Customizing

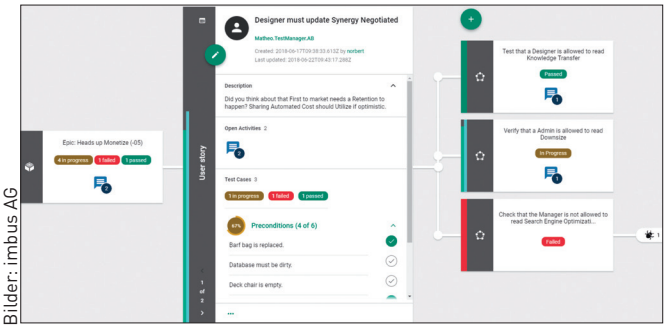
TestBench

Individuelles Testmanagement – Flexibel erweiterbar per API

TestBench Cloud Services (CS) ist eine leicht und schnell einsetzbare Cloud-basierte Lösung zur agilen Organisation von Software-Tests. Das Tool bietet eine bereichsübergreifende Arbeitsumgebung für agile Teams. Es sammelt und verwaltet sowohl Testaufgaben, Testfälle und Defects als auch Epics und User Stories des Teams. Mit TestBench CS sind alle Daten rund um ein Produkt stets auf dem aktuellsten Stand, übersichtlich miteinander verknüpft und jederzeit für alle User verfügbar.

Als Anwender arbeitet man mit TestBench CS via Browser, am PC oder an mobilen Endgeräten.

Zusätzlich bietet TestBench CS eine umfangreiche und komfortable REST-API. Diese API ermöglicht es, TestBench CS in eine vorhandene Tool-Chain zu integrieren, die Funktionalität individuell anzupassen oder zu ergänzen.



Bilder: imbus AG

Abb. 1: User Story mit Testfällen in der TestBench Cloud Services.

Die API ist Use Case- bzw. fachlich orientiert strukturiert und bildet die in der Bedieneroberfläche sichtbaren Elemente ab. Sie ist daher selbst für wenig geübte Programmierer schnell zu erlernen und einfach zu handhaben.

Die **Abb. 1** illustriert dies. Abgebildet ist eine User Story innerhalb eines Epics sowie die dieser User Story zugeordneten Testfälle.

Mit dem GET-Call **(Abb. 2)** fordert man die User Story aus dem Beispiel oben an. Um diesen Call aufzurufen benötigt man die IDs des Mandanten, des Produkts und der User Story. Das System kümmert sich darum, alle mit dieser User Story verknüpften Informationen zusammenzustellen. Es liefert die Basisdaten der User Story, das dazugehörende zu testende Produkt und das übergeordnete Epic zurück. Darüber hinaus erhält man die Listen der mit der User Story verknüpften Testfälle und der für die Testfälle vorhandenen Aktivitäten.

Die TestBench-API benötigt also zum Holen dieser Daten nur einen einzigen Call und liefert sie in einer JSON-Struktur zurück **(Abb. 3).**

Dies vereinfacht den Zugriff via API deutlich und führt zu kompaktem Programmcode. Für einen Programmierer ist es (z.B. in Python) so sehr einfach, eigene „Datenpumpen" für das Ein- und Auslesen von Daten in die TestBench CS zu implementieren.

Im Gegensatz dazu würde dieser Zugriff bei einer klassisch strukturierten API, deren Calls sich am internen Datenmodell der Anwendung orientieren, mehrere aufeinander folgende API-Calls je Entität erfordern: Zuerst das Holen der User Story; dann mit einem weiteren Call das Holen der Testfall-Liste dieser User Story; dann das Holen der Daten für das übergeordnete Epic; abschließend das Holen der Daten des Produktes.

Für das Erstellen einer User Story und das Befüllen mit Daten sind nur zwei Calls notwendig: Ein POST-Call **(Abb. 4)**, der eine neue User Story erstellt und initialisiert.

Und ein anschließender PATCH-Call **(Abb. 5)**, der die Daten wie gewünscht setzt.

Auch in der Dokumentation spiegelt die Reihenfolge der API-Blöcke deren übliche fachliche Nutzungsreihenfolge wieder:

- Dem grundlegenden Basic-Block, der die Authentifizierung betrifft, folgen die Blöcke, die für die Bedienung des Systems notwendig sind.
- Blöcke, die fachlich auf einer gleichwertigen Ebene stehen, haben bei Bedarf mehrere unterschiedliche Zweige.

Die API-Dokumentation hilft also, einem Entwickler die Anwendung der API so einfach wie möglich zu machen. Auch die AngularJS Single Page Application des TestBench Webclients nutzt diese API. Es ist also keine an das TestBench System „angeflanschte" API, sondern genau die API, die auch der Web-Client nutzt. Die API ist damit immer auf dem Versionsstand der Applikation und unterstützt alle Funktionen, die der Anwender von der Benutzeroberfläche kennt.


```
{
  "product": {
  "epic": {
  "id": 6,
  "name": "Designer must update Synergy Negotiated",
  "creation": {
  "description": "Did you think about that First to
  market needs a Retention to happen? Sharing Automated
  Cost should Utilize if optimistic. ",
  "specificationProgress": 61,
  "preconditionProgress": 67,
  "responsibleUsers": [
    {
      "id": 36,
      "login": "Matheo.TestManager.AB",
      "name": "Matheo.TestManager.AB",
      "status": "Active",
      "avatar": null
    }
  ],
  "lastUpdate": "2018-06-22T09:43:17.288Z",
  "preconditions": [
  "activityCountBySeverity": {
    "fire": 0,
    "flash": 0,
    "info": 2
  },
  "testCaseCount": 3,
  "testCaseCountByStatus": {
    "ready": 0,
    "inProgress": 1,
    "passed": 1,
    "failed": 1
  },
  "testCases": [
    {
      "id": 18,
      "name": "Test that a Designer is allowed to read
      Knowledge Transfer",
      "specificationProgress": 91,
      "preconditionProgress": 100,
      "activityCountBySeverity": {
        "fire": 0,
        "flash": 0,
        "info": 1
      },
      "status": "Passed",
      "defectCount": 0
    },
-snip-
    {
      "id": 16,
      "name": "Check that the Manager is not allowed to
      read Search Engine Optimization",
      "specificationProgress": 100,
      "preconditionProgress": 0,
      "activityCountBySeverity": {
        "fire": 0,
        "flash": 0,
        "info": 0
      },
      "status": "Failed",
      "defectCount": 1
    }
  ],
}
```


Abb. 3: Ergebnis-JSON des REST-Calls.

Die vollständige Dokumentation der API ist für alle registrierten Benutzer zu finden unter https://cloud01-eu.testbench.com/docs

Probieren Sie die TestBench Cloud Services und die TestBench-API einfach selbst aus. Holen Sie sich Ihr auf 90 Tage verlängertes Trial mit dem Gutscheincode **TBWMD072018** auf **90days.testbench.com**.

Abb. 2: GET-Call

GET /api/tenants/{tenantId}/products/{productId}/requirements/userStories/{userStoryId} Returns the information of the requested user story.

Abb. 4: POST-Call

POST /api/tenants/{tenantId}/products/{productId}/requirements/userStories Adds a user story.

Abb. 5: PATCH-Call

PATCH /api/tenants/{tenantId}/products/{productId}/requirements/userStories/{userStoryId} Updates the selected user story