Clarification of requirements is need

- What will this endpoint be used for?
- If there's any specific specification related to the endpoint, please provide it for further clarification or to address any specific requirements.
- What data format will be used in the API?
- What is the data structure in the database?
- How is the design specified for 'Coordinates and/or dimensions of the rectangle according to your choice of design'? How can we choose the operating mode based on coordinates or area calculation?
- Should we consider the shape's size, and in what cases should we return size data?
- Implementation implies one endpoint for adding a record and a separate one for calculating inclusion, or is it one endpoint with different methods? (example, POST some rectangle's and GET rectangles by point)
- Are we using the PATCH, PUT, or DELETE methods for this endpoint to update or remove records?
- If this functionality involves coordinates on the Earth's surface, do we need to know in which coordinate system we are working or supporting?
- Are we considering an inclusion if a point falls on the rectangle's boundaries?
- What behavior do we expect when passing coordinates for a shape that is not a rectangle? Should validation be implemented?
- What behavior do we expect when not passing coordinates (what coordinate format is supported, and what is the maximum fractional part)?
- In what format should the response about inclusion in rectangles be provided?
- What response do we expect if no inclusions are found?
- If the database contains shapes that are not rectangles, will validation break backward compatibility?
- Should only one coordinate be passed in a single request, or is passing an array of coordinates also supported?
- Is a duplicated rectangle allowed?
- For creating a rectangle, a minimum of 2 points is required. Does the system operate with 2 points / 3 points / 4 points of coordinates?
- Is the coordinate system 2D or 3D…nD?

I wouldn't ask developers to create some records in DB because according to the task I could create some rectangles using API by myself.
To correctly prepare test data and avoid mistakes, I would use programs with a graphical interface where I could visually display points based on coordinates and rectangles. Then, I would simply transfer the test rectangles to DB.
If I were working with geographic coordinates, I would use geospatial databases to verify the correctness of point inclusion.
For automation I will use Java(RestAssured, TestNG)
It is impossible to create API test cases without documentation or working product, that is why the checklist was prepared.

- [ ] Send some point that has inclusion to one rectangle
- [ ] Send some point that hasn’t inclusion to any rectangle
- [ ] Check behavior when for coordinates no inclusion to rectangle found
- [ ] Check behavior when large number of rectangle were found for test point
- [ ] Check is real all rectangles were returned correctly
- [ ] Send some point that has long floating part/long value
- [ ] Send some point with empty string coordinates (“”, “”, ””, ””)
- [ ] Send some point with x or y has wrong format
- [ ] Check behavior when if coordinates intersections on rectangle’s borders
- [ ] Check correct/incorrect calculation of square(with with coordinates below zero included)
- [ ] Check behavior if square of rectangle is too long
- [ ] Create one rectangle and then create such one(duplicate)
- [ ] Try to create non-rectangle
- [ ] Create one rectangle with coordinates below zero
- [ ] Create one rectangle with coordinates with all zero coordinates
- [ ] Create one rectangle with the same coordinates ({x,y}, {x,y}, {x,y}, {x,y})
- [ ] Create rectangle by 2 points and dimensions
- [ ] Create a rectangle only by dimensions -> ?
- [ ] Create a rectangle by zero dimensions -> ?
- [ ] Try to create rectangle empty coordinates ({null, null}, {null, null},{null, null},{null, null})
- [ ] Try to create rectangle empty string coordinates (“”, “”, ””, ””)
- [ ] Try to create rectangle with 3 correct and one coordinates with wrong format(if more then 2 is allowed)
- [ ] Try to create rectangle with large number of floating part
- [ ] Send some SQL injection
- [ ] Create some load test(measure time to answer under large number of requests
- [ ] Check response structure/data format according specification

Testcase should by something like:

id	Name	requirement(optional)	testdata	Steps	Expected Result
1.	GET intersection rectangle (positive)	R1	{x,y}	curl --location --globoff 'https://test.com/json?p1={coorX1}{coorY1}&p2={coorX2}{	{ "type": "FeatureCollection", "features": [{

				coorY2}&key=YOUR_API_KEY' \
				--header 'accept: application/json' \
				--header 'Content-Type: application/x-www-form-urlencoded'
					"type": "Feature",
					"properties": {},
					"geometry": {
					"type": "rectangle",
					"coordinates": [
					[
					[
					x1,
					y1
					],
					[
					x2,
					y2
					],
					[
					x3,
					y3
					],
					[
					x4,
					y4
					],
					]
					]
					}
					}
					]
					}