'intégration d'autres outils, d'automatisation et de reporting. Ce qui favorise la centralisation de tous les tests et une meilleure gestion du projet via:

- La création de cas de test, en définissant des scénarios de test pour valider les fonctionnalités.
- L'exécution des tests, en enregistrant les résultats des tests manuels ou automatisés.
- Le suivi des défauts, en reliant les échecs de test aux bugs dans Jira.
- Les rapports de test, en analysant la qualité du logiciel grâce à des métriques et des tableaux de bord.

Script Javascript

Création de compte utilisateur

```
1 // 1. BASIC RESPONSE VALIDATION
2 pm.test("Response structure is valid", function() {
3     const response = pm.response.json();
4     pm.expect(response).to.have.property("success").that.is.a("boolean");
5     pm.expect(response).to.have.property("message").that.is.a("string");
6 });
7
8 // 2. SCENARIO-BASED TESTS
9 const response = pm.response.json();
10 const requestBody = JSON.parse(pm.request.body.raw || '{}');
11
12 switch(pm.response.code) {
13     case 200:
14         pm.test("[200] Should register successfully with valid data", function() {
15             pm.expect(response).to.deep.equal({
16                 success: true,
17                 message: "Votre compte a été créé, vous pouvez dès à présent vous connecter"
18             });
19
20             // Verify password meets schema requirements
21             const passwordRegex = /^(?=.*[0-9])(?=.*[!@#$%^&*()\-_+={};:,<.>])(?=.*\d)((?=.*[a-z])(?=.*[A-Z])(?!.*[A-Z]).*)$/;
22             pm.expect(requestBody.password).to.match(passwordRegex);
23
24             // Verify email format
25             pm.expect(requestBody.email).to.match(/^[\s@]+\.[\s@]+\.[\s@]+$/);
26         });
27         break;
28
29     case 422:
30         if (response.message === "Tous les champs doivent être remplis") {
31             pm.test("[422] Should reject missing fields", function() {
32                 const requiredFields = ['firstPerson', 'secondPerson', 'email', 'password'];
33                 const missingFields = requiredFields.filter(field => !requestBody[field]);
34
35                 pm.test("[422] Missing fields: ${missingFields.join(', ')}", () => {
36                     pm.expect(missingFields.length).to.be.above(0);
37                 });
38             });
39         } else {
40             pm.test("[422] Should reject invalid password format", function() {
41                 pm.expect(response.message).to.include("Le mot de passe doit contenir");
42
43                 // Verify password fails schema requirements
44                 if (requestBody.password) {
45                     const passwordRegex = /^(?=.*[0-9])(?=.*[!@#$%^&*()\-_+={};:,<.>])(?=.*\d)((?=.*[a-z])(?=.*[A-Z])(?!.*[A-Z]).*)$/;
46                     pm.expect(requestBody.password).to.not.match(passwordRegex);
47                 }
48             });
49         }
50         break;
51 }
```

(QASM-TC-11: Verify the register form validation) EXEMPLE

Jira

Périmètre et ensembles de test

Le projet couvre l'ensemble du site Mariage en main. Il est divisé en EPICs couvrant différentes zones/fonctionnalités, traitées par priorité.

Tous les tickets

Partager ▾

Exporter ▾

Accéder à tous les tickets

VUE LISTE ≡

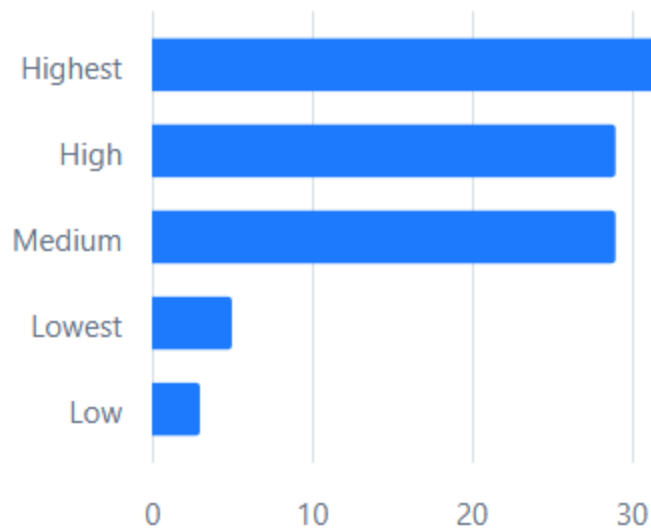
VUE DÉTAILLÉE ≡

...

🔍 AI	project = "QASM" ORDER BY priority DESC					↻ ? 🔍	Réinitialiser	Enregistrer le filtre	DE BASE	JQL
<input type="checkbox"/>	>	Type	Résumé	Priorité ▾	État					
<input type="checkbox"/>	>	🔗	Dashboard Main Features	🔥 Highest	WIP ▾					...
<input type="checkbox"/>	>	🔗	Settings	🔥 High	WIP ▾					...
<input type="checkbox"/>	>	🔗	Dashboard	🔥 High	WIP ▾					...
<input type="checkbox"/>	>	🔗	Side Navigation	🔥 High	WIP ▾					...
<input type="checkbox"/>	>	🔗	Homepage	🔥 High	WIP ▾					...
<input type="checkbox"/>	>	🔗	Animations	👉 Low	WIP ▾					...
+ Créer										6 sur 6 ↻

Rapport

La répartition des tickets dans le projet reflète une stratégie de priorisation alignée sur l'importance fonctionnelle et l'impact utilisateur.



- Priorités hautes (highest/high) pour les fonctionnalités critiques et structurantes – 55%

Ces éléments constituent le socle indispensable de l'application. Ils garantissent :

- ❖ La sécurité (ex : *authentification*)
 - ❖ L'accès aux fonctionnalités métier clés (ex : *gestion des invités, des tables*)
 - ❖ Une expérience utilisateur fluide (ex : *navigation, tableau de bord*).
- Priorité moyenne (medium) pour les fonctionnalités secondaires utiles – 30%

Ces fonctionnalités sont utilisées régulièrement mais ne bloquent pas l'application. Elles peuvent être remplacées temporairement (ex : *Excel pour les dépenses*). Également pour des éléments techniques, comme le fonctionnement des redirections ou des coquilles dans les URLs, qui n'impactent pas directement l'UX ou l'accès à la redirection est assurée par un autre moyen (ex: *plusieurs façons d'accéder aux fonctionnalités métier, par la navigation latérale ou pas le tableau de bord*).

- Priorité basse (low/lowest) pour les améliorations UI – 15%

Ces éléments représentent un impact négligeable sur la fonctionnalité, améliore le confort mais n'est pas essentiel (ex: *style des boutons, mettre en avant un onglet actif*).

Automatisation

Cypress

Cypress est un outil de test automatisé pour applications web, qui permet d'écrire et exécuter des tests end-to-end (E2E), d'intégration et unitaires via un navigateur. Dans ce projet, il aide à :

- Automatiser les tests critiques (authentification, redirections, gestion des invités/tables).
- Valider les interactions utilisateur (clics sur les boutons), soumissions de formulaires.
- Générer des rapports (via le plugin *cypress-jira-reporter*).

Script Javascript

Connexion utilisateur

```
39
40   it('should allow user to login with valid credentials (QASM-TC-82)', () => {
41
42     cy.openLoginForm();
43     cy.url().should('eq', `${baseUrl}/login`);
44
45     // The inputs are already prefilled with the valid user credentials
46     cy.loginViaUI();
47
48     // Check redirection to the dashboard page
49     cy.url().should("include", "/tableau-de-bord");
50   });
51
52   it('should not allow user to login with invalid credentials (QASM-TC-79)', () => {
53     cy.openLoginForm();
54     cy.url().should('eq', `${baseUrl}/login`);
55
56     cy.loginViaUI("unregistered@mail.com", "Azertyuiop123!");
57
58     cy.contains("Echec connexion").should("be.visible");
59     cy.url().should("eq", `${baseUrl}/login`);
60   });
61
```

Rapport d'exécution

Création de compte utilisateur
Connexion utilisateur
Page d'accueil

✓	auth/login.cy.ts	01:11	4	4	-	-	-
×	auth/register.cy.ts	00:53	2	1	1	-	-
✓	home/home.cy.ts	00:37	1	1	-	-	-
×	1 of 3 failed (33%)	02:42	7	6	1	-	-

Postman

Postman est un outil permettant de créer, tester, documenter et partager des API.

Au cours de la phase de développement, il m'a permis de tester la validité des requêtes de l'API MEM de l'application back end. Il est également pertinent de l'utiliser en phase de test notamment grâce à l'ajout de scripts et à son intégration à d'autres outils externes afin d'améliorer le workflow.

Scripts

L'API ayant déjà été testée au cours de la création du site, le projet se base sur cette collection déjà existante.

Pour rédiger les tests de scripts, on utilise la fonction correspondante à la fonctionnalité (endpoint) à tester.

Fonction Javascript

Création de compte utilisateur

```
57 exports.register = async (req, res) => {
58   try {
59     const { firstPerson, secondPerson, email, password } = req.body;
60
61     if (!firstPerson || !secondPerson || !email || !password) {
62       return res
63         .status(422)
64         .json({ success: false, message: "Tous les champs doivent être remplis" });
65     }
66
67     const existedUser = await Admin.findOne({ email: email });
68     if (existedUser) {
69       return res
70         .status(400)
71         .json({ success: false, message: "Un compte existe déjà avec cet email" });
72     }
73
74     let mariage = new Mariage({ ...req.body });
75
76     const newMariage = await mariage.save();
77
78     let hash = bcrypt.hashSync(req.body.password, 10);
79     let admin = new Admin({
80       ...req.body,
81       password: hash,
82       role: "admin",
83       mariageID: newMariage._id,
84     });
85
86     await admin.save();
87
88     res
89       .status(200)
90       .json({ success: true, message: "Votre compte a été créé, vous pouvez dès à présent vous connecter" });
91   } catch (err) {
92     res.status(500).json({ success: false, message: "Erreur serveur" });
93   }
94 };
95
```

C'est à partir de cette fonction que l'on peut déterminer de façon claire et exhaustive comment construire son test. Elle sert donc à rédiger le script de test Postman:

Script Javascript

Création de compte utilisateur

```
1 // 1. BASIC RESPONSE VALIDATION
2 pm.test("Response structure is valid", function() {
3     const response = pm.response.json();
4     pm.expect(response).to.have.property("success").that.is.a("boolean");
5     pm.expect(response).to.have.property("message").that.is.a("string");
6 });
7
8 // 2. SCENARIO-BASED TESTS
9 const response = pm.response.json();
10 const requestBody = JSON.parse(pm.request.body.raw || '{}');
11
12 switch(pm.response.code) {
13     case 200:
14         pm.test("[200] Should register successfully with valid data", function() {
15             pm.expect(response).to.deep.equal({
16                 success: true,
17                 message: "Votre compte a été créé, vous pouvez dès à présent vous connecter"
18             });
19
20             // Verify password meets schema requirements
21             const passwordRegex = /^(?=.*[0-9])(?=.*[!@#$%^&*()\-_+={};:,<.>])(?=.*\d)((?=.*[a-z])(?=.*[A-Z])(?=.*%)/;
22             pm.expect(requestBody.password).to.match(passwordRegex);
23
24             // Verify email format
25             pm.expect(requestBody.email).to.match(/^[^@]+@[^@]+\.[^@]+\$/);
26         });
27         break;
28
29     case 422:
30         if (response.message === "Tous les champs doivent être remplis") {
31             pm.test("[422] Should reject missing fields", function() {
32                 const requiredFields = ['firstPerson', 'secondPerson', 'email', 'password'];
33                 const missingFields = requiredFields.filter(field => !requestBody[field]);
34
35                 pm.test("[422] Missing fields: ${missingFields.join(', ')}", () => {
36                     pm.expect(missingFields.length).to.be.above(0);
37                 });
38             });
39         } else {
40             pm.test("[422] Should reject invalid password format", function() {
41                 pm.expect(response.message).to.include("Le mot de passe doit contenir");
42
43                 // Verify password fails schema requirements
44                 if (requestBody.password) {
45                     const passwordRegex = /^(?=.*[0-9])(?=.*[!@#$%^&*()\-_+={};:,<.>])(?=.*\d)((?=.*[a-z])(?=.*[A-Z])(?=.*%)/;
46                     pm.expect(requestBody.password).to.not.match(passwordRegex);
47                 }
48             });
49         }
50         break;
```

```

52 case 400:
53   pm.test("[400] Should reject existing email", function() {
54     pm.expect(response).to.deep.equal({
55       success: false,
56       message: "Un compte existe déjà avec cet email"
57     });
58
59     // Verify email format was correct
60     pm.expect(requestBody.email).to.match(/^[^\s@]+@[^\s@]+\.[^\s@]+$/);
61   });
62   break;
63
64 case 500:
65   pm.test("[500] Should handle internal errors", function() {
66     pm.expect(response).to.deep.equal({
67       success: false,
68       message: "Erreur serveur"
69     });
70   });
71   break;
72
73 default:
74   pm.test(`[UNEXPECTED] Status ${pm.response.code}`, function() {
75     pm.expect.fail("Unexpected response: ${JSON.stringify(response)}");
76   });
77 }
78
79 // 3. PASSWORD VALIDATION TESTS
80 if (pm.response.code === 422 && requestBody.password) {
81   const passwordTests = {
82     "min length": () => pm.expect(requestBody.password.length).to.be.below(6),
83     "uppercase": () => pm.expect(requestBody.password).to.not.match(/[A-Z]/),
84     "number": () => pm.expect(requestBody.password).to.not.match(/[0-9]/),
85     "special char": () => pm.expect(requestBody.password).to.not.match(/[\!@#%&*(){}^_+{};:,<.>]/)
86   };
87
88   Object.entries(passwordTests).forEach(([name, testFn]) => {
89     try {
90       testFn();
91       pm.test(`[422] Password should fail ${name} requirement`, () => true);
92     } catch (e) {
93       // If test doesn't fail, the password meets this requirement
94     }
95   });
96 }
97
98 // 4. QUALITY CHECKS
99 pm.test("Response time should be <1s", function() {
100   pm.expect(pm.response.responseTime).to.be.below(800);
101 });
102
103 pm.test("Should have correct content-type", function() {
104   pm.expect(pm.response.headers.get("Content-Type"))
105     .to.equal("application/json; charset=utf-8");
106 });

```

```

108 // 5. SECURITY CHECKS
109 pm.test("Password should not be exposed", function() {
110   pm.expect(JSON.stringify(response)).to.not.include(requestBody.password);
111 });
112
113 pm.test("Should use HTTPS", function() {
114   pm.expect(pm.request.url).to.match(/^https:/);
115 });

```

1. Vérifier la structure de la réponse serveur
2. Tester le jeu de données
3. Vérifier la conformité du mot de passe
4. Effectuer des tests de performance et qualité (optionnel)
5. Effectuer des tests de sécurité

Cas de tests

On crée un jeu de données pour chaque cas de test:

- Formulaire valide
- Formulaire incomplet
- Formulaire avec mot de passe invalide
- Formulaire contenant une adresse mail existante

Test case	Type	Input Data	Expected status code	Expected result
Valid registration	Success	firstPerson: "Jean" secondPerson: "Michelle" email: "durand@example.com" password: "Passw0rd!23!"	200	Successful message
Missing field(s)	Error	firstPerson: "Jean" secondPerson: "" email: "test@example.com" password: "Passw0rd!23!"	422	"Tous les champs doivent être remplis"
Password too short	Error	firstPerson: "Jean" secondPerson: "Michelle" email: "test@example.com" password: "Pass"	422	"Le mot de passe doit contenir au moins 6 caractères, une majuscule, un nombre et caractère spécial."
Password without uppercase	Error	firstPerson: "Jean" secondPerson: "Michelle" email: "test@example.com" password: "password!23!"	422	"Le mot de passe doit contenir au moins 6 caractères, une majuscule, un nombre et caractère spécial."
Password without number	Error	firstPerson: "Jean" secondPerson: "Michelle" email: "test@example.com" password: "SuperPassword!"	422	"Le mot de passe doit contenir au moins 6 caractères, une majuscule, un nombre et caractère spécial."
Password without special char	Error	firstPerson: "Jean" secondPerson: "Michelle" email: "test@example.com" password: "SuperPassword123"	422	"Le mot de passe doit contenir au moins 6 caractères, une majuscule, un nombre et caractère spécial."
Email already exists	Error	firstPerson: "Jean" secondPerson: "Michelle" email: "durand@example.com" password: "Passw0rd!23!"	400	"Un compte existe déjà avec cet email"
Server error	Error/Exception	(Simulate DB failure)	500	"Erreur serveur"

Une fois les données disponibles, on peut tester le point d'entrée avec Postman.

Résultats des tests

Création de compte utilisateur

The screenshot displays a REST client interface for a POST request to `{{apiURL}}/api/auth/createAccount`. The request body is a JSON object with the following data:

```
{  "firstPerson": "Jean",  "secondPerson": "Michelle",  "email": "duzard@example.com",  "password": "Passw@rd123!"}
```

The response status is `200 OK` with a time of `343 ms` and a size of `579 B`. The test results section shows five tests, all of which passed:

- PASS: Response structure is valid
- PASS: [200] Should register successfully with valid data
- PASS: Response time should be <1s
- PASS: Should have correct content-type
- PASS: Should use HTTPS

Tous les tests sont passés, la ressource a été créée:

- La structure de la réponse serveur est correcte
- La requête a été envoyée avec des données valides et renvoie un statut HTTP 200
- Le serveur renvoie la réponse en moins d'une seconde
- La requête a transmis le(s) bons *header(s)*
- Les communications sont sécurisées et transitent bien par le protocole HTTPS

Réponse de l'API

Création de compte utilisateur

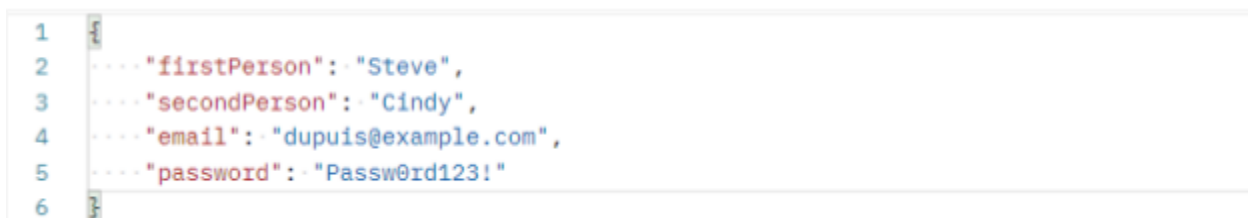


Runner

Il est possible d'organiser des suites de tests sur Postman avec l'outil Runner. On peut par exemple tester la connexion utilisateur après la création de compte.

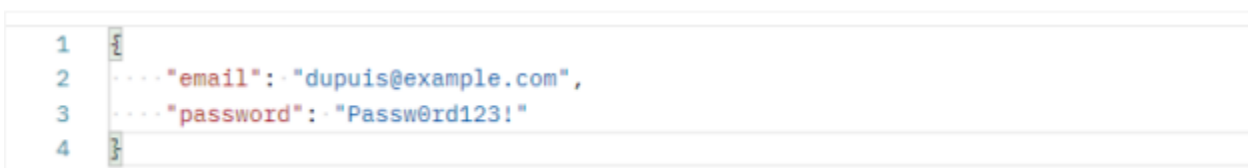
Jeu de données

Création de compte utilisateur



Jeu de données

Connexion utilisateur



Runner

Création de compte utilisateur - Connexion utilisateur

All Tests	Passed (8)	Failed (0)
Iteration 1		
1		
POST	MEM - POST Register HIGH QASM-TC-13	{{apiURL}}/api/auth/createAccount / Auth / MEM - POST Register HIGH QASM-TC-13 200 OK 384 ms 579 B
	Pass	Response structure is valid
	Pass	[200] Should register successfully with valid data
	Pass	Response time should be <1s
	Pass	Should have correct content-type
	Pass	Should use HTTPS
POST	MEM - POST Login HIGH QASM-TC-14	{{apiURL}}/api/auth/adminLogin / Auth / MEM - POST Login HIGH QASM-TC-14 200 OK 171 ms 849 B
	Pass	Response has correct structure
	Pass	[200] Status is 200
	Pass	[200] Valid credentials response

Les deux requêtes ont été exécutées avec succès.

Il est possible de tester la fiabilité des tests avec un nouveau jeu de données invalides.

Jeu de données invalide

Création de compte utilisateur avec un email déjà enregistré

```
1  {
2    "firstPerson": "Jean",
3    "secondPerson": "Michelle",
4    "email": "durand@example.com",
5    "password": "Passw@rd123!"
6  }
```

Jeu de données invalide

Connexion utilisateur avec un compte inexistant

```
1 {  
2   ... "email": "invalidmail@example.com",  
3   ... "password": "Passw0rd123!"  
4 }
```

Runner

Création de compte utilisateur invalide - Connexion utilisateur invalide

All Tests	Passed (8)	Failed (0)
Iteration 1		
POST	MEM - POST Register CRITICAL QASM-TC-13	{{apiURL}}/api/auth/createAccount / Auth / MEM - POST Register CRITICAL QAS... 400 Bad Request 232 ms 555 B
	Pass	Response structure is valid
	Pass	[400] Should reject existing email
	Pass	Response time should be <1s
	Pass	Should have correct content-type
	Pass	Should use HTTPS
POST	MEM - POST Login CRITICAL QASM-TC-14	{{apiURL}}/api/auth/adminLogin / Auth / MEM - POST Login CRITICAL QASM-TC-14 400 Bad Request 146 ms 569 B
	Pass	Response has correct structure
	Pass	[FAIL] Status is 400/404
	Pass	[400] Invalid credentials response

Tous les tests passent, car les cas d'erreurs sont gérés par le script (ici il s'agit d'erreurs 400 pour les deux requêtes).

Export de résultats

Une fois les tests réalisés, on exporte la collection Postman ainsi que ses variables d'environnement contenant des données de tests utiles et/ou indispensables telles que des tokens, des clés d'API etc, afin de générer un rapport Newman.

Newman

CLI et Automatisation

Newman est l'outil en ligne de commande de Postman. Il permet d'exécuter des collections Postman (ensembles de requêtes et de tests d'API) depuis un terminal ou un script, sans avoir à utiliser l'interface graphique de Postman. Il permet d'automatiser les tests d'API, les intégrer dans des pipelines CI/CD (Intégration Continue/Déploiement Continu) et à générer des rapports sur les résultats des tests.

L'exportation de la collection et des variables d'environnement depuis Postman génèrent deux fichiers JSON qui pourront être exécutés avec le CLI `newman`.

Runner

Création de compte utilisateur - Connexion utilisateur

```
Windows PowerShell
neuman
Api mariage Node
Auth
MEM - POST Register CRITICAL QASM-TC-13
POST https://my-wedding-backend.onrender.com/api/auth/createAccount [200 OK, 5798, 438ms]
✓ Response structure is valid
✓ [200] Should register successfully with valid data
✓ Response time should be <1s
✓ Should have correct content-type
✓ Should use HTTPS
MEM - POST Login CRITICAL QASM-TC-14
POST https://my-wedding-backend.onrender.com/api/auth/adminLogin [200 OK, 8508, 946ms]
✓ Response has correct structure
✓ [200] Status is 200
✓ [200] Valid credentials response
```

	executed	failed
iterations	1	0
requests	2	0
test-scripts	4	0
prerequisite-scripts	2	0
assertions	8	0
total run duration: 11.6s		
total data received: 471B (approx)		
average response time: 688ms [min: 438ms, max: 946ms, s.d.: 258ms]		

```
PS C:\Users\ >
```

- La suite de test a été exécutée une seule fois, soit 1 itération.
- La suite de test est composée de 2 requêtes: <Register> et <Login>.
- Les scripts sont composés de 8 assertions.
- Tous les tests ont été validés, aucun des tests n'a échoué.
- La suite de test a été exécutée en 11.6s (la commande newman a pris en compte le paramètre --delay-request 5000s de temps d'exécution)

pour permettre à la base de données de créer l'objet de la requête <Register>).

Rapport

Le runner de newman permet également de générer des rapports à partir de l'exécution des tests sous forme de fichier JSON, reprenant le détail de la collection Postman.

AIO Tests

Intégration des tests d'API Postman

Une fois les scripts de tests d'API créés avec Postman, ils sont transposables en cas de tests dans l'outil AIO. Le résultat du rapport Newman est importé dans un cycle AIO et génère automatiquement un cas de test avec Gherkin et son résultat/statut d'exécution.

Cas de test autogénéré

Création de compte utilisateur

✓

Aperçu

Cas

Ensembles

Cycles

Rapports

IA

Créer

←

QASM-TC-13: MEM - POST Register HIGH

Détails

Étapes

Associations

Historique

Titre *

MEM - POST Register HIGH

Description

POST {{apiURL}}/api/auth/createAccount

Préconditions

-

Dossier

-

Propriétaire *

DM

DM

Priorité

-

Statut *

Published

Type

-

Composants

-

Versions

-

Effort estimé

-

Étiquettes

-

Champs personnalisés ?

No custom fields defined

Automatisation ?

Statut

Automated

Propriétaire

DM

DM

Clé de l'automatisation ?

MEM - POST Register HIGH

Gherkin autogénéré
Création de compte utilisateur

✓

Aperçu

Cas

Ensembles

Cycles

Rapports >

Créer

QA Site Mariage

🔔

★

?

⚙️

<

QASM-TC-13: MEM - POST Register CRITICAL QASM-TC-13

>

🔗

⋮

Détails

Étapes

Associations

Historique

📄

Créer une nouvelle version

1

Classique

BDD/Gherkin

✎ Éditer

	Étape	Données
1	Given POST {{apiURL}}/api/auth/createAccount	<pre>{ "firstPerson": "Simon", "secondPerson": "Sophie", "email": "lemarchand@example.com", "password": "Passw0rd123!" }</pre>
2	Then Response structure is valid	
3	And [400] Should reject existing email	
4	And Response time should be <1s	
5	And Should have correct content-type	
6	And Should use HTTPS	

S'il s'agit de tests de base de données, effectués via Postman ou Newman, AIO autogénère un Gherkin en fonction du fichier JSON de reporting Newman.

Exécution et résultats

Il importe également les résultats d'exécution Postman dans le cycle AIO correspondant.

29

Résultat d'exécution

Création de compte utilisateur (avec email déjà enregistré)

QASM-TC-13 1		MEM - POST Register CRITICAL QASM-TC-13		Ajouter une exécution	
Exécuter 2		Automatisé		0h 0m 0s	
Tue, 06 May 2025					
Étape	Données	Résultats réels			
1 Given POST ([[apiURL]])/api/auth/createAccount	<pre>{ "firstPerson": "Simon", "secondPerson": "Sophie", "email": "semurhand@example.com", "password": "Password123!"}</pre>	<div>Variable Values for run : [apiURL : https://my-wedding-backend.onrender.com]</div> <div>Response (status="Bad Request", code=400, responseTime=340, responseSize=68)</div>			
2 Then Response structure is valid		<div>Ajouter les résultats réels</div>			
3 And [400] Should reject existing email		<div>Ajouter les résultats réels</div>			
4 And Response time should be <1s		<div>Ajouter les résultats réels</div>			
5 And Should have correct content-type		<div>Ajouter les résultats réels</div>			
6 And Should use HTTPS		<div>Ajouter les résultats réels</div>			

Chaque étape du test a été exécutée avec succès (indiqué par les checks de vérification ✓).

Intégration des tests d'interface Cypress

On utilise les clés de référence des cas de tests AIO existants (QASM-TC-**) pour le mapping entre Cypress et AIO Tests. Ce qui permet de synchroniser directement le résultat d'exécution des tests.

Script Cypress

Tests IHM/serveur e2e liés au formulaire de connexion

```
1 describe('login Form', () => {
2   const baseUrl = Cypress.config('baseUrl');
3
4   beforeEach(() => {
5     cy.visit(baseUrl || 'marriage-en-main.com');
6   });
7
8   it('should open the login form when the "Connexion" button is clicked (QASM-TC-80)', () => {
9     cy.openLoginForm();
10  });
11
12  it('should open the login form and match the URL "/login" (QASM-TC-6)', () => {
13    cy.openLoginForm();
14    cy.url().should('eq', `${baseUrl}/login`);
15  });
16
17  it('should close the login form', () => {
18    cy.openLoginForm();
19    cy.get('.auth-modal-close > button').click();
20    cy.contains('Se connecter').should('not.exist');
21  });
22
23  it('should match the base URL "/" when the form is closed (QASM-TC-9)', () => {
24    cy.openLoginForm();
25    cy.get('.auth-modal-close > button').click();
26    cy.url().should('eq', `${baseUrl}/`);
27  });
28
29  it('should switch from register to login form when the bottom link is clicked and match the URL "/login" (QASM-TC-8)', () => {
30    // BEWARE !! This is not a link but a button !
31    cy.openRegisterForm();
32    cy.url().should('eq', `${baseUrl}/register`);
33
34    const loginBtn = cy.get('.register_signup > button').contains('Connectez-vous');
35    loginBtn.should('be.visible').click();
36    cy.contains('Se connecter').should('be.visible');
37    cy.url().should('eq', `${baseUrl}/login`);
38  });
39
40  it('should allow user to login with valid credentials (QASM-TC-82)', () => {
41
42    cy.openLoginForm();
43    cy.url().should('eq', `${baseUrl}/login`);
44
45    // The inputs are already prefilled with the valid user credentials
```

Exécution du script/Mapping

Tests IHM/serveur e2e liés au formulaire de connexion

```
✓ TERMINAL

***** AIO Tests Reporter : Determining cycle to update *****
Reporting results to cycle : QASM-CY-Adhoc

Running: login.cy.ts (1 of 1)

Login Form
  ✓ should open the login form when the "Connexion" button is clicked (QASM-TC-80) (5711ms)
  ✓ should open the login form and match the URL "/login" (QASM-TC-6) (937ms)
  ✓ should close the login form (1619ms)
  ✓ should match the base URL "/" when the form is closed (QASM-TC-9) (854ms)
  ✓ should switch from register to login form when the bottom link is clicked and match the URL "/login" (QASM-TC-8) (1306ms)
  ✓ should allow user to login with valid credentials (QASM-TC-82) (4667ms)
  ✓ should not allow user to login with invalid credentials (QASM-TC-79) (2849ms)
  ✓ should redirect user to the homepage if unauthenticated (QASM-TC-78) (2247ms)
  ✓ should log out (QASM-TC-86) (4562ms)

9 passing (26s)

***** AIO Tests Reporter : Initiating reporting results for login.cy.ts *****
Number of case keys found 8
***** Updating passed cases *****
Successfully reported 8 passed cases.
***** AIO Tests Reporter : Reporting results completed for login.cy.ts *****

(Results)

Tests:      9
Passing:    9
Failing:    0
Pending:    0
Skipped:    0
Screenshots: 0
Video:      false
Duration:   25 seconds
Spec Ran:   login.cy.ts

=====

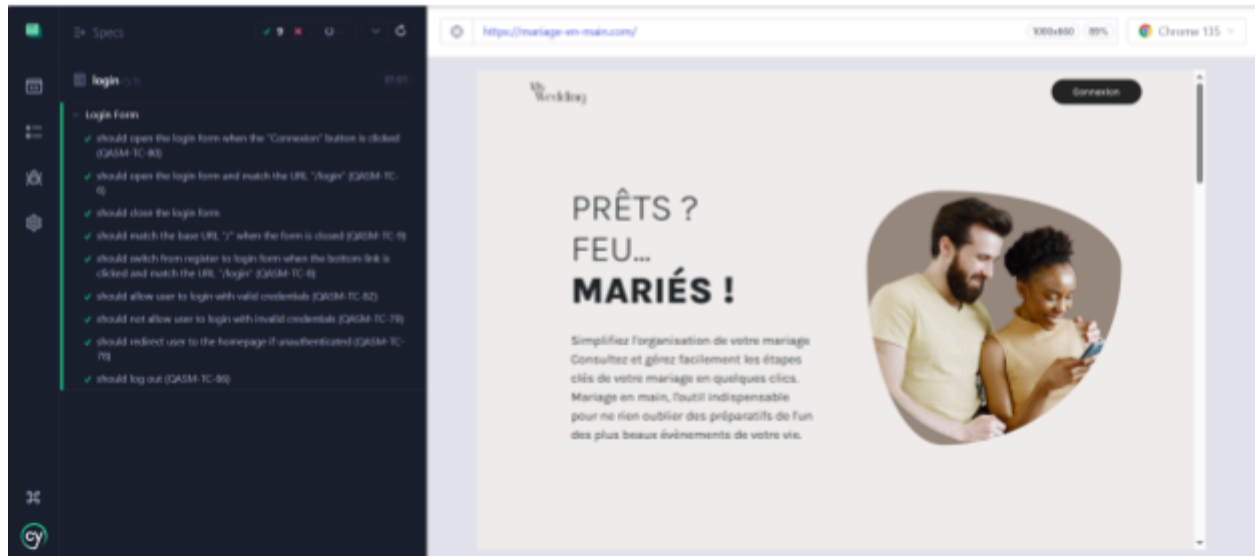
(Run Finished)

Spec                                Tests  Passing  Failing  Pending  Skipped
✓ login.cy.ts                       00:25    9        9        -        -        -
✓ All specs passed!                 00:25    9        9        -        -        -
```

Sur les 9 cas de tests passants (l'intégralité des tests liés à la connexion utilisateur), seul un test n'a pas été synchronisé avec AIO Tests, car il ne contient aucune référence de cas de tests existant.

Exécution via Cypress App

Tests IHM/serveur e2e liés au formulaire de connexion



L'application Cypress permet de visualiser l'exécution des tests end-to-end à travers une interface navigateur.

API AIO Tests










L'API AIO Tests permet de gérer le projet en passant par des requêtes serveur. Ainsi, il est possible de créer des cas de tests, des cycles, configurer le projet, etc.

Remontée d'anomalies

Lors de l'exécution des tests sur l'outil AIO, on déclare les anomalies. Elles peuvent être rattachées à des tickets JIRA existants sous forme de tickets de type "bug" préconfigurés et préremplis.

Remontée d'anomalies

Déclarations de "défauts" sur AIO Tests

Cas	Cycles	Défauts	
Affichage de 1-19 de 19 défauts			
Clé	Résumé	Priorité	Statut
 QASM-117	Échoué : Verify that the user can add a new guest to the list	Medium	À faire
 QASM-118	Échoué : Verify that the user can add a new table to the seating plan	Low	À faire
 QASM-119	Échoué : Verify that the user can add an element to the reception menu // HANDLE ERROR POPUP	Medium	À faire
 QASM-120	Échoué : Verify that the user can add a task to the list // +250 characters forbidden	High	À faire
 QASM-121	Échoué : Verify that the user can add a new expense to the list // +250 chards FORBIDDEN	High	À faire
 QASM-122	Échoué : Verify that the user can add a new expense to the list // PRICE +1M FORBIDDEN	High	À faire
 QASM-123	Échoué : Verify that the user can add a new expense to the list // PRICE SHOULDN'T BE LESS THAN 1	Medium	À faire
 QASM-124	Échoué : Verify that the user can update a guest // +50 chars FORBIDDEN	Medium	À faire
 QASM-125	Échoué : Verify that the user can update a table // Save button is disabled - tables hasn't been updated	Medium	À faire

Remontée d'anomalies

Tickets "bug" JIRA

- ❌ Échoué : Verify that the user can add a new guest to the list
QASM-117 - QA Site Mariage
- ❌ Échoué : Verify that the user can add a new table to the seating plan
QASM-118 - QA Site Mariage
- ❌ Échoué : Verify that the user can delete an element from the reception menu // successful message should be displayed
QASM-133 - QA Site Mariage
- ❌ Échoué : Verify that the user can change the first name // [INPUT RULE] Name should be at least 2 characters long - Error message should be red
QASM-134 - QA Site Mariage
- ❌ Échoué : Verify that the user can update an expense // DATE ISSUE (NOT VALID)
QASM-132 - QA Site Mariage
- ❌ Échoué : Verify that the user can update an expense // mode button should be disabled after clicking the cancel button
QASM-131 - QA Site Mariage
- ❌ Échoué : Verify that the user can update an expense // Expense error after clicking the cancel button
QASM-130 - QA Site Mariage
- ❌ Échoué : Verify that the user can update an expense // disable save button (+ CSS) + display error message(s)
QASM-129 - QA Site Mariage
- ❌ Échoué : Verify that the user can update a task / add success popup
QASM-128 - QA Site Mariage
- ❌ Échoué : Verify that the user can update a task // Change disabled button CSS + display error message
QASM-127 - QA Site Mariage
- ❌ Échoué : Verify that the user can update reception menu's element // Change save button CSS + add error message
QASM-126 - QA Site Mariage
- ❌ Échoué : Verify that the user can update a table // Save button is disabled - tables hasn't been updated
QASM-125 - QA Site Mariage

Remontée d'anomalies

Exemple de ticket "bug" JIRA

QASM-32 / QASM-120

Échoué : Verify that the user can add a task to the list // +250 characters forbidden

Ajouter Apps

Description

Jeu de données 2

content	QA engineers must thoroughly test input field limits. This 260-character string verifies that the system correctly enforces a 250-character limit. The remaining 10 characters should be truncated or rejected. Proper edge case testing ensures robust validation. 1234567890
result	An error message is displayed. The input can't contain more than 250 characters.
checkElement	
element	

Étapes

À faire Améliorer le ticket

Détails

Personne assignée D M

Étiquettes Aucun

Parent QASM-32 Dashboard Main Fe

Date d'échéance Aucun

Team Aucun

Développement Créer une branche Créer un commit

Rapporteur D M

Automatisation Exécutions de règles

QASM-32 / QASM-120

Étapes

N° de série	Étape
1.	Click on [+ Ajouter] button to display the add form
2.	Type QA engineers must thoroughly test input field limits. This 260-character string verifies that the system correctly enforces a 250-character limit. The remaining 10 characters should be truncated or rejected. Proper edge case testing ensures robust validation. 1234567890 in the name input
3.	FAILED: Click on [Valider] button

Détails de l'exécution du test

Pièces jointes de l'exécution:

Tickets associés

relates to

QASM-39 As a user, I can add to task to the t... TERMINÉ

À faire Améliorer le ticket

Détails

Personne assignée D M

Étiquettes Aucun

Parent QASM-32 Dashboard Main Fe

Date d'échéance Aucun

Team Aucun

Développement Créer une branche Créer un commit

Rapporteur D M

Automatisation Exécutions de règles

Le bug JIRA est prérempli avec les informations relatives à l'exécution du test: le cas de test concerné, le jeu de donné et l'étape ayant échoué.

Rapports et métriques

AIO Tests met à disposition un tableau de bord contenant de nombreux rapports détaillés afin d'analyser les résultats de la campagne de tests.

Les principales métriques principales analysées sont les suivantes:

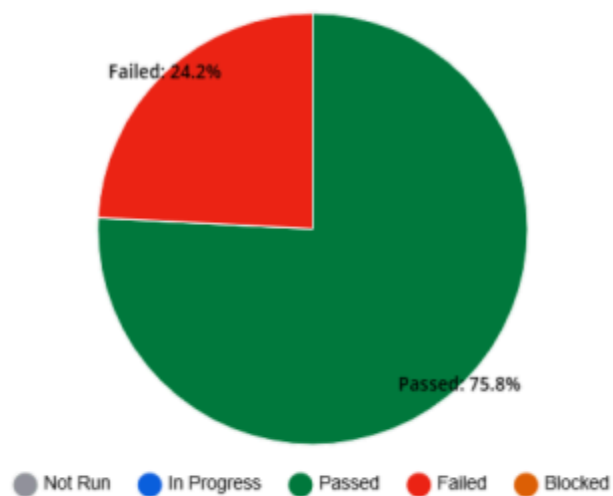
- **Taux de réussite/échec :**
 - Pourcentage de tests *Passed* vs *Failed/Blocked*
 - Indicateur clé de la stabilité des fonctionnalités testées
- **Couverture de test :**
 - Nombre de cas de tests exécutés vs. total planifié
 - Cartographie des exigences couvertes (traceability)
- **Défauts identifiés :**
 - Nombre de bugs remontés, classés par criticité (Blocker, Critical, Major, etc.)
 - Liens directs entre les cas en échec et les tickets Jira
- **Statut par composant :**
 - Répartition des résultats par module/fonctionnalité (ex : "*Authentication*" = 85% *Passed*)

Statistiques globales de la campagne

Cycle Statistics



Execution Distribution



- 91 cas de test au total, dont 0 cas incomplets
- 69 cas passés (75.8% de succès), avec 22 échecs (24.2%)
- Effort : 1h 04m 43s
- 19 anomalies
- 100% des tests ont été exécutés

L'analyse des rapports met en lumière un taux d'échec élevé, qui nécessite une attention particulière. Le niveau de criticité de ces tests doit être étudié pour prioriser le traitement des bugs et ainsi mettre en place des actions correctives.

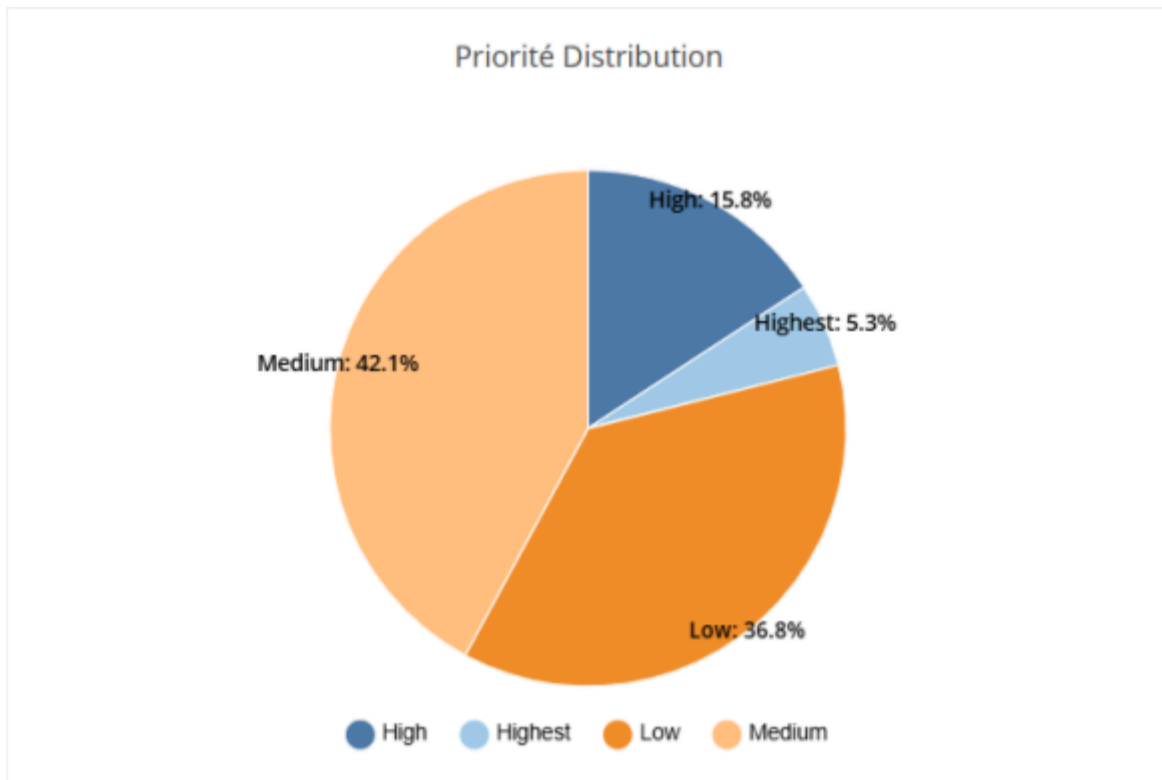
Rapport de test

Résumé d'exécution des tests manuels

Test	Case Count	Not Run	In Progress	Passed	Failed	Blocked	Defect Count
Manual	55 (0)	0 (0)	0 (0)	44 (0)	11 (0)	0 (0)	19
Authentication	4	0	0	3	1	0	0
Authorization	2	0	0	2	0	0	0
Main features	23	0	0	17	6	0	10
Redirections	7	0	0	7	0	0	0
Settings	3	0	0	0	3	0	1
UI/Display	7	0	0	7	0	0	0
URLs	5	0	0	4	1	0	0
UI/Style	4	0	0	4	0	0	0

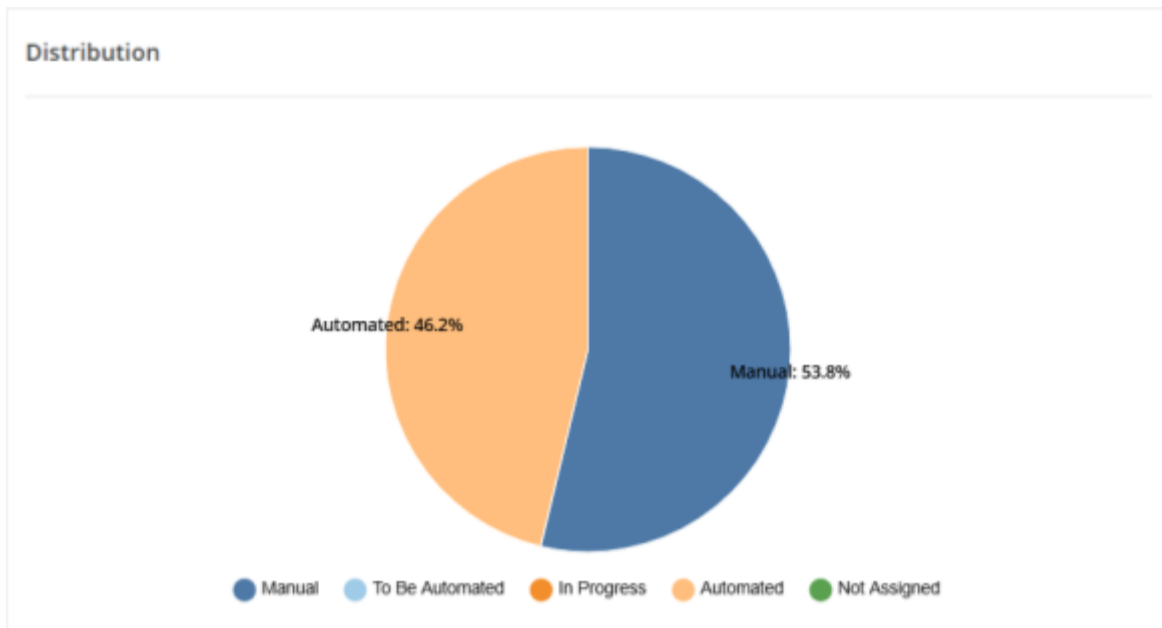
Sur 55 cas de tests manuels, 11 ont échoué (20%).

Répartition des anomalies par priorité



Les tickets à priorité haute et très haute représentent une faible proportion des bugs remontés (21.1%). Ils concernent des règles de gestion type longueur de caractères autorisés. Les tickets à priorité moyenne et basse représentent respectivement à peu près les mêmes proportions, avec une légère prédominance pour les tickets à priorité moyenne (42.1%). Sachant que ces derniers concernent des éléments d'UI et UX design, notamment des popups/messages de confirmation.

Couverture d'automatisation



Les tests automatisés représentent plus de la moitié de l'ensemble (53.8%). Ils représentent essentiellement des tests d'API avec Postman/Newman. Les tests automatisés Cypress sont peu nombreux dans ce cycle. De nouveaux tests seront créés dans les cycles à venir.

Les tests automatisés sont un gain de temps considérable surtout lorsque la campagne contient des centaines de tests à exécuter.