Appendix A: Program Source Listings

Program Source Listings

- **Title:** Program Source Code Listings

Section A.1: Frontend Code

 JobDetails.tsx
```typescript
// ... existing code ...
interface Job {
    _id: string;
    title: string;
    description: string;
    categoryId: string;
    category?: string;
    budget?: { $numberDecimal: string } | string | number | null;
    status?: 'Pending' | 'Open' | 'In Progress' | 'Completed' |
'Cancelled';
    createdAt: string;
    updatedAt?: string;
    attributes?: Record<string, string>;
    providerId?: string;
    seekerId: string;
    location?: { type: 'Point'; coordinates: [number, number]; };
    agreedAmount?: { $numberDecimal: string } | string | number | null;
    completedAt?: string;
}

const JobDetailsProviderView = () => {
    const router = useRouter();
    const searchParams = useSearchParams();
    const { getToken, userId: clerkUserId } = useAuth();
    const { user } = useUser();

    const [jobDetails, setJobDetails] = useState<Job | null>(null);
    const [loading, setLoading] = useState(true);
    const [loadingError, setLoadingError] = useState<string |
null>(null);
    const [actionError, setActionError] = useState<string | null>(null);
    const [mapCenter, setMapCenter] = useState(defaultMapCenter);

    // ... existing code ...
}
```

 ViewJobs.tsx
```typescript
// ... existing code ...
interface Job {
    _id: string;
```

```
    title: string;
    description: string;
    categoryId: string;
    category?: string;
    budget?: { $numberDecimal: string } | string | number | null;
    status?: 'In Progress' | 'Completed' | 'Cancelled';
    createdAt: string;
    updatedAt?: string;
    providerId?: string;
    seekerId: string;
    location?: { type: 'Point'; coordinates: [number, number]; };
    agreedAmount?: { $numberDecimal: string } | string | number | null;
    completedAt?: string;
    seekerInfo?: { firstName?: string; lastName?: string };
}

const ViewProviderJobs = () => {
    const [jobs, setJobs] = useState<Job[]>([]);
    const [loading, setLoading] = useState(true);
    const [loadingError, setLoadingError] = useState<string |
null>(null);
    const [activeTab, setActiveTab] = useState(0);

    // ... existing code ...
}
```

Section A.2: Backend Code

Server Entry Point (server.js)

```javascript
require("dotenv").config();
const express = require("express");
const cors = require("cors");
const mongoose = require("mongoose");
const http = require("http");
const { Server } = require("socket.io");
const { connectDB } = require("./src/config/db.js");
const routes = require("./routes/routes");
const { ClerkExpressWithAuth } = require("@clerk/clerk-sdk-node");

const app = express();
const server = http.createServer(app);

// Increase request size limits
app.use(express.json({ limit: '50mb' }));
app.use(express.urlencoded({ limit: '50mb', extended: true }));

// Body parser
app.use(express.json());

// Connect to database
connectDB();
```

```
// Mount routes
app.use("/api", routes);

// Error handling middleware
app.use((err, req, res, next) => {
    console.error(err.stack);
    res.status(500).json({ error: 'Something went wrong!' });
});

// Start server
const PORT = process.env.PORT || 5000;
server.listen(PORT, '0.0.0.0', () => {
    console.log(`ðŸš€ Server running on port ${PORT}`);
}).on('error', (err) => {
    console.error('Server error:', err);
    process.exit(1);
});
```

 Mongoose Schemas (models.js)
```javascript
const mongoose = require("mongoose");

// User Schema
const userSchema = new mongoose.Schema({
    keycloakId: { type: String, required: true, unique: true },
    firstName: { type: String, required: true },
    lastName: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    phoneNumber: { type: String, required: true },
    role: { type: String, enum: ["Seeker", "Provider"], required: true },
    location: {
        type: { type: String, enum: ["Point"], default: "Point" },
        coordinates: { type: [Number], required: true, index: "2dsphere"
}
    }
}, { timestamps: true });

// Job Schema
const jobSchema = new mongoose.Schema({
    title: { type: String, required: true },
    description: { type: String },
    categoryId: { type: mongoose.Schema.Types.ObjectId, ref: "Category",
required: true },
    seekerId: { type: mongoose.Schema.Types.ObjectId, ref: "User",
required: true },
    location: {
        type: { type: String, enum: ["Point"], default: "Point" },
        coordinates: { type: [Number], required: true, index: "2dsphere"
}
    },
    budget: { type: mongoose.Types.Decimal128, required: true },
```

```
    status: { type: String, enum: ["Open", "In Progress", "Completed",
"Cancelled"], default: "Open" },
    assignedProviderId: { type: mongoose.Schema.Types.ObjectId, ref:
"User", default: null }
}, { timestamps: true });

// Bid Schema
const bidSchema = new mongoose.Schema({
    jobId: { type: mongoose.Schema.Types.ObjectId, ref: "Job", required:
true },
    providerId: { type: mongoose.Schema.Types.ObjectId, ref: "User",
required: true },
    amountOffered: { type: mongoose.Types.Decimal128, required: true },
    message: { type: String },
    status: { type: String, enum: ["Pending", "Accepted", "Rejected"],
default: "Pending" }
}, { timestamps: true });
```

 Route Handlers (routes.js)
```javascript
// ... existing code ...

// GET all open jobs
router.get("/getOpenJobs", async (req, res) => {
    try {
        const db = getDB();
        const openJobs = await db.collection("jobs").find({ status:
"Open" }).toArray();
        res.json(openJobs);
    } catch (error) {
        console.error("Error fetching open jobs:", error);
        res.status(500).json({ error: "Failed to fetch open jobs" });
    }
});

// POST to submit a bid
router.post("/submitBid", async (req, res) => {
    try {
        const db = getDB();
        const { jobId, providerId, amount, description } = req.body;

        // Validate ObjectId format
        if (!ObjectId.isValid(jobId) || !ObjectId.isValid(providerId)) {
            return res.status(400).json({ error: "Invalid Job ID or
Provider ID format." });
        }

        const jobExists = await db.collection("jobs").findOne({ _id: new
ObjectId(jobId) });
        const providerExists = await db.collection("users").findOne({
_id: new ObjectId(providerId), userType: "Provider" });

        if (!jobExists || !providerExists) {
```

```
            return res.status(404).json({ error: "Job or Provider not
found." });
        }

        const newBid = new Bid({
            jobId: jobId,
            providerId: providerId,
            amount: parseFloat(amount),
            description: description,
        });

        const savedBid = await newBid.save();
        res.status(201).json(savedBid);
    } catch (error) {
        console.error("Error submitting bid:", error);
        res.status(400).json({ error: error.message });
    }
});
```