Opted for json objects to do the modelling. Seemed the most straight forward way. Another option would have been to use relational database schema and design but that seemed overkill.

Would like to point out that I'm not sure if I fully understand what is being asked so I will provide what I think are possible solutions. Be it similar. This is meant to be an easy to digest high overview of the solution. One major assumption to note is that name + surname or full name is a unique identifier. Other assumptions will be highlighted along the way.

Used this website as a guideline regarding json data types -> https://json-schema.org/understanding-json-schema/reference/type.html

Approach 1:

Question 1:

```
{
   "churchMember": {
      "name": "string",
      "surname": "string",
      "birthday": "string",
      "telephoneNumber": "string",
      "stillAlive": "boolean"
   }
}
```

Church Member is represented as a json object with required fields listed.

Question 2:

```
{
   "churchMember": {
      "name": "string",
      "surname": "string",
      "birthday": "string",
      "telephoneNumber": "string",
      "stillAlive": "boolean",
      "marriedTo": "string"
   }
}
```

Added a "marriedTo" field which would have the spouse's full name. If null or blank, then churchMember is currently unmarried. So, to indicate that Kate is married to Tshepo (and vice versa) their objects would look as follows:

```
{
  "church": [
    {
      "churchMember": {
        "name": "Tshepo",
        "surname": "Nyathi",
        ...
        "marriedTo": "Kate Nyathi"
```

```
    }
   },
   {
    "churchMember": {
     "name": "Kate",
     "surname": "Nyathi",
     ...
     "marriedTo": "Tshepo Nyathi"
    }
   }
 ]
}
```

Sudo Code would be something along the line of:
// say looking for couples
List of couples = Get all church members that marriedTo is not Null or Empty
Loop over list by looking for spouse of Member who is currently the head of the list then remove the two entries from the list
Add the pair in a map -> Map <Spouse1, Spouse2>
When list is empty, Map will have all the couples
// **assumption**: If one is married then that person and their spouse are alive.
// **assumption**: church members only marry within the church. So, length of list of couples
// will be an even number.

Question 3:

```
{
   "churchMember": {
      "name": "string",
      "surname": "string",
      "birthday": "string",
      "telephoneNumber": "string",
      "stillAlive": "boolean",
      "marriedTo": "string",
      "children": "array"
   }
}
```

Continuing with the single-object approach. Added children field to object of type array / list. This will have the list of children. If empty, then person is not a parent. Assumption here is that children are also church members. So, if Kate from above had 2 Children, then she would be represented as:

```
{
   "churchMember": {
      "name": "Kate",
      ...
      "marriedTo": "Tshepo Nyathi",
      "children": ["Garry Nyathi", "Lucy Nyathi"]
   }
```

}
If the children list matches that of Tshepo then we can assume that both children have the same pair of parents. And that they are related / siblings. Also, if Tshepo has 3 children, then we can identify which child is not from his current marriage to Kate. And seeing as everyone is a member of the church, we can identify the mother of Tshepo's 3rd child. This can be done by:
//sudo code
Get church object where childrenList contains "Tshepo's 3rd child".

Question 4
Number of parents -> count number of churchMembers where children > 0 (isNotEmpty)

Question 5
single parent(s) -> get # of parents
then -> filter where marriedTo is empty or null
then -> filter where age is less than 10. Age = get date of birth **greater than** (current date minus 10 years)

Approach 2:

Apologies if this doc is getting long now. This approach will introduce more objects for specific data, as opposed to representing everything in 1 object.

Question 1:
This remains the same as above.

```
{
   "churchMember": {
      "name": "string",
      "surname": "string",
      "birthday": "string",
      "telephoneNumber": "string",
      "stillAlive": "boolean"
   }
}
```

Question 2:
Introduce a new object which represents couples.

```
{
   "marriedChurchMember": {
      "fullname": "string",
      "Spouse": "string",
      "stillAlive": "boolean"
   }
}
```

In the unfortunate event of a death or divorce then the entry can simply be dropped. And also, the opposite applies, when a marriage happens, then an entry can be added.

Question 3:
Introduce yet another object

```
{
   "childrenParents": {
      "child": "string",
      "parentNames": "array"
   }
}
```

Parents Names array can be empty or hold several names. The assumption here will be looking at biological parents, thus the list will be a max length of 2.

childrenParents object draws the relations directly. And other information such as date of birth etc can be retrieved from churchMember object. This object is in most cases updated when a child is born.
In the event of a child or parent dying, should that record be kept???

Question 4:

Here one can count the unique names in the parents field. Possibly consider making use of a Set.

Question 5:

Get **churchMembers** under the age of 10 (who is alive??).

- ➔ Date of birth > Current date – 10 years

Then get parent names in the **parentChildren** for said children.

Finally, look for names that do not appear in **marriedChurchMember.**

Approach 3:

After further deliberations I decided to include a 3rd approach. This is a condensed item as you can probably see the line of thinking above. The traditional approach of dealing with data is to use id(s), GUID(s) or some other form of unique identifier.

```
{
   "churchMember": {
      "id": "string",
      "name": "string",
      "surname": "string",
      "birthday": "string",
      "telephoneNumber": "string",
      "stillAlive": "boolean",
      "marriedTo": "string", // id
      "isChild": "boolean",
      "childId": "string", // can be null or empty
      "isParent": "boolean",

   }
}
```

Question 3:

To get child, get object where isChild is true.

To get relationship with parent: get childId from churchMember where isParent is true. Then look up childId.

Isn't everyone a child???

Question 4:

Here one can count the number of churchMembers where isParent field is true.

Question 5:

Get church members where isParent is true and marriedTo is empty or null (single parents).

➔ For each church member childId, check that age is under 10.