HÖGSKOLAN
I SKÖVDE

Institutionen för Informationsteknologi

TENTAMEN

Kurs Utveckling av IT i Organisationer – Inriktning Systemvetenskap

Examinationsmoment Tentamen

Kurskod IT363G

Högskolepoäng för examinationsmomentet 6hp

Datum 30/01

Tentamenstid 8.15-12.30

Ansvarig lärare Hanife Rexhepi

Berörda lärare Johan Bjurén och Mikael Berndtsson

Hjälpmedel/bilagor

Övrigt

Anvisningar

- ☐ Ta nytt blad för varje lärare
- ☒ Ta nytt blad för varje ny fråga
- ☒ Skriv endast på en sida av papperet.
- ☒ Skriv namn och personnummer på samtliga inlämnade blad.
- ☒ Numrera lösbladen löpande.
- ☒ Använd inte röd penna.
- ☒ Markera med kryss på omslaget vilka uppgifter som är lösta.

Poänggränser

Tentamen består av tre delar. Respektive del består av tre frågor; två G-frågor och en VG-fråga. För att erhålla ett godkänt betyg på tentamen måste samtliga G-frågor inom respektive del vara godkända. Ett korrekt svar krävs för att frågan ska bedömas som godkänt.

För att erhålla betyg VG måste minst två VG-frågor vara väl godkänd samt övriga frågor godkända. Ett korrekt och utförligt svar krävs för att frågan ska bedömas som väl godkänt.

HÖGSKOLAN
I SKÖVDE

Skrivningsresultat bör offentliggöras inom 18 arbetsdagar

Lycka till!

Antal sidor totalt

Del 1: Implementering av Prototyp

Du ska i denna uppgift visa att du uppfyller dessa lärandemål (Ur kursplan):

- Använda utvecklingsfunktioner i ett ERP-system
- Redovisa och reflektera över olika problem som kan uppstå under ett systemutvecklingsprojekt och hur dessa kan hanteras av en projektgrupp

Frågor:

a) När vi jobbade med REST API:et så fick ni använda er av CRUD-operationer. För att använda dessa så fick man skicka förfrågan på olika sätt, tex GET för att hämta data, för de resterande 3 operationerna, förklara vilka operatorer som är kvar, vad operationen gör samt hur den ska skickas? **(G-fråga).**

b) Beskriv med text samt modell som visar flödet för process(erna) för hur ni gick till väga när ni implementerade funktionaliteten att boka en tid hos er vårdcentral. Börja med att en användare ska logga in, (dvs Inkludera processen för inloggning och vilka system som påverkas) **(G-fråga).**

a) I ert projekt har ni arbetat med REST API och gjort anrop till olika endpoints. För att göra detta har ni använt olika metoder för REST API:et, där vi tillsammans på handledningarna tittat på dokumentationen *(Se bilaga A längst bak på denna tenta)*.

Utifrån endpointen nedan hur skulle vi göra en förfrågan för att hämta alla som har ett patient_name som börjar på G9 och som har status Completed.

Vi vill bara få fram vem patienten är (patient_name) samt vem som behandlat patienten (paractitioner_name)

Endpoint:

/api/resource/Patient Encounter

Utgå från endpointen ovan, ange hur endpointen skulle se ut för att få fram rätt data enligt beskrivningen **(VG-fråga).**

Del 2: Kravspecifikation och Utvecklingsprojekt

Du ska i denna uppgift visa att du uppfyller dessa lärandemål (Ur kursplan):

- Realisera delar av en kravspecifikation inom ramen för ett befintligt ERP-system samt skapa ett webbgränssnitt gentemot en relationsdatabas

Frågor:

a) Förklara vad förkortningen ERP står för och beskriv hur ett sådant system kan bidra till *effektivare* verksamhetsstyrning. **(G-fråga)**

b) Utgå från de krav som har samlats in på Mölndals vårdcentral **(G-fråga).** Välj tre krav och förklara:
   I. hur de bör prioriteras utifrån verksamhetens behov och varför.

   II. på vilket sätt realiseringen av kraven stödjer verksamheten.

c) Identifiera tre centrala verksamhetsprocesser vid Mölndals vårdcentral. För varje process ska du:
   a. beskriva processens syfte och huvudsakliga aktiviteter
   b. analysera om processen lämpar sig bäst för stöd via ett ERP-system, via en specialiserad webblösning eller både ock.
   c. motivera ditt ställningstagande utifrån *användarbehov*, *processens komplexitet* och behov av *systemintegration.*

(VG-fråga)

Del 3: Centrala begrepp

Du ska i denna uppgift visa att du uppfyller dessa lärandemål (Ur kursplan):

- Förklara och konkretisera för projektet centrala begrepp relaterat till systemvetenskap
- Ansvara för specifikation av krav på ett informationssystem relaterat till grundläggande arkitektur och funktionalitet

Frågor:

a) Beskriv följande. **(G-fråga)**;
   I. Vad är ett krav?
   II. Vad är skillnaden mellan ett funktionellt och icke funktionellt krav.
   III. Ge exempel på två funktionella och två icke funktionella krav.

b) *Formulera två krav* (ett funktionellt och ett icke funktionellt krav) som adresserar de problem som Mölndals vårdcentralen upplever **(G-fråga).**

c) Redogör för kravhanteringsprocessens olika steg och förklara hur de har tillämpats i ert projekt. Analysera vilka konsekvenser era val i kravhanteringen har haft för *projektets resultat* med fokus på *funktionalitet* och *användbarhet*. Använd konkreta exempel. **(VG-nivå)**

BILAGA A – REST API

Framework

REST API

Edit

Frappe framework generates REST API for all of your DocTypes out of the box. You can also run arbitrary python methods using their dotted module path.

Authentication

There are two ways to authenticate through Frappe REST API. Token based authentication and password based authentication.

1. Token Based Authentication

A token is a pair of API Key and API Secret. To generate these tokens follow these steps:

1. Go to User list and open a user.
2. Click on the "Settings" tab. (skip this step if you don't see tabs)
3. Expand the API Access section and click on Generate Keys.
4. You will get a popup with the API Secret. Copy this value and keep it somewhere safe (Password Manager).
5. You will also see another field "API Key" in this section.

The token is generated by concatenating `api_key` and `api_secret` with a colon `:`. Pass the string `token api_key:api_secret` to the `Authorization` header in the request.

```
fetch('http://<base-url>/api/method/frappe.auth.get_logged_user', {
    headers: {
        'Authorization': 'token api_key:api_secret'
    }
})
.then(r => r.json())
.then(r => {
    console.log(r);
})
```

```
→ curl http://<base-url>/api/method/frappe.auth.get_logged_user -H "Authorization: token api_key:api_secret"
```

Every request you make with these tokens will be logged against the user you selected in Step 1. This also means that roles will be checked against this user. You can also create a new user just for API calls.

2. Password Based Authentication

Password based authentication relies on cookies and session data to maintain authentication in subsequent requests. In most cases, the library you are using to issue REST calls will handle session data, but if it doesn't you should use Token based authentication.

```
fetch('http://<base-url>/api/method/login', {
    method: 'POST',
    headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
    },
    body: JSON.stringify({
        usr: 'username or email',
        pwd: 'password'
    })
})
.then(r => r.json())
.then(r => {
    console.log(r);
})
```

```
→ curl --cookie-jar snowcookie --request POST "http://<base-url>/api/method/login" -H 'Content-Type: application/json' -H 'Accept: application/json' --data-raw
{"message":"Logged In","home_page":"/app","full_name":"<user:full_name>","dashboard_route":"/sites"}

→ curl --cookie snowcookie --request POST "http://<base-url>/api/method/frappe.auth.get_logged_user" -H 'Accept: application/json'
{"message":"<username>"}
```

3. Access Token

Refer documentation for How to setup OAuth.

Use the generated `access_token` in request header.

```
fetch('http://<base-url>/api/method/frappe.auth.get_logged_user', {
```

```
    headers: {
        'Authorization': 'Bearer access_token'
    }
})
.then(r => r.json())
.then(r => {
    console.log(r);
})
```

Listing Documents

To get a list of records of a DocType, send a GET request at `/api/resource/:doctype` . By default, it will return 20 records and will only fetch the name of the records. The result for the query can be found under `data` of the payload.

We'll be using the ToDo DocType to show example responses for the queries below.

```
GET /api/resource/:doctype
```

Response


```
{
  "data":[
    {"name":"f765eef382"},
    {"name":"2a26fa1c64"},
    {"name":"f32c68060f"},
    {"name":"9065fa9832"},
    {"name":"419082fc38"},
    {"name":"6234d15099"},
    {"name":"62f2181ee0"},
    {"name":"a50afbbfaa"},
    ...
  ]
}
```


You can specify which fields to fetch in the `fields` param. It should be a JSON array.

```
GET /api/resource/:doctype?fields=["field1", "field2"]
```

Response


```
{
  "data":[
    {"description":"Business worker talk society. Each try theory prove notice middle. Crime couple trouble guy project hit.","name":"f765eef382"},
    {"description":"This reveal as look near sister. Car staff bar specific address.","name":"2a26fa1c64"},
    {"description":"Wear bag some walk. Movie partner new class tough run. Brother Democrat imagine.","name":"f32c68060f"},
    {"description":"Break laugh apply reveal new now focus heavy. Outside local staff research total. Else point try despite.","name":"9065fa9832"},
    {"description":"Truth reduce baby artist actually model. Cost phone us others himself wife almost. Language thing wonder share talk. Factor glass significa
    {"description":"Tv memory understand opportunity window beat physical.","name":"6234d15099"},
    {"description":"Should floor situation in response sell. Our assume company mean red majority shoulder.","name":"62f2181ee0"},
    {"description":"Performance seem sign recent. Court form me tonight simple trouble. Address job garden play teach. Happy speech amount offer change then.",
    ...
  ]
}
```


You can specify which fields to expand in the `expand` param. It should be a JSON array.

```
GET /api/resource/:doctype?expand=["priority"]
```

Response


```
{
  "data":[
    {
        "name":"f765eef382"
        "priority": {
            "name":"a1b2c3",
            "title": "Medium",
            "creation": "2025-11-05 19:02:19.106966",
        },
    },
```

```
    {
        "name":"f765eef393"
        "priority": {
            "name":"a1b2c4",
            "title": "High",
            "creation": "2025-11-05 20:02:19.106966",
        },
    },
    ...
  ]
}
```

You can filter the records by passing `filters` param. Filters should be an array, where each filter is of the format: `[field, operator, value]`

```
GET /api/resource/:doctype?filters=[["field1", "=", "value1"], ["field2", ">", "value2"]]
```

Response

```
{
  "data":[
    {"name":"f765eef382"},
    {"name":"2a26fa1c64"},
    {"name":"f32c68060f"},
    {"name":"9065fa9832"},
    {"name":"419082fc38"},
    {"name":"6234d15099"},
    {"name":"62f2181ee0"},
    {"name":"a50afbbfaa"},
    ...
  ]
}
```

`filters` parameter joins all the specified filters using `AND` SQL operator, if you want `OR` filters you can use the `or_filters` param. Syntax for `or_filters` is same as `filters.
You can also provide the sort field and order. It should be of the format `fieldname asc` or `fieldname desc` . The space should be URL encoded. In the following line, we're taking fieldname to be `title` .

```
GET /api/resource/:doctype?order_by=title%20desc
```

You can also page the results by providing the `limit_start` and `limit_page_length` params.

```
GET /api/resource/:doctype?limit_start=5&amp;limit_page_length=10
```

Response

```
{
  "data": [
    {"name":"6234d15099"},
    {"name":"62f2181ee0"},
    {"name":"a50afbbfaa"},
    {"name":"aa12a5cf71"},
    {"name":"6ac9800d4e"},
    {"name":"4bcf8b701c"},
    {"name":"aee15f4c20"},
    {"name":"6ba753afef"},
    ...
  ]
}
```

`limit` is an alias for `limit_page_length` for accessing `/api/resource` in Version 13. This means the following should also return the same payload as the above query.

```
GET /api/resource/:doctype?limit_start=5&amp;limit=10
```

By default, you will receive the data as `List[dict]` . You can retrieve your data as `List[List]` by passing `as_dict=False` .

```
GET /api/resource/:doctype?limit_start=5&amp;limit=5&amp;as_dict=False
```

Response


```
{
  "data": [
    ["6234d15099"],
    ["62f2181ee0"],
    ["a50afbbfaa"],
    ["aa12a5cf71"],
    ["6ac9800d4e"]
  ]
}
```


To debug the query built for your reqeusts, you can pass `debug=True` with the request. This returns the executed query and execution time under `exc` of the payload.

```
GET /api/resource/:doctype?limit_start=10&amp;limit=5&amp;debug=True
```

Response


```
{
  "data": [
    {"name":"4bcf8b701c"},
    {"name":"aee15f4c20"},
    {"name":"6ba753afef"},
    {"name":"f4b7e24abc"},
    {"name":"bd9156096c"}
  ],
  "exc": "[\"select `tabToDo`.`name`\\n\\t\\t\\tfrom `tabToDo`\\n\\t\\t\\t\\n\\t\\t\\t\\n\\t\\t\\t order by `tabToDo`.`modified` DESC\\n\\t\\t\\tlimit 5 offset
}
```


CRUD Operations

Frappe generates REST endpoints for CRUD operations for all DocTypes automatically. Make sure you set the following headers in your requests so that you get proper JSON responses.

```
{
    "Accept": "application/json",
    "Content-Type": "application/json",
}
```

Create

Create a new document by sending a `POST` request to `/api/resource/:doctype`. Send the document as JSON in the Request Body.

```
POST /api/resource/:doctype

# Body
{"description": "New ToDo"}
```

Response


```
{
  "data": {
    "name": "af2e2d0e33",
    "owner": "Administrator",
    "creation": "2019-06-03 14:19:00.281026",
    "modified": "2019-06-03 14:19:00.281026",
    "modified_by": "Administrator",
    "idx": 0,
    "docstatus": 0,
    "status": "Open",
    "priority": "Medium",
    "description": "New ToDo",
    "doctype": "ToDo"
  }
}
```


Read

Get a document by sending a `GET` request to `/api/resource/:doctype/:name`.

```
GET /api/resource/:doctype/:name
```

Response


```
{
  "data": {
    "name": "bf2e760e13",
    "owner": "Administrator",
    "creation": "2019-06-03 14:19:00.281026",
    "modified": "2019-06-03 14:19:00.281026",
    "modified_by": "Administrator",
    "idx": 0,
    "docstatus": 0,
    "status": "Open",
    "priority": "Medium",
    "description": "<p>Test description</p>",
    "doctype": "ToDo"
  }
}
```


Expand all link fields by sending a `GET` request to `/api/resource/:doctype/:name?expand_links=True` .

```
GET /api/resource/:doctype/:name?expand_links=True
```

Response


```
{
  "data": {
    "name": "bf2e760e13",
    "owner": "Administrator",
    "creation": "2019-06-03 14:19:00.281026",
    "modified": "2019-06-03 14:19:00.281026",
    "modified_by": "Administrator",
    "idx": 0,
    "docstatus": 0,
    "status": "Open",
    "priority": {
        "name":"a1b2c3",
        "title": "Medium",
        "creation": "2025-11-05 19:02:19.106966",
    },
    "description": "<p>Test description</p>",
    "doctype": "ToDo"
  }
}
```


Update

Update a document by sending a `PUT` request to `/api/resource/:doctype/:name` . You don't need to send the whole document, instead you can just send the fields that you want to update.

```
PUT /api/resource/:doctype/:name

# Body
{"description": "New description"}
```

Response


```
{
  "data": {
    "name": "bf2e760e13",
    "owner": "Administrator",
    "creation": "2019-06-03 14:19:00.281026",
    "modified": "2019-06-03 14:21:00.785117",
    "modified_by": "Administrator",
    "idx": 0,
    "docstatus": 0,
    "status": "Open",
    "priority": "Medium",
    "description": "New description",
    "doctype": "ToDo"
  }
}
```

Delete

Delete a document by sending a `DELETE` request to `/api/resource/:doctype/:name`.

```
DELETE /api/resource/:doctype/:name
```

Response

```
{"message": "ok"}
```

Remote Method Calls

Frappe allows you to trigger arbitrary python methods using the REST API for handling custom logic. These methods must be marked as *whitelisted* to make them accessible via REST.

To run a whitelisted python method at `frappe.auth.get_logged_user`, send a request to the endpoint `/api/method/frappe.auth.get_logged_user`.

```
GET /api/method/frappe.auth.get_logged_user
```

Response

```
{
  "message": "john@doe.com"
}
```

- If your method returns some values, you should send a `GET` request.
- If your method changes the state of the database, use `POST`. After a successful `POST` request, the framework will automatically call `frappe.db.commit()` to commit the changes to the database.
- A successful response will return a JSON object with a `message` key.
- An errored response will return a JSON object with `exc` key which contains the stack trace and `exc_type` which contains the thrown Exception.
- The return value of the method will be converted to a JSON and sent as the response.

File Uploads

There is a dedicated method `/api/method/upload_file` that accepts binary file data and uploads it into the system.

Here is the curl command for it:

```
curl -X POST \
  http://<base-url>/api/method/upload_file \
  -H 'Accept: application/json' \
  -H 'Authorization: token xxxx:yyyy' \
  -F file=@/path/to/file/file.png
```

If you are using client side Javascript to upload files, you can append the uploaded files as FormData and send an XHR request. Here is the implementation code in Frappe Desk.

PREVIOUS PAGE
Hooks

NEXT PAGE
FullTextSearch API

Last updated 2 weeks ago

Was this helpful?