**Server.js**

```
const mongoose = require('mongoose');
const dotenv = require('dotenv');
dotenv.config({ path: './.env' });

const app = require('./app');

const DB = process.env.MONGODB_URI;

mongoose
  .connect(DB, {
    useNewUrlParser: true,
    useCreateIndex: true,
    useFindAndModify: false,
    useUnifiedTopology: true,
  })
  .then(() => console.log('DB connection successful!'));

const port = process.env.PORT || 3000;
app.listen(port, () => {
  console.log(`App running on port ${port}...`);
});
```

`server.js` dosyası içinde `express.js` kütüphanesini barındıran `app` nesnesi ve **MongoDB** veritabanı ile çalışan mongoose kütüphanesini barındıran `mongoose` nesnesi ayarlanır. `mongoose.connect()` metodu **MongoDB** veritabanına bağlantıyı sağlar. `app.listen()` metoduyla `server.js` çalışmaya başlayarak kullanıcı talepleri dinlenir.

**app.js**

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

//controllers imports
const tourRouter = require('./routes/tourRoutes');

const app = express();

// Necessary express middlewares
app.use(express.json());
app.use(express.static(`${__dirname}/public`));

// ROUTES MIDDLEWARES
// Those routers contain controllers to check user requests
app.use('/api/v1/tours', tourRouter);

module.exports = app;
```

`app.js` dosyası içinde bulunan `app.use('/api/v1/tours', tourRouter);` “/api/v1/tours” url’si üzerinden kullanıcı taleplerini **tourRouters** nesnesine yönlendirir.

**tourRoutes.js**

```
const express = require('express');
const tourController = require('./../controllers/tourController');

const router = express.Router();

router
  .route('/top-5-cheap')
  .get(tourController.aliasTopTours, tourController.getAllTours);

// Aggregation
router.route('/tour-stats').get(tourController.getTourStats);
router.route('/monthly-plan/:year').get(tourController.getMonthlyPlan);

router
  .route('/')
  .get(tourController.getAllTours)
  .post(tourController.createTour);
router
  .route('/:id')
  .get(tourController.getTour)
  .patch(tourController.updateTour)
  .delete(tourController.deleteTour);

module.exports = router;
```


Kullanıcılar web tarayıcıların url alanlarına üstte bulunan `.../api/v1/tour/:id` gibi değerler girer. Bu değerler `tourRoute.js` dosyası üzerinden yönlendirilir(`routing`).
`express.js` kütüphanesinden örneklenen `router` nesnesinin metodlarına inceleyelim. `.route()` metodu aldığı `string` argümanı ile kullanıcı girişlerinin yönlendirmesini yapar. `.get(), .post(), .patch(), .delete()` metodları argüman olarak aldığı kontrol fonksiyonları ile veri kaynadığı üzerinde işlem yaparlar. `.get()` veri almaya, `.post()` veri oluşturmaya, `.patch()` veri güncellemeye ve `.delete()` veri silmeye işlemlerini yapar.

Örnek bir işlemi inceleyelim; kullanıcı `.../api/v1/tours` url'sini girdiğinde
`app.js => içindeki app.use('/api/v1/tours', tourRouter); işletilir buradan sonra, tourRoute.js=> içindeki router.route('/').get(tourController.getAllTours) işletilir ve tüm tour değerleri kullanıcıya listelenir.`

**tourController.js**

```
const Tour = require('./../models/tourModel');
const APIFeatures = require('./../utils/apiFeatures');

exports.aliasTopTours = (req, res, next) => {
  req.query.limit = '5';
  req.query.sort = '-ratingsAverage,price';
  req.query.fields = 'name,price,ratingsAverage,summary,difficulty';
  next();
};

exports.getAllTours = async (req, res) => {
  try {
    // EXECUTE QUERY
    const features = new APIFeatures(Tour.find(), req.query)
      .filter()
      .sort()
      .limitFields()
      .paginate();
    const tours = await features.query;

    // SEND RESPONSE
    res.status(200).json({
      status: 'success',
      results: tours.length,
      data: {
        tours,
      },
    });
  } catch (err) {...}
};
exports.getTour = async (req, res) => {...};
exports.createTour = async (req, res) => {...};
exports.updateTour = async (req, res) => {...};
exports.deleteTour = async (req, res) => {...};
exports.getTourStats = async (req, res) => {...};
exports.getMonthlyPlan = async (req, res) => {...};
```

`tourController.js` dosyası içinde bulunan fonksiyonlar mongoose.js ORM(object relational model) kütüphanesi tarafından oluşturulan Tour model nesnesini kullanarak veri MongoDB veritabanı üzerinde işlemler yapar.
`getAllTours()` metodu tüm tour verilerini kullanıcının girdiği filtre değerlerine göre listeler. Eğer kullanıcı hiç bir filtre değeri girmediyse doğrudan `.../api/v1/tours` şeklinde bir giriş yaptıysa tüm tur değerleri listelenir. Eğer kullanıcı `/tours?sort=-price,ratingsAverage` soru işaretinden sonra filtre değerleri girdiyse tüm tur değerleri değil filtrelenmiş tur değerleri listelenecektir. Bu filtre işlemleri `APIFeatures(Tour.find(), req.query)` nesnesi yardımıyla değerlendirilir ve sorgu oluşturulur.

```
router
  .route('/top-5-cheap')
  .get(tourController.aliasTopTours, tourController.getAllTours);
```

`/top-5-cheap` route'unda diğer tüm `route()` metodlarından farklı bir durum var. `getAllTours()` metodu işletilmeden önce `aliasTopTours()` metodu işletiliyor. Bu metot `getAllTours()` içinde kısa yoldan bir filtre oluşturmaya yarıyor. `req.query` ile gerekli filtreler alındıktan sonra `next()` metodu işletilerek kontrol işlemi içinde bir middleware oluşturuluyor ve gerekli filtre `getAllTours()` metodu değiştirilmeden yapılabiliyor.