Exam Instructions

- All electronic devices, except for laptops, are not allowed during the exam.
- The exam is open book, which means you can use any online resources but you cannot seek help from anyone or use AI engines.
- The exam will last for two hours.
- In case you encounter incomplete or vague questions, make assumptions and answer to the best of your knowledge.
- No questions are allowed during the exam.
- Please enter the last five digits of your BCIT student ID in the "Student ZipGrade ID" section of the answer sheet. For instance, if your ID is "A09912345", you should fill in "12345".

Q1.

All the following HTTP methods are not idempotent except:

A. GET
B. POST
C. PATCH
D. DELETE

Q2.

Which of the following is NOT a benefit of using API servers?

A. Improved security
B. Better scalability
C. Faster data processing
D. Simplified user interface design

Q3.

Which of the following options is false for the following code.

```
const myFirstPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Success!"); // Yay! Everything went well!
  }, 250);
});

myFirstPromise.then((successMessage) => {
  console.log(`Yay! ${successMessage}`);
});
```

A. The given code creates a Promise that resolves with the string "Success!" after a 250 millisecond delay. It then logs "Yay! Success!" to the console using a .then() callback function when the Promise is *fulfilled*.

B. The given code creates a Promise that resolves with the string "Success!" after a 250 millisecond delay. It then logs "Yay! Success!" to the console using a .then() callback function when the Promise is *resolved*.

C. The given code creates a Promise that resolves with the string "Success!" after a 250 millisecond delay. It then logs "Yay! Success!" to the console using a .then() callback function when the Promise is *rejected*.

D. The following code has the same output as the given code.

```
const myFirstPromise = () => {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve("Success!"); // Yay! Everything went well!
    }, 250);
  });
};

(async () => {
  const successMessage = await myFirstPromise();
    console.log(`Yay! ${successMessage}`);
})();
```

Q4.

What is the output of the following code

```
function doStep1(init) {
  return new Promise((resolve) => {
    const result = init + 10;
    resolve(result);
  });
}

function doStep2(init) {
  return new Promise((resolve) => {
    const result = init + 2;
    resolve(result);
  });
}

function doStep3(init) {
  return new Promise((resolve) => {
    const result = init + 3;
    resolve(result);
  });
}
```

```
async function doOperation() {
  try {
    const result1 = await doStep1(0);
    const result2 = await doStep2(result1);
    const result3 = await doStep3(result2);
    console.log(`result: ${result3}`);
  } catch (error) {
    console.error(`Error: ${error}`);
  }
}

doOperation();
```

A. result: 15
B. result: 10
C. result: 3
D. None of the above

Q5.

All the following are false except:
A. Promises are used to handle synchronous operations in JavaScript.
B. Async/await is a way to handle asynchronous operations in JavaScript using callback functions.
C. Promisify is a built-in JavaScript function used to convert asynchronous functions with callbacks into promises.
D. Async/await is only used in Node.js, not in browser-based JavaScript.

For the following Express server, answer the questions from 6-10

```
const express = require('express');
const app = express();

// Bird data array
const birds = [
  { id: 1, name: 'Eagle', species: 'Aquila chrysaetos', color: 'Brown' },
  { id: 2, name: 'Owl', species: 'Strigiformes', color: 'Varies' },
  { id: 3, name: 'Hummingbird', species: 'Trochilidae', color: 'Various colors' },
  // Add more bird data objects as needed
];

// Middleware to parse request body as JSON
app.use(express.json());

// GET endpoint to get all birds
app.get('/birds', (req, res) => {
  res.json(birds);
```

```
});

// GET endpoint to get bird by ID
app.get('/birds/:id', (req, res) => {
  const birdId = Number(req.params.id);
  const bird = birds.find(bird => bird.id === birdId);

  if (!bird) {
    res.status(404).json({ message: 'Bird not found' });
  } else {
    res.json(bird);
  }
});

// POST endpoint to add a new bird
app.post('/birds', (req, res) => {
  const newBird = req.body;
  birds.push(newBird);
  res.status(201).json(newBird);
});

// PUT endpoint to update a bird by ID
app.put('/birds/:id', (req, res) => {
  const birdId = Number(req.params.id);
  const updatedBird = req.body;
  const index = birds.findIndex(bird => bird.id === birdId);

  if (index === -1) {
    res.status(404).json({ message: 'Bird not found' });
  } else {
    birds[index] = { ...birds[index], ...updatedBird };
    res.json(birds[index]);
  }
});

// DELETE endpoint to remove a bird by ID
app.delete('/birds/:id', (req, res) => {
  const birdId = Number(req.params.id);
  const index = birds.findIndex(bird => bird.id === birdId);

  if (index === -1) {
    res.status(404).json({ message: 'Bird not found' });
  } else {
    const removedBird = birds.splice(index, 1);
    res.json(removedBird[0]);
  }
});

// Start the server
const port = 3000;
app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```

Q6.

Which of the following is *false* regarding the `app.put` route?

A. It updates the bird with the given id extracted from the url params.
B. It replaces the whole bird object with the passed object in the request body.
C. It updates the bird object only with the properties passed in the request body.
D. It returns the updated bird object.

Q7.

Which of the following axios requests is a proper test to the POST `/birds` endpoint of the Express API server:

A.

```
const axios = require('axios');

// Bird data to be added
const newBird = {
  id: 4,
  name: 'Penguin',
  species: 'Aptenodytes',
  color: 'Black and White'
  // Add more properties as needed
};

// POST request to add a new bird
axios.post('http://localhost:3000/bird', newBird)
  .then(response => {
    console.log('New bird added:', response.data);
  })
  .catch(error => {
    console.error('Error adding bird:', error.message);
  });
```

B.

```
const axios = require('axios');


// POST request to add a new bird
await axios.post('http://localhost:3000/birds', "newBird")
```

C.

```
const axios = require('axios');

// Bird data to be added
const newBird = {
  id: 4,
  name: 'Penguin',
  species: 'Aptenodytes',
  color: 'Black and White'
  // Add more properties as needed
};

// POST request to add a new bird
axios.post('http://localhost:3000/birds', newBird)
  .then(response => {
    console.log('New bird added:', response.data);
  })
  .catch(error => {
    console.error('Error adding bird:', error.message);
  });
```

D.

```
const axios = require('axios');

// Bird data to be added
const newBird = [
  id: 4,
  name: 'Penguin',
  species: 'Aptenodytes',
  color: 'Black and White'
  // Add more properties as needed
];

// POST request to add a new bird
axios.post('http://localhost:3000/birds', newBird)
  .then(response => {
    console.log('New bird added:', response.data);
  })
  .catch(error => {
    console.error('Error adding bird:', error.message);
  });
```

Q8.

For the following custom error handler, which of the following options is wrong?

```
// Assuming you have Express installed and required it in your code
const express = require('express');

// Create an Express app
const app = express();

// Route handler
app.get('/', (req, res) => {
  // Simulating an error by trying to access a property of an undefined object
  const undefinedObject = undefined;
  const property = undefinedObject.property; // This will throw an error

  // This line will not be executed due to the error above
  res.send('Hello World!');
});

// error handler
app.use((err, req, res, next) => {
  // Log the error
  console.error(err);

  // Send an error response to the client
  res.status(500).json({ error: 'Internal Server Error' });
});

// Start the Express app
app.listen(3000, () => {
  console.log('App is running on http://localhost:3000');
});
```

A. The error handler will pass the error to the next error handler in the middleware stack.
B. it is a middleware function that will be called when an error is thrown in the route handler.
C. The error handler will log the error and send a 500 response to the client.
D. The error handler will catch all errors thrown in the route handler including asynchronous errors.

Q9.

Which of the comments within the following code is incorrect?

```
// Assuming you have Express and the 'fs' module installed and required in your
code
const express = require('express');
const fs = require('fs');

// Create an Express app
const app = express();

// Route handler
app.get('/', async (req, res, next) => {
```

```
  try {
    // Simulating an asynchronous error by reading a non-existent file
    await fs.promises.readFile('nonexistent-file.txt');

    // This line will be executed despite the error above
    res.send('Hello World!');
  } catch (err) {
    // Pass the error to the next middleware
    next(err);
  }
});

// error handler
app.use((err, req, res, next) => {
  // Log the error
  console.error(err);

  // Send an error response to the client
  res.status(500).json({ error: 'Internal Server Error' });
});

// Start the Express app
app.listen(3000, () => {
  console.log('App is running on http://localhost:3000');
});
```

A. `// Pass the error to the next middleware`
B. `// This line will be executed despite the error above`
C. `// Send an error response to the client`
D. `// Default error handler`

Q10.

What is the output of the following code?

```
const express = require('express');
const app = express();



// Local Middleware
const authenticate = (req, res, next) => {
  // Perform authentication logic here
  if (req.headers.authorization === 'my-secret-token') {
    next(); // If authenticated, pass control to the next middleware or route
handler
  } else {
    res.status(401).send('Unauthorized'); // If not authenticated, send 401
Unauthorized response
```

```
  }
};

// Route with local and global middleware
app.get('/protected', authenticate, (req, res) => {
  console.log('Local Middleware executed.'); // Local middleware
  res.send('This is a protected route.');
});

// Global Middleware
app.use((req, res, next) => {
  console.log('Global Middleware executed.'); // Global middleware
  next(); // Pass control to the next middleware or route handler
});

// Route without middleware
app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

if it was tested with this axios request

```
const axios = require('axios');



// POST request to add a new bird
axios.get('http://localhost:3000/protected', {
  headers: {
    authorization: 'my-secret-token'
  }
})
  .then(response => {
    console.log(response.data);
  })
  .catch(error => {
    console.error('Error', error.message);
  });
```

A.

```
Local Middleware executed.
Global Middleware executed.
```

B.

```
Local Middleware executed.
```

C.

```
Global Middleware executed.
Local Middleware executed.
```

D.

```
Global Middleware executed.
```

Q11.

What is the value of `x` and `y` after executing the following code?

```
const numbers = [1, 2, 3, 4, 5];

const x = numbers.some(num => num > 3);

console.log(x);

const y = numbers.every(num => num > 0);

console.log(y);
```

A. `x` is `true` and `y` is `false`
B. `x` is `false` and `y` is `true`
C. `x` is `true` and `y` is `true`
D. `x` is `false` and `y` is `false`

Q12.

What is the output of the following code?

```
const fruits = ['apple', 'banana', 'orange', 'grape'];


const hasOrangeFromIndexOrPosition = fruits.includes('orange', 2);
```

```
console.log(hasOrangeFromIndexOrPosition);
```

A. `true`. The code checks if 'orange' is present in the fruits array, starting from index 2 using includes()
B. `true`. The code checks if 'orange' is present in the fruits array, starting from index 1 using includes()
C. `false`. The code checks if 'orange' is present in the fruits array, starting from index 1 using includes()
D. `false`. The code checks if 'orange' is present in the fruits array, starting from index 2 using includes()

Q13.

What is the output of the following code?

```
const fruits = ['apple', 'banana', 'orange', 'grape'];

const x = fruits.slice(-2);

console.log(x);
```

A. `[undefined, undefined]`
B. `['banana', 'orange']`
C. `['orange', 'grape']`
D. `['apple', 'banana']`

Q14.

For the following JSON object, How do you access the name of the first pet in JS?

```
[{
  "name": "Alice",
  "age": 32,
  "gender": "female",
  "hobbies": ["reading", "hiking", "cooking"],
  "pets": [
    {
      "name": "Buddy",
      "species": "Dog",
      "age": 3
    },
    {
      "name": "Whiskers",
      "species": "Cat",
      "age": 5
    }
  ],
  "languages": ["English", "French", "Spanish"]
```

```
}
]
```

A.

```
const firstPetName = jsonObject.pets.name;
console.log(firstPetName);
```

B.

```
const firstPetName = jsonObject.pets[0].name;
console.log(firstPetName);
```

C.

```
const firstPetName = jsonObject.pets[1].name;
console.log(firstPetName);
```

D.

```
const firstPetName = jsonObject.pets.name[0];
console.log(firstPetName);
```

Q15.

For the following JSON object, all the options below are valid options to create a new array containing the names of all the animals in the zoo except which option?

```
{
  "name": "Zoo",
  "location": "San Francisco",
  "opening_hours": {
    "weekday": "9:00 AM - 5:00 PM",
    "weekend": "10:00 AM - 6:00 PM"
  },
  "animals": [
    {
      "name": "Lion",
      "species": "Panthera leo",
      "age": 5,
      "habitat": "African savannah",
```

```
      "diet": ["meat", "wild game"]
    },
    {
      "name": "Penguin",
      "species": "Aptenodytes forsteri",
      "age": 2,
      "habitat": "Antarctica",
      "diet": ["fish", "krill"]
    },
    {
      "name": "Elephant",
      "species": "Loxodonta africana",
      "age": 10,
      "habitat": "African grasslands",
      "diet": ["grass", "leaves", "fruits"]
    }
  ],
  "popular_exhibits": ["Big Cats", "Penguin Point", "African Savanna"],
  "ticket_price": {
    "weekday_adult": "$15",
    "weekday_child": "$10",
    "weekend_adult": "$18",
    "weekend_child": "$12"
  }
}
```

A.

```
const zoo = {...}; // JSON object representing the zoo
const animalNames = zoo.animals.map(animal => animal.name);
```

B.

```
const zoo = {...}; // JSON object representing the zoo
const animalNames = zoo.animals.map(animal => {
  return animal.name;
});
```

C.

```
const zoo = {...}; // JSON object representing the zoo
const animalNames = zoo.animals.map(function(animal) {
  return animal.name;
});
```

D.

```
const zoo = {...}; // JSON object representing the zoo
const animalNames = zoo.map(animal => animal.name);
```

Q16.

For the following object, how can you decrement the ages of animals in the `zoo` object by one?

```
const zoo = {
  "name": "My Zoo",
  "location": "City X",
  "animals": [
    {
      "name": "Lion",
      "species": "Panthera leo",
      "age": 5
    },
    {
      "name": "Tiger",
      "species": "Panthera tigris",
      "age": 3
    },
    {
      "name": "Bear",
      "species": "Ursus arctos",
      "age": 7
    }
  ]
};
```

A.

```
const decrementedAges = zoo.animals.map(animal => {
  animal.age--;
});
```

B.

```
const decrementedAges = zoo.animals.map(animal => {
  return { ...animal, age: animal.age-- };
});
```

C.

```
const decrementedAges = zoo.animals.map(animal => {
  return { ...animal, age: animal.age - 1 };
});
```

D.

```
const decrementedAges = zoo.animals.map(animal => {
  animal.age - 1;
});
```

Q17.

For all the following options, the following code snippets will decrement the ages of animals in the `zoo` object by one except:

```
const zoo = {
  "name": "My Zoo",
  "location": "City X",
  "animals": [
    {
      "name": "Lion",
      "species": "Panthera leo",
      "age": 5
    },
    {
      "name": "Tiger",
      "species": "Panthera tigris",
      "age": 3
    },
    {
      "name": "Bear",
      "species": "Ursus arctos",
      "age": 7
    }
  ]
};
```

A.

```
for (let i = 0; i < zoo.animals.length; i++) {
  zoo.animals[i].age = zoo.animals[i].age - 1;
}
```

B.

```
zoo.animals.forEach(animal => {
  return { ...animal, age: animal.age - 1 };
});
```

C.

```
const decrementedAges = zoo.animals.map(animal => {
  animal.age = animal.age - 1;
  return animal;
});
```

D.

```
const zoo.animals = zoo.animals.map(animal => {
  return { ...animal, age: animal.age - 1 };
});
```

Q18.

For the following collection, which of the following queries would return the name of the person, the age of the person, the name of the first pet, and the name of the second pet?

```
[{
  "name": "Alice",
  "age": 32,
  "gender": "female",
  "hobbies": ["reading", "hiking", "cooking"],
  "pets": [
    {
      "name": "Buddy",
      "species": "Dog",
      "age": 3
    },
    {
```

```
      "name": "Whiskers",
      "species": "Cat",
      "age": 5
    }
  ],
  "languages": ["English", "French", "Spanish"]
}
]
```

A.

```
db.your_collection_name.aggregate([
  {
    $project: {
      person_name: "$name",
      person_age: "$age",
      first_pet_name: { $arrayElemAt: ["$pets.name", 0] },
      second_pet_name: { $arrayElemAt: ["$pets.name", 1] }
    }
  }
])
```

B.

```
db.your_collection_name.aggregate([
  {
    $project: {
      person_name: "$name",
      person_age: "$age",
      first_pet_name: { $first: "$pets.name" },
      second_pet_name: { $last: "$pets.name" }
    }
  }
])
```

C.

```
db.your_collection_name.aggregate([
  {
    $project: {
      person_name: "$name",
      person_age: "$age",
      first_pet_name: "$pets.name[0]",
      second_pet_name: "$pets.name[1]"
    }
  }
])
```

D.

```
db.your_collection_name.aggregate([
  {
    $project: {
      person_name: "$name",
      person_age: "$age",
      first_pet_name: { $arrayElemAt: ["$pets.name", 0] },
      second_pet_name: { $arrayElemAt: ["$pets.name", 1] }
    }
  }
])
```

E.

```
db.your_collection_name.aggregate([
  {
    $arrayElemAt: ["$name", 0]
  }
])
```

Q19.

Which of the following statements is true about JSON Web Tokens (JWTs)?

A. JWTs are stored on the client-side as small text files
B. JWTs can not be used for server-side session management
C. JWTs contain encoded data in JSON format and are used for authentication and authorization
D. JWTs are automatically cleared when a user closes their browse

Q20.

What is the primary advantage of using JWTs over cookies for authentication in a distributed system?

A. JWTs allow for stateless authentication, reducing the need for server-side storage
B. JWTs can be used for long-term session management
C. JWTs provide enhanced security compared to cookies
D. JWTs allow for centralized session management across multiple servers

Q21.

In a web application, if a user disables cookies in their browser, how would it affect the functionality of the application?

A. The application would not be able to function at all
B. The application would still function but may have limited functionality
C. The application would function normally as cookies are not required for web applications
D. The application would automatically switch to using sessions or JWTs for data storage.

Use the following `weather` collection for the questions 22-28.


```
[
  {
    "city": "New York",
    "country": "United States",
    "temperature": 22.5,
    "humidity": 60,
    "precipitation": 0.1,
    "windSpeed": 8,
    "conditions": ["sunny", "partly cloudy"],
    "updatedDate": "2023-04-06T10:30:00Z"
  },
  {
    "city": "London",
    "country": "United Kingdom",
    "temperature": 12.9,
    "humidity": 75,
    "precipitation": 0.3,
    "windSpeed": 12,
    "conditions": ["rainy", "windy"],
    "updatedDate": "2023-04-06T09:45:00Z"
  },
  {
    "city": "Sydney",
    "country": "Australia",
    "temperature": 26.7,
    "humidity": 50,
    "precipitation": 0,
    "windSpeed": 5,
    "conditions": ["sunny"],
    "updatedDate": "2023-04-06T08:15:00Z"
  }
]
```


Q22.

Using the MongoDB find method, retrieve all the weather data documents from the "weather" collection where the temperature is greater than or equal to 20°C and the conditions include "sunny".

A.

```
db.weather.find({ temperature: { $gte: 20 }, conditions: "sunny" })
```

B.

```
db.weather.find({ temperature: { $gte: 20 }, conditions: { $in: "sunny" } })
```

C.

```
db.weather.find({ temperature: { $gt: 20 }, conditions: "sunny" })
```

D.

```
db.weather.find({ temperature: { $gte: 20, $lt: 25 }, conditions: "sunny" })
```

Q23.

Using the MongoDB `find` function, which of the following will retrieve only the `city` and `temperature` fields for all the weather data documents from the `weather` collection where the `conditions` field is `rainy`.

A.

```
db.weather.find({ conditions: "rainy" }, { city: 1, temperature: 1 })
```

B.

```
db.weather.find({ conditions: "rainy" }, { _id: 0, city: 1, temperature: 1 })
```

C.

```
db.weather.find({ conditions: "rainy" }, { _id: 0, city: 0, temperature: 1 })
```

D.

```
db.weather.find({ conditions: "rainy" }, { city: 0, temperature: 1 })
```

Q24.

Which city will be retrieved for the following query?

```
db.weather.find({ "conditions": "sunny" }).skip(1).limit(2)
```

A. London
B. Sydney
C. New York
D. None of the above

Q25.

Which city will be retrieved for the following query?

```
db.weather.find({ "temperature": { $gt: 20 } }).sort({ "humidity": -1 }).limit(1)
```

A. London
B. Sydney
C. New York
D. None of the above

Q26.

Which cities will be retrieved for the following query?

```
db.weather.find({
  $or: [
    {
      "precipitation": {
        $lte: .1
      }
    }, {
      "country": "Australia"
    }]
})
```

A. London, Sydney
B. Sydney, New York
C. New York, London
D. None of the above

Q27.

The following query will

```
db.weather.find({ "conditions": { $all: ["sunny", "partly cloudy"] } }, { "city": 1 })
```

A. retrieve cities that have both "sunny" and "partly cloudy" conditions in their "conditions" field in the exact order

B. retrieve cities that have either "sunny" or "partly cloudy" conditions in their "conditions" field in the exact order

C. retrieve cities that have both "sunny" and "partly cloudy" conditions in their "conditions" field in any order

D. retrieve cities that have either "sunny" or "partly cloudy" conditions in their "conditions" field in any order

Q28.

Which of the following queries will produce the following output

```
  {
    "city": "London",
    "country": "United Kingdom",
    "temperature": 12.9,
    "windSpeed": 12,
    "conditions": [
      "rainy",
      "windy"
    ],
    "updatedDate": "2023-04-06T09:45:00Z"
  }
```

A.

```
db.weather.updateOne(
  { "city": "London" },
  { $remove: { "precipitation": "" } }
)

db.weather.find({ "city": "London" }, { "_id": 0})
```

B.

```
db.weather.updateOne(
  { "city": "London" },
  { $remove: { "precipitation": "" } }
)

db.weather.find({ "city": "London" }, { "_id": 0, humidity:0 })
```

C.

```
db.weather.updateOne(
  { "city": "London" },
  { $unset: { "precipitation": "" } }
)

db.weather.find({ "city": "London" }, { "_id": 0 })
```

D.

```
db.weather.updateOne(
  { "city": "London" },
  { $unset: { "precipitation": "" } }
)

db.weather.find({ "city": "London" }, { "_id": 0, humidity:0 })
```

For the following `cities` db, answer questions 29-36 using the following collection:

```
[
  {
    "city": "Toronto",
    "province": "Ontario",
    "country": "Canada",
    "population": 2731571,
    "latitude": 43.651070,
    "longitude": -79.347015,
    "area": 630.2,
    "officialLanguage": "English"
  },
  {
    "city": "Vancouver",
    "province": "British Columbia",
    "country": "Canada",
    "population": 2463431,
    "latitude": 49.282730,
    "longitude": -123.120735,
```

```
    "area": 2878.5,
    "officialLanguage": "English"
  },
  {
    "city": "Montreal",
    "province": "Quebec",
    "country": "Canada",
    "population": 1780000,
    "latitude": 45.501690,
    "longitude": -73.567253,
    "area": 365.1,
    "officialLanguage": "French"
  },
  {
    "city": "Calgary",
    "province": "Alberta",
    "country": "Canada",
    "population": 1336000,
    "latitude": 51.050110,
    "longitude": -114.085290,
    "area": 825.3,
    "officialLanguage": "English"
  },
  {
    "city": "Ottawa",
    "province": "Ontario",
    "country": "Canada",
    "population": 934243,
    "latitude": 45.421530,
    "longitude": -75.697193,
    "area": 2790.3,
    "officialLanguage": "English, French"
  }
]
```

Q29.

Which of the following schema is the most suitable match for the db

A.

```
const mongoose = require('mongoose');

const citySchema = new mongoose.Schema({
  city: { type: String, index: true }, // Index added to the 'city' field
  province: String,
  country: String,
  population: Number,
  latitude: Number,
  longitude: Number,
```

```
  area: Number,
  officialLanguage: String
});

citySchema.index({ country: 1 }); // Index added to the 'country' field

const City = mongoose.model('City', citySchema);

module.exports = City;
```

B.

```
const mongoose = require('mongoose');

const citySchema = new mongoose.Schema({
  city: { type: String, required: true  },
  province: { type: String, required: true },
  country: { type: String, required: true },
  population: { type: Number, required: true },
  latitude: { type: Number, required: true },
  longitude: { type: Number, required: true },
  area: { type: Number, required: true },
  officialLanguage: { type: String, required: true }
});

const City = mongoose.model('City', citySchema);

module.exports = City;
```

C.

```
const mongoose = require('mongoose');

const citySchema = new mongoose.Schema({
  city: { type: String, required: true , index: true },
  province: { type: String, required: true },
  country: { type: String, required: true },
  population: { type: Number, required: true },
  latitude: { type: Number, required: true },
  longitude: { type: Number, required: true },
  area: { type: Number, required: true },
  officialLanguage: { type: String, required: true }
});

const City = mongoose.model('City', citySchema);
```

```
module.exports = City;
```

D.

```
const mongoose = require('mongoose');

const citySchema = new mongoose.Schema({
  city: 'String',
  province: 'String',
  country: 'String',
  population: 'Number',
  latitude: 'Number',
  longitude: 'Number',
  area: 'Number',
  officialLanguage: 'String'
});

const City = mongoose.model('City', citySchema);

module.exports = City;
```

Q30.

What is the output of the following query?

```
const City = require('./city-model'); // Assuming the City model is defined in a
separate file

  const cities = await City.findOne({
    population: { $gt: 2000000 }
  }, {
    city: 1,
    _id: 0
  });
  console.log(cities);
```

A. `{city: 'Toronto' }`
B. `[ { city: 'Toronto' }, { city: 'Vancouver' } ]`
C. `[]`
D. `[ { city: 'Toronto' } ]`

Q31.

What is the output of the following query?

```
const City = require('./city-model'); // Assuming the City model is defined in a
separate file

   const cities = await City.find({
    $and: [
      { population: { $gt: 1000000 } },
      {
        $or: [
          { province: 'Ontario' },
          { province: 'Quebec' }
        ]
      },
      {
        $or: [
          { officialLanguage: 'English' },
          { officialLanguage: 'French' }
        ]
      }
    ]
  }, {
    city: 1,
    _id: 0

  });

  console.log(cities);
```

A. `[ { city: 'Toronto' }, { city: 'Montreal' } ]`
B. `[ { city: 'Toronto' }, { city: 'Vancouver' } ]`
C. `[ { city: 'Montreal' }]`
D. `[ { city: 'Toronto' } ]`

Q32.

Which of the following queries will group cities by `province` and calculate the total population for each province:

A.

```
City.aggregate([
  { $group: { _id: '$city', totalPopulation: { $sum: '$population' } } },
  { $project: { _id: 0, province: '$_id', totalPopulation: 1 } }
])
```

B.

```
 City.aggregate([
  { $group: { _id: '$province', totalPopulation: { $avg: '$population' } } },
  { $project: { _id: 0, province: '$_id', totalPopulation: 1 } }
]);
```

C.

```
City.aggregate([
  { $group: { _id: '$province', totalPopulation: { $sum: '$population' } } },
  { $project: { _id: 0, province: '$_id', totalPopulation: 1 } }
]);
```

D.

```
City.aggregate([
  { $match: { _id: '$province', totalPopulation: { $sum: '$population' } } },
  { $project: { _id: 0, province: '$_id', totalPopulation: 1 } }
]);
```

Q33.

What does the following query output?

```
 await City.aggregate([
    { $group: { _id: '$province', avgArea: { $avg: '$area' }, totalPopulation: {
$sum: '$population' } } }, // Group by province and calculate avg area and total
population
    { $sort: { _id: 1 } }, // Sort by province name in ascending order
    { $project: { _id: 0, province: '$_id', avgArea: 1, totalPopulation: 1 } } //
Project to include province, avg area, and total population fields
  ]);
```

A. The query calculates the average area and total population for cities in each province, and sorts the result by province name in ascending order.

B. The query calculates the total area and total population for cities in each province, and sorts the result by province name in ascending order.

C. The query calculates the average area and total population for cities in each province, and sorts the result by province name in descending order.

D. The query calculates the total area and total population for cities in each province, and sorts the result by province name in descending order.

Q34.

Which of the following queries will produce the following output ?

```
[ { city: 'Vancouver', province: 'British Columbia' } ]
```

A.

```
await City.find()
    .where('population').gt(2000000)
    .select('city province -_id')
    .limit(1)
    .sort('city')
    ;
```

B.

```
await City.find()
    .where('population').gt(2000000)
    .select('city province -_id')
    .limit(1)
    .sort('-city')
    ;
```

C.

```
await City.find()
    .where('population').gt(2000000)
    .select('city province')
    .deselect('_id')
    .limit(1)
    .sort('-city')
    ;
```

D.

```
await City.find()
    .where('population').gt(2000000)
    .select('city province')
```

```
    .deselect('_id')
    .limit(1)
    .sort('city')
    ;
```

Q35.

What is the output of the `console.log` in the following code?

```
const result = await City.findOneAndUpdate(
  { 'city': 'Toronto' },
  { 'population': 2800000 }
);
console.log(result);
```

A.

```
{
  "acknowledged": true,
  "matchedCount": 1,
  "modifiedCount": 1,
  "upsertedId": null
}
```

B.

```
{
  "acknowledged": true,
  "matchedCount": 1,
  "modifiedCount": 0,
  "upsertedId": null
}
```

C.

```
{
  _id: new ObjectId("642f2fed32cd06eeca164b26"),
  city: 'Toronto',
  province: 'Ontario',
  country: 'Canada',
  population: 2731571,
  latitude: 43.65107,
  longitude: -79.347015,
  area: 630.2,
```

```
  officialLanguage: 'English'
}
```

D.

```
{
  _id: new ObjectId("642f2fed32cd06eeca164b26"),
  city: 'Toronto',
  province: 'Ontario',
  country: 'Canada',
  population: 2800000,
  latitude: 43.65107,
  longitude: -79.347015,
  area: 630.2,
  officialLanguage: 'English'
}
```

Q36.

What is the output of the `console.log` in the following code?

```
  await City.findOneAndReplace(
    { 'city': 'Toronto' },
    { 'population': 2800000 },

  );

  const result = await City.find({
    'city': 'Toronto'
  })
  console.log(result);
```

A.

```
[
  {
    _id: new ObjectId("642f2fed32cd06eeca164b26"),
    city: 'Toronto',
    province: 'Ontario',
    country: 'Canada',
    population: 2800000,
    latitude: 43.65107,
    longitude: -79.347015,
    area: 630.2,
```

```
    officialLanguage: 'English'
  }
]
```

B.

```
[]
```

C.

```
[
  {
    _id: new ObjectId("642f2fed32cd06eeca164b26"),
    city: 'Toronto',
    province: 'Ontario',
    country: 'Canada',
    population: 2731571,
    latitude: 43.65107,
    longitude: -79.347015,
    area: 630.2,
    officialLanguage: 'English'
  }
]
```

D.

```
[
  {
    _id: new ObjectId("642f2fed32cd06eeca164b26"),
    city: 'Toronto',
    population: 2800000,
  }
]
```

Q37.

Which of the comments in code below is a false statement?

```
import React, { useState, useEffect } from "react";

const ExampleComponent = () => {
  const [count, setCount] = useState(0);
```

```
  useEffect(() => {
    console.log("Component mounted!"); // This will run on component mount


  }, []); // Empty dependency array means this effect will never run

  useEffect(() => {
    console.log(`Count changed: ${count}`); // This will run whenever count
changes
  }, [count]); // Dependency array with count means this effect only runs when
count changes

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};

export default ExampleComponent;
```

A. `// Dependency array with count means this effect only runs when count changes`
B. `// This will run whenever count changes`
C. `// Empty dependency array means this effect will never run`
D. `// This will run on component mount`

Q38.

What is the purpose of the render method in a React component?
A. To define the initial state of a component
B. To update the state of a component
C. To specify the layout and structure of a component's UI
D. To perform side effects or data fetching

Q39.

In React, what is the purpose of the useEffect hook with an empty dependency array `([])`?
A. To handle form submissions
B. To perform side effects or data fetching only once, when the component mounts
C. To perform side effects or data fetching whenever the component updates
D. To perform side effects or data fetching when a specific dependency changes

Q40.

The output of the following code is:

```
const regex = /ab+c/;

console.log(regex.test('ac')); // false
console.log(regex.test('abc')); // true
console.log(regex.test('abbbc')); // true
console.log(regex.test('a')); // false
console.log(regex.test('d')); // false
```

A. `true, true, true, false, false`
B. `false, true, true, false, false`
C. `false, true, true, false, true`
D. `true, true, true, false, true`